Pemrograman Berorientasi Objek

Irfan Subakti



Profitina
profitina.com

Pemrograman Berorientasi Objek

dengan Java dan Python

Irfan Subakti



Hak cipta © 2026 Irfan Subakti. Hak cipta dilindungi undang-undang.

Diterbitkan oleh Profitina · profitina.com

Edisi pertama, September 2026.

Setiap program dalam buku ini sudah dikompilasi dan dijalankan sebelum keluarannya dicetak. Kode sumber lengkapnya ada di folder Code yang menyertai buku ini. Program-program tersebut disediakan untuk keperluan belajar-mengajar; Irfan Subakti dan penerbit tidak bertanggung jawab atas penggunaannya di luar keperluan itu.

Daftar Isi

Kata Pengantar ..... **lvii**
Untuk siapa buku ini ditulis lvii
Mengapa dua bahasa lvii
Isi buku ini lvii
Cara membacanya lviii
Tentang penulis lviii
Ucapan terima kasih lviii

BAB 1 Objek, Program, dan Meja Kerja Pertama Anda **1**
1.1 Tempat Kita Memulai 1
1.2 Apa Itu Program 2
1.2.1 Resep untuk meja layanan 2
1.2.2 Bahasa untuk manusia, bahasa untuk mesin 3
1.2.3 Dari kode sumber menjadi program yang berjalan 4
1.2.4 Data dan operasi terhadapnya 5
1.3 Berpikir dalam Objek 5
1.3.1 Melihat-lihat di dalam kios 5
1.3.2 Keadaan, perilaku, dan identitas 6
1.3.3 Kelas: formulir kosong 6
1.3.4 Pesan: bagaimana objek-objek bekerja sama 8
1.3.5 Abstraksi: menyimpan hanya yang penting 8
1.3.6 Empat gagasan besar, sekilas 9
1.4 Dari Prosedur ke Objek 9
1.4.1 Program buku catatan 9
1.4.2 Apa yang salah 11
1.4.3 Jawaban objek 11
1.5 Mengapa Java, dan Mengapa Python 12
1.5.1 Java: sejarah singkat 12
1.5.2 Mengapa Java: portabel, didukung di mana-mana 13
1.5.3 Python: sejarah singkat, dan mengapa Python juga 13
1.5.4 Apa kata peringkat popularitas 14
1.5.5 Kedua bahasa dibandingkan 17
1.5.6 Satu cara berpikir, dua ejaan 18
1.6 Perkakas Kerja 18
1.6.1 JDK dan JRE 18
1.6.2 Di dalam JDK: folder `bin` dan PATH 19
1.6.3 Interpreter Python dan kawan-kawannya 20
1.6.4 Lingkungan pengembangan terpadu 21
1.7 Menyiapkan VS Code untuk Java dan Python di Windows, macOS, dan Linux 22
1.7.1 Windows, langkah demi langkah 23
1.7.2 macOS, langkah demi langkah 24
1.7.3 Linux, langkah demi langkah 25
1.7.4 VS Code dan kedua ekstensinya 26
1.7.5 Program Java pertama Anda 27
1.7.6 Program Python pertama Anda 28
1.7.7 *Debugging*: mengamati program berpikir 29
1.7.8 Ketika ada yang tidak beres 31
1.8 Kelas Pertama Anda: Sebuah Pesanan *Laundry* 32
1.8.1 Sebuah struk: variabel dan aritmetika 33
1.8.2 Merancang kelas `Order` 34

1.8.3 Kelas `Order` dalam Java 34
1.8.4 Kelas `Order` dalam Python 37
1.8.5 Menghidupkan pesanan-pesanan 39
1.8.6 Apa yang disimpan variabel bertipe kelas 41
1.8.7 Apa yang kita peroleh dari kelas 41
1.9 Peta Perjalanan Buku Ini 41
1.10 Profitina dalam Praktik 43
1.10.1 Studi kasus terapan: sehari di Profitina Laundry 44
1.11 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya 46
1.12 Ringkasan Bab 47
1.13 Soal Latihan 47
1.14 Penyelesaian Soal Latihan 49
Daftar Pustaka 53

BAB 2 Lingkungan Pengembangan: Menyunting, Menjalankan, dan Men-debug 55
2.1 Di Mana Kita Berhenti 55
2.2 Apa Itu IDE, dan Mengapa Anda Memerlukannya 56
2.2.1 Meja kerja, bukan kotak ajaib 56
2.2.2 Lima tugas sebuah IDE 56
2.2.3 Eclipse dan VS Code: dua meja kerja, satu perangkat gagasan 57
2.3 Bagaimana Program Berjalan: Kompilasi, Interpretasi, dan Mesin Virtual 58
2.3.1 Prosesor hanya menguasai satu bahasa 58
2.3.2 Kompilator dan interpreter, serta kapan masing-masing mengeluh 58
2.3.3 Java: javac, bytecode, dan Java Virtual Machine 59
2.3.4 Python: interpreter, kompilator tersembunyinya, dan `__pycache__` 61
2.3.5 Kapan kesalahan tertangkap: waktu kompilasi dan waktu jalan 62
2.4 Memasang dan Memulai Eclipse 63
2.4.1 Mengunduh dan memasang 63
2.4.2 *Workspace* 65
2.4.3 Tampilan 66
2.5 Meja Kerja Eclipse: *View*, Editor, dan *Perspective* 67
2.5.1 *View* dan editor 67
2.5.2 *Perspective* 68
2.5.3 *View* pada *perspective* Java 70
2.5.4 Proyek dan *nature*-nya 72
2.6 Proyek, Paket, dan Kelas Eclipse Pertama Anda 72
2.6.1 Membuat proyek 72
2.6.2 Membuat paket 73
2.6.3 Membuat kelas dan menjalankannya 74
2.6.4 Menjalankan program di luar Eclipse: berkas JAR 77
2.7 Pekerjaan yang Sama di VS Code 79
2.7.1 Folder sebagai pengganti *workspace* 79
2.7.2 Java di VS Code: *view* Java Projects 80
2.7.3 Python di VS Code: memilih interpreter 80
2.7.4 Konfigurasi peluncuran: `launch.json` 80
2.7.5 Terminal selalu ada 81
2.8 Proyek, Paket, Modul, dan *Import* 81
2.8.1 Mengapa kode perlu dikelompokkan 81
2.8.2 Paket adalah folder 82
2.8.3 Proyek dengan dua paket 82
2.8.4 *Import* yang terlupa, di kedua bahasa 86
2.8.5 Mengompilasi proyek berpaket secara manual 88
2.8.6 Mengekspor, mengimpor, dan menjelajahi proyek 88
2.9 Perkakas Andalan Penyuntingan: *Content Assist*, *Quick Fix*, *Refactoring*, dan Pemformatan 90
2.10 *Debugging* dari Nol 91

2.10.1 Apa itu bug, dan dua cara mencarinya ... 91
2.10.2 Kosakatanya ... 92
2.10.3 *Breakpoint* dan memulai *debugger* di Eclipse ... 92
2.10.4 Mengendalikan eksekusi: *resume*, *step over*, *step into*, *step out* ... 95
2.10.5 Tumpukan pemanggilan ... 96
2.10.6 Melihat variabel, dan mengubahnya ... 97
2.10.7 Mengelola *breakpoint*: menonaktifkan, melewati, *hit count*, dan kondisi ... 99
2.10.8 *Debugger* yang sama di VS Code, untuk Java dan untuk Python ... 100
2.11 Perburuan Terbimbing: Tagihan Cucian yang Terlalu Mahal ... 101
2.11.1 Gejalanya ... 101
2.11.2 Langkah 1: ketahui jawaban yang seharusnya ... 104
2.11.3 Langkah 2: susun hipotesis, lalu pasang *breakpoint* ... 104
2.11.4 Langkah 3: tur terbimbing dengan *step into*, *step over*, dan *step out* ... 104
2.11.5 Langkah 4: amati total berjalan ... 108
2.11.6 Langkah 5: perbaiki, lalu periksa ... 111
2.11.7 Metode di balik perburuan ... 114
2.12 *Breakpoint* Bersyarat: Menemukan Jarum dalam Tumpukan Jerami ... 114
2.13 Membaca *Stack Trace* ... 118
2.13.1 Membaca *stack trace* Java ... 119
2.13.2 Membaca *traceback* Python ... 120
2.14 Profitina dalam Praktik ... 120
2.15 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya ... 121
2.16 Ringkasan Bab ... 122
2.17 Soal Latihan ... 123
2.18 Penyelesaian Soal Latihan ... 124
Daftar Pustaka ... 129

BAB 3 Kelas, Objek, String, Tipe, Percabangan, dan Perulangan ... 130
3.1 Titik Berangkat Kita ... 130
3.2 Menengok ke Dalam Sebuah Kelas ... 130
3.2.1 Hello, sekali lagi ... 130
3.2.2 Membaca kata dari baris perintah ... 132
3.2.3 Nama dan konvensi ... 133
3.2.4 Komentar yang menolong pembaca ... 134
3.3 Kelas dan Objek ... 135
3.3.1 Cetak biru dan rumah ... 135
3.3.2 Anatomi sebuah kelas ... 136
3.3.3 Kata `this`, dan kata `self` ... 140
3.3.4 *Getter*, *setter*, dan `toString` ... 141
3.3.5 Membuat objek dengan `new` ... 141
3.3.6 Memeriksa bahwa kelas bekerja ... 143
3.4 Referensi dan Gambar Memori ... 146
3.4.1 Nilai dan panah ... 146
3.4.2 Python: setiap variabel adalah nama ... 149
3.4.3 Cara menggambar diagram objek ... 150
3.5 String ... 151
3.5.1 Teks adalah objek ... 151
3.5.2 Posisi dan metode sehari-hari ... 151
3.5.3 String tidak pernah berubah ... 154
3.5.4 Memformat keluaran ... 157
3.5.5 Dari teks ke bilangan, dan kembali ... 159
3.6 Tipe ... 160
3.6.1 Delapan tipe primitif Java ... 160
3.6.2 Pengetikan statis dan pengetikan dinamis ... 161
3.6.3 Karakter dan nilai kebenaran ... 163

3.7 Aritmetika dan Jebakannya 165
3.7.1 Operator dan prioritas 165
3.7.2 Jebakan pertama: luapan bilangan bulat 166
3.7.3 Jebakan kedua: komputer tidak bisa menghitung sampai satu dalam persepuluhan 168
3.7.4 Jebakan ketiga: pembagian bilangan bulat 170
3.8 Percabangan 173
3.8.1 Melakukan sesuatu hanya bila sebuah kondisi terpenuhi 173
3.8.2 Rantai keputusan 174
3.8.3 `switch`: memilih di antara banyak nilai 175
3.9 Perulangan 178
3.9.1 `while` dan `for` 178
3.9.2 Empat deret 181
3.9.3 Perulangan for-each 183
3.9.4 Memilih perulangan yang tepat 184
3.10 Tabel Penelusuran 184
3.11 Latihan Terbimbing, Langkah demi Langkah 186
3.11.1 Luas lingkaran 186
3.11.2 Dari satuan imperial ke kilogram 187
3.11.3 Investasi selama lima ratus tahun 190
3.12 Profitina Laundry: Customer dan Service 195
3.12.1 Menemukan kelas-kelasnya 195
3.12.2 Aturan harga sebagai rumus 196
3.12.3 Kodenya 197
3.12.4 Memeriksa harga dengan tangan 204
3.13 Profitina dalam Praktik 205
3.14 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya 206
3.15 Ringkasan Bab 206
3.16 Soal Latihan 207
3.17 Penyelesaian Soal Latihan 208

BAB 4 Evaluasi dan Pembahasan I: Dari Spesifikasi Sederhana ke Kelas yang Teruji 218
4.1 Dari Mana Kita Berangkat 218
4.2 Cara Menggarap Soal Pemrograman 219
4.2.1 Delapan langkah 220
4.2.2 Cara uji dalam bab ini ditulis 221
4.2.3 Susunan setiap soal 221
4.3 Soal 4.1 — Kalkulator Pecahan 222
4.3.1 Soalnya 222
4.3.2 Memahaminya dengan tangan 223
4.3.3 Kelas, field, dan metode 223
4.3.4 Penyelesaian Java 223
4.3.5 Penyelesaian Python 225
4.3.6 Baris demi baris 227
4.3.7 Penyelesaian keliru yang menggoda 227
4.4 Soal 4.2 — Sudut Jarum Jam Analog 228
4.4.1 Soalnya 228
4.4.2 Memahaminya dengan tangan 229
4.4.3 Kelas, field, dan metode 231
4.4.4 Penyelesaian Java 231
4.4.5 Penyelesaian Python 233
4.4.6 Baris demi baris 235
4.4.7 Penyelesaian keliru yang menggoda 235
4.5 Soal 4.3 — Akun Komputer 237
4.5.1 Soalnya 237
4.5.2 Memahaminya dengan tangan 237

4.5.3 Kelas, field, dan metode 237
4.5.4 Penyelesaian Java 238
4.5.5 Penyelesaian Python 240
4.5.6 Baris demi baris 242
4.5.7 Penyelesaian keliru yang menggoda 243
4.6 Soal 4.4 — Bilangan Kompleks 244
4.6.1 Soalnya 244
4.6.2 Memahaminya dengan tangan 244
4.6.3 Kelas, field, dan metode 245
4.6.4 Penyelesaian Java 245
4.6.5 Penyelesaian Python 247
4.6.6 Baris demi baris 249
4.6.7 Penyelesaian keliru yang menggoda 250
4.7 Soal 4.5 — Konversi Berat 251
4.7.1 Soalnya 251
4.7.2 Memahaminya dengan tangan 251
4.7.3 Kelas, field, dan metode 251
4.7.4 Penyelesaian Java 251
4.7.5 Penyelesaian Python 253
4.7.6 Baris demi baris 254
4.7.7 Penyelesaian keliru yang menggoda 254
4.8 Soal 4.6 — Kelas Pecahan 256
4.8.1 Soalnya 256
4.8.2 Memahaminya dengan tangan 256
4.8.3 Kelas, field, dan metode 256
4.8.4 Penyelesaian Java 257
4.8.5 Penyelesaian Python 259
4.8.6 Baris demi baris 261
4.8.7 Penyelesaian keliru yang menggoda 261
4.9 Soal 4.7 — Pemain dan Musuh 263
4.9.1 Soalnya 263
4.9.2 Memahaminya dengan tangan 264
4.9.3 Kelas, field, dan metode 264
4.9.4 Penyelesaian Java 265
4.9.5 Penyelesaian Python 267
4.9.6 Baris demi baris 270
4.9.7 Penyelesaian keliru yang menggoda 271
4.10 Soal 4.8 — Uang dalam Rupiah 272
4.10.1 Soalnya 272
4.10.2 Memahaminya dengan tangan 272
4.10.3 Kelas, field, dan metode 273
4.10.4 Penyelesaian Java 273
4.10.5 Penyelesaian Python 275
4.10.6 Baris demi baris 277
4.10.7 Penyelesaian keliru yang menggoda 277
4.11 Soal 4.9 — Waktu dalam Sehari 278
4.11.1 Soalnya 278
4.11.2 Memahaminya dengan tangan 278
4.11.3 Kelas, field, dan metode 279
4.11.4 Penyelesaian Java 279
4.11.5 Penyelesaian Python 281
4.11.6 Baris demi baris 283
4.11.7 Penyelesaian keliru yang menggoda 283
4.12 Soal 4.10 — Nota Laundry 284
4.12.1 Soalnya 284

4.12.2 Memahaminya dengan tangan 284
4.12.3 Kelas, field, dan metode 284
4.12.4 Penyelesaian Java 285
4.12.5 Penyelesaian Python 288
4.12.6 Baris demi baris 290
4.12.7 Penyelesaian keliru yang menggoda 291
4.13 Soal 4.11 — Merapikan Nama 292
4.13.1 Soalnya 292
4.13.2 Memahaminya dengan tangan 292
4.13.3 Kelas, field, dan metode 293
4.13.4 Penyelesaian Java 293
4.13.5 Penyelesaian Python 295
4.13.6 Baris demi baris 296
4.13.7 Penyelesaian keliru yang menggoda 296
4.14 Soal 4.12 — Aturan Kata Sandi 298
4.14.1 Soalnya 298
4.14.2 Memahaminya dengan tangan 298
4.14.3 Kelas, field, dan metode 298
4.14.4 Penyelesaian Java 299
4.14.5 Penyelesaian Python 300
4.14.6 Baris demi baris 302
4.14.7 Penyelesaian keliru yang menggoda 302
4.15 Soal 4.13 — Menelusuri Perulangan dengan Tangan 303
4.15.1 Soalnya 303
4.15.2 Memahaminya dengan tangan 303
4.15.3 Kelas, field, dan metode 304
4.15.4 Penyelesaian Java 305
4.15.5 Penyelesaian Python 306
4.15.6 Baris demi baris 308
4.15.7 Penyelesaian keliru yang menggoda 308
4.16 Soal 4.14 — Tahun Kabisat dan Hari Berikutnya 309
4.16.1 Soalnya 309
4.16.2 Memahaminya dengan tangan 309
4.16.3 Kelas, field, dan metode 310
4.16.4 Penyelesaian Java 310
4.16.5 Penyelesaian Python 312
4.16.6 Baris demi baris 314
4.16.7 Penyelesaian keliru yang menggoda 314
4.17 Soal-Soal Berdampingan 315
4.18 Profitina dalam Praktik 316
4.19 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya 317
4.20 Ringkasan Bab 318
4.21 Soal Latihan 319
4.22 Penyelesaian Soal Latihan 319
Daftar Pustaka 321

BAB 5 Larik, List, Masukan, Eksepsi, dan Berkas 323
5.1 Di Mana Kita Berhenti 323
5.2 Banyak Nilai di Bawah Satu Nama: Larik 324
5.2.1 Rak kotak surat 324
5.2.2 Membuat, mengisi, dan menelusuri larik 325
5.2.3 Melangkah melewati ujung 328
5.2.4 Larik bilangan, dan larik berisi larik 330
5.2.5 Variabel larik menyimpan referensi 331
5.2.6 Larik di dalam `main`: argumen baris perintah 333

5.3 List yang Tumbuh: `ArrayList` dan `list` Python 334
5.3.1 Mengapa ukuran tetap tidak cukup 334
5.3.2 Operasi sehari-hari 335
5.3.3 Bagaimana list tumbuh 338
5.3.4 List berisi bilangan: *boxing* 339
5.3.5 Larik atau list? 341
5.4 Membaca Masukan dari Papan Ketik 341
5.4.1 `Scanner` dan `input()` 341
5.4.2 Membaca sampai bertemu *sentinel* 343
5.5 Ketika Sesuatu Salah: Eksepsi 345
5.5.1 Program berjumpa dengan pengguna 345
5.5.2 Membaca *stack trace* 346
5.5.3 Silsilah kegagalan 349
5.5.4 `try` dan `catch` 351
5.5.5 Beberapa klausa `catch`: jaring yang makin lama makin besar 353
5.5.6 `finally`: blok yang selalu dijalankan 355
5.5.7 Melempar eksepsi sendiri 358
5.6 Eksepsi Terperiksa dan Tak Terperiksa 360
5.6.1 Eksepsi tak terperiksa: biasanya kesalahan kita sendiri 360
5.6.2 Eksepsi terperiksa: kompilator bersikeras 362
5.6.3 Tangkap, atau deklarasikan? 364
5.6.4 Sikap Python 364
5.7 Eksepsi Buatan Sendiri 365
5.8 Berkas: *Stream*, *Reader*, dan *Writer* 368
5.8.1 *Stream* dan *reader* 368
5.8.2 Menutup apa yang Anda buka 370
5.8.3 `try`-with-resources dan `with` 372
5.8.4 Menulis dan menambahkan teks 374
5.8.5 Berkas CSV 377
5.9 Profitina Laundry: Memuat dan Menyimpan Pesanan 379
5.9.1 Kelas `Order` 381
5.9.2 Eksepsinya 383
5.9.3 Memuat dan menyimpan 384
5.9.4 Aplikasinya 389
5.10 Profitina dalam Praktik 393
5.11 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya 394
5.12 Ringkasan Bab 395
5.13 Soal Latihan 396
5.14 Penyelesaian Soal Latihan 398
Bacaan lanjutan 410
Daftar Pustaka 410

BAB 6 Dokumentasi, Pengujian, dan Antarmuka 412
6.1 Titik Berangkat Kita 412
6.2 Menengok Kembali Sebuah Kelas 413
6.2.1 Lebih dari satu cara lahir: beberapa konstruktor 413
6.3 Komentar Dokumentasi: Javadoc dan Docstring 418
6.3.1 Javadoc, langkah demi langkah 418
6.3.2 Docstring, langkah demi langkah 420
6.3.3 Contoh yang memeriksa dirinya sendiri: doctest 420
6.4 Membangkitkan dan Membaca Dokumentasi 421
6.4.1 javadoc 421
6.4.2 pydoc 423
6.4.3 Membaca API: kelas `Random` 423
6.5 Pengujian dari Nol 425

6.5.1 Kelas yang diuji 425
6.5.2 Uji tanpa pustaka apa pun 427
6.6 Anatomi Sebuah Uji Unit 430
6.6.1 Membandingkan bilangan pecahan 431
6.7 JUnit 6 dan unittest 432
6.7.1 Kelas uji JUnit 6 433
6.7.2 Menjalankan JUnit 6 di VS Code dan di terminal 434
6.7.3 unittest 435
6.7.4 pytest 436
6.8 Memilih Kasus Uji, dan Menulisnya Lebih Dulu 437
6.8.1 Kelas ekuivalensi dan batas 437
6.8.2 Uji lebih dulu 438
6.9 Kapan Dua Objek Dianggap Sama? 438
6.9.1 Identitas dan kesamaan 438
6.9.2 Menulis equals dan hashCode dengan benar 444
6.10 Menggambarkan Sebuah Objek: toString, str, dan repr 450
6.11 Banyak Versi Satu Metode: Overloading 450
6.11.1 Cara Java memilih sebuah overload 453
6.11.2 Python tidak mengenal overloading — dan tidak memerlukannya 455
6.12 Antarmuka 457
6.12.1 Mengapa antarmuka penting: kisah plug-in 463
6.13 Menukar Representasi di Balik Antarmuka 463
6.13.1 Tanpa antarmuka 463
6.13.2 Dengan antarmuka 468
6.13.3 Hierarki tipe: subtipe dan supertipe 472
6.13.4 Berbagi kode: metode default 472
6.14 Antarmuka yang Sudah Anda Pakai 475
6.14.1 Comparable, dan cast menuruni hierarki 476
6.14.2 Pendengar: gagasan di balik ActionListener 483
6.15 Antarmuka di Python: Kelas Dasar Abstrak dan Protocol 487
6.16 Memprogram terhadap Antarmuka: Penentuan Harga di Profitina Laundry 488
6.16.1 Pesanan 488
6.16.2 Antarmukanya 492
6.16.3 Tiga aturan dasar 494
6.16.4 Aturan yang membungkus aturan lain 497
6.16.5 Program yang memakai aturan-aturan itu 501
6.16.6 Menguji setiap aturan 505
6.16.7 Apa yang kita peroleh dari antarmuka 513
6.17 Profitina dalam Praktik 514
6.18 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya 514
6.19 Ringkasan Bab 516
6.20 Soal Latihan 516
6.21 Penyelesaian Soal Latihan 518
Daftar Pustaka 537

BAB 7 Pewarisan, Polimorfisme, dan Kontrol Akses 538

7.1 Di Mana Kita Berhenti 538
7.2 Masalah Duplikasi: Dua Basis Data yang Saling Mengulang 539
7.2.1 Kosakata 540
7.3 Superkelas dan Subkelas di Java dan Python 541
7.3.1 “Seolah-olah LinearDB didefinisikan sebagai…” 543
7.3.2 Program yang sama dalam Python 544
7.4 Mendefinisikan Ulang Metode: Overriding 546
7.4.1 Aturan yang diperiksa Java ketika Anda meng-override 555
7.5 Konstruktor dan `super` 556

7.6 Polimorfisme: Satu Panggilan, Banyak Perilaku 561
7.6.1 Membaca kelas Animal 568
7.6.2 Membaca program uji 569
7.6.3 Overloading versus overriding 570
7.7 Cara Kerja Dynamic Dispatch yang Sebenarnya: Gambaran Memori 575
7.7.1 Dua tipe untuk setiap variabel 575
7.7.2 Gambaran memori: tabel metode 578
7.7.3 Gagasan yang sama dalam Python 580
7.8 Akar Setiap Hierarki: `Object` dan `object` 582
7.9 Kelas Abstrak 586
7.9.1 Pustaka Java dibangun dengan cara yang sama 594
7.10 Mewarisi Kelas Konkret, dan Mengapa Harus Berhati-hati 594
7.11 Komposisi versus Pewarisan 599
7.11.1 Kelas dasar yang rapuh 600
7.11.2 Kapan pewarisan tepat? Prinsip substitusi 604
7.12 Pewarisan Jamak: MRO Python dan Antarmuka Java 604
7.12.1 Java: satu superkelas, banyak antarmuka 604
7.12.2 Python: pewarisan jamak sejati dan MRO 607
7.13 Kontrol Akses 610
7.13.1 Empat tingkat akses Java 610
7.13.2 Memilih tingkat akses 613
7.13.3 Python: konvensi, bukan aturan 613
7.14 Enkapsulasi: Objek yang Menepati Janjinya 615
7.14.1 Property di Python 617
7.15 Profitina Laundry: Hierarki Layanan dan Penetapan Harga Polimorfis 619
7.16 Profitina dalam Praktik 631
7.17 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya 632
7.18 Ringkasan Bab 633
7.19 Soal Latihan 634
7.20 Penyelesaian Soal Latihan 636
Daftar Pustaka 647

BAB 8 Evaluasi dan Pembahasan II: Struktur Rekursif, Kelas yang Tangguh, dan Hierarki Kelas 649
8.1 Titik Tolak Kita 649
8.2 Cara Menghadapi Soal Perancangan dengan Banyak Kelas 650
8.3 Dua Struktur dalam Bab Ini: Pohon dan *List* Cons 652
8.3.1 Pohon biner yang bekerja melalui delegasi 652
8.3.2 *List* imutabel yang dibangun dari nil dan cons 657
8.4 Soal-Soal tentang Pohon Biner 665
8.4.1 Soal 8.1 — Membangun pohon dengan tangan 665
8.4.2 Soal 8.2 — Melipattigakan setiap simpul 670
8.4.3 Soal 8.3 — Mencetak penelusuran *depth-first* 676
8.4.4 Soal 8.4 — Salinan segar sebuah pohon 680
8.4.5 Soal 8.5 — Menyimpan penelusuran ke dalam larik 687
8.4.6 Soal 8.6 — Mencari nilai di pohon pencarian biner 692
8.4.7 Soal 8.7 — Menyisipkan nilai ke pohon pencarian biner 698
8.5 Soal-Soal tentang *List* Rekursif 704
8.5.1 Soal 8.8 — Menegasikan sebuah *list* 704
8.5.2 Soal 8.9 — Menemukan posisi sebuah elemen 707
8.5.3 Soal 8.10 — Apakah semua elemen positif? 711
8.5.4 Soal 8.11 — Mempertahankan hanya elemen positif 714
8.5.5 Soal 8.12 — Apakah *list* terurut? 717
8.5.6 Soal 8.13 — Menggabungkan dua *list* terurut 720
8.5.7 Soal 8.14 — Melewati setiap salinan sebuah elemen 723

8.5.8 Soal 8.15 — Menghapus duplikat 726
8.6 Soal Baru yang Dikerjakan: Eksepsi, Berkas, dan Larik 729
8.6.1 Soal 8.16 — Pengurai berkas pesanan yang tahan eksepsi 729
8.6.2 Soal 8.17 — Larik terdokumentasi yang tumbuh sendiri 738
8.7 Soal Baru yang Dikerjakan: Antarmuka dan Ujinya 744
8.7.1 Soal 8.18 — Satu antarmuka, dua implementasi, satu set uji 744
8.7.2 Uji yang sama dalam JUnit 6 751
8.8 Soal Baru yang Dikerjakan: Pewarisan, Polimorfisme, dan Kontrol Akses 752
8.8.1 Soal 8.19 — Hierarki kelas yang ditelusuri dengan tangan 752
8.8.2 Soal 8.20 — Me-*refactor* dua kelas yang berulang menjadi sebuah hierarki 760
8.8.3 Soal 8.21 — Perburuan *bug* enkapsulasi 768
8.9 Perbandingan Soal-Soal dalam Bab Ini 774
8.10 Profitina dalam Praktik 775
8.11 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya 776
8.12 Ringkasan Bab 777
8.13 Soal Latihan 778
8.14 Penyelesaian Soal Latihan 779
Daftar Pustaka 786

BAB 9 Koleksi dan Generik 788
9.1 Di Mana Kita Berhenti 788
9.2 Dua Perkakas Bekal: Konstanta dalam Antarmuka dan Pembantu Rekursif 789
9.2.1 Konstanta dalam antarmuka 789
9.2.2 Rekursi sekali lagi: kasus dasar dan panggilan rekursif 794
9.2.3 Metode pembantu dengan parameter tambahan 795
9.3 Collections Framework: Peta Wilayahnya 800
9.3.1 Tiga keluarga: set, list, dan map 801
9.3.2 Memprogram terhadap antarmuka 801
9.3.3 Koleksi di Python 802
9.4 List 803
9.4.1 Operasi sehari-hari 803
9.4.2 ArrayList versus LinkedList 807
9.4.3 Mengunjungi setiap elemen, dan menghapus dengan aman 808
9.5 Set (Himpunan) 811
9.5.1 Himpunan bilangan, dan himpunan nama 811
9.5.2 Bagaimana HashSet menemukan elemen: hashing 815
9.5.3 Bagaimana TreeSet menjaga urutan 817
9.6 Map 818
9.6.1 Map dari kunci ke nilai 818
9.6.2 Membaca berkas ke dalam koleksi 822
9.7 Antrean, Tumpukan, Deque, dan Antrean Prioritas 826
9.8 Generik dari Nol 830
9.8.1 Kehidupan sebelum generik 830
9.8.2 Kelas generik: GList 833
9.8.3 Metode generik, dan akhirnya sebuah Pair 837
9.8.4 Parameter tipe terbatas 842
9.8.5 Wildcard: mengapa List bukan List 849
9.8.6 Type erasure: apa yang tersisa saat eksekusi 856
9.8.7 Generik di Java dan Python: ringkasan 859
9.9 Menata Urutan: Comparable dan Comparator 860
9.9.1 Iterator, duplikat, iterable, dan sebuah comparator 860
9.9.2 Comparator untuk pemakaian sungguhan 867
9.10 Memilih Koleksi yang Tepat 872
9.11 Profitina Laundry: Buku Pesanan dengan Beberapa Indeks 874
9.12 Profitina dalam Praktik 889

9.13 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya 889
9.14 Ringkasan Bab 890
9.15 Soal Latihan 891
9.16 Penyelesaian Soal Latihan 894
Daftar Pustaka 905

BAB 10 Antarmuka Grafis dan Penanganan Peristiwa 906
10.1 Di Mana Kita Berhenti 906
10.2 Dari Konsol ke Jendela: Siapa yang Memegang Kendali? 907
10.3 Tiga Toolkit: AWT, Swing, dan Tkinter 908
10.3.1 AWT: toolkit yang pertama 909
10.3.2 Swing: komponen yang menggambar dirinya sendiri 909
10.3.3 Tkinter: toolkit standar Python 910
10.4 Jendela Pertama Anda 911
10.4.1 Menjalankan jendela tanpa layar: uji mandiri 913
10.5 Komponen, Kontainer, dan Panel 915
10.5.1 Content pane 915
10.5.2 Panel, dan mengapa setiap jendela di sini adalah kelas 916
10.5.3 Pohon komponen sebuah jendela sungguhan 916
10.6 Gelung Peristiwa, Dijelaskan dari Nol 917
10.6.1 Dari mana peristiwa datang 917
10.6.2 Gelung itu sendiri 917
10.6.3 Gelung peristiwa yang dapat Anda baca 918
10.7 Mendengarkan Peristiwa 923
10.7.1 Membaca field teks 924
10.7.2 Objek peristiwa, dan komponen mana yang memicunya 927
10.7.3 Peristiwa item: kotak centang 927
10.7.4 Empat jenis peristiwa dalam bab ini 931
10.8 Layout Manager versus pack dan grid 931
10.8.1 FlowLayout: seperti kata dalam paragraf 932
10.8.2 BorderLayout: lima wilayah 932
10.8.3 BoxLayout: satu baris atau satu kolom 933
10.8.4 GridBagLayout: yang paling luwes 933
10.8.5 Tkinter: pack, grid, dan place 934
10.8.6 Mengamati para pengelola bekerja 935
10.9 Melukis: paintComponent dan Canvas 942
10.9.1 Sistem koordinat, dan objek Graphics 942
10.9.2 Warna 943
10.9.3 Siapa yang memanggil paintComponent? 943
10.9.4 Kotak yang menggelap dan menerang 944
10.9.5 Canvas Tkinter: bentuk yang mengingat dirinya sendiri 949
10.10 Kelas Dalam, Kelas Anonim, dan Lambda 952
10.10.1 Kelas dalam 952
10.10.2 Kelas anonim dan lambda 952
10.10.3 Lima gaya berdampingan 953
10.10.4 Effectively final: mengapa Java menyalin 957
10.10.5 Python: bound method, closure, lambda, partial, objek yang dapat dipanggil 957
10.10.6 Menggambar dengan tetikus 961
10.11 Event Dispatch Thread dan Aturan Thread 966
10.11.1 Satu thread memiliki jendela 966
10.11.2 Pekerjaan lambat, cara yang salah dan cara yang benar 967
10.12 Profitina Laundry: Jendela Pencatatan Pesanan 974
10.12.1 Jendela, dideskripsikan dengan cermat 975
10.12.2 Harga, secara cermat 976
10.12.3 Rancangan: model yang tidak tahu apa-apa tentang jendela 977

10.12.4 Program Swing 978
10.12.5 Uji mandiri 988
10.12.6 Program Tkinter 989
10.13 Profitina dalam Praktik 998
10.14 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya 999
10.15 Ringkasan Bab 1000
10.16 Soal Latihan 1001
10.17 Penyelesaian Soal Latihan 1003
Daftar Pustaka 1012

BAB 11 Pola Model–View–Controller 1014
11.1 Di Mana Kita Berhenti 1014
11.2 Sebuah Kisah: Meja Depan yang Tahu Terlalu Banyak 1015
11.3 Tiga Peran: Model, View, Controller 1017
11.3.1 Model, view, dan controller yang dapat ditunjuk 1018
11.3.2 Siapa boleh mengenal siapa 1020
11.3.3 Asal-usul gagasan ini 1021
11.4 Program MVC Pertama: Data Mahasiswa 1021
11.5 Pola Observer di Bawah MVC 1027
11.5.1 Java: dari `Observable` ke `PropertyChangeSupport` 1028
11.5.2 Python: pengamat dalam selusin baris 1031
11.5.3 Satu perubahan, banyak panggilan 1034
11.5.4 Didorong atau ditarik? 1035
11.6 Pencacah, Dibangun Ulang: MVC dengan Jendela 1035
11.6.1 Membaca program 1041
11.6.2 Jendelanya 1041
11.6.3 Uji mandiri 1042
11.7 Papan Tombol Ponsel: Teks Prediktif dalam MVC 1046
11.7.1 Kata, tombol, dan tanda tangan 1046
11.7.2 Programnya 1048
11.7.3 Jendelanya 1055
11.7.4 Uji mandiri 1056
11.7.5 Papan tombol yang sama dalam Python 1057
11.8 Menguji Model Tanpa GUI 1063
11.9 Meja Depan Profitina Laundry: Aplikasi MVC yang Lengkap 1069
11.9.1 Aturan-aturan model 1069
11.9.2 Rancangannya 1070
11.9.3 Modelnya 1071
11.9.4 Controller dan uji mandiri 1079
11.9.5 View-view Swing 1087
11.9.6 Meja depan yang sama dalam Python 1092
11.10 Unit Test untuk Model Meja Depan 1101
11.11 Variasi: MVP dan MVVM 1105
11.12 Profitina dalam Praktik 1107
11.13 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya 1107
11.14 Ringkasan Bab 1109
11.15 Soal Latihan 1109
11.16 Penyelesaian Soal Latihan 1111
Daftar Pustaka 1122

BAB 12 Evaluasi dan Pembahasan III: Teks Prediktif dengan List, Map, Pohon, dan Papan Tombol 1123
12.1 Titik Tolak Kita 1123
12.2 Cara Menghadapi Soal Koleksi dan Antarmuka 1124
12.3 Masalah Teks Prediktif 1125

12.3.1 Tombol, huruf, dan signature 1125
12.3.2 Berkas kamus 1126
12.3.3 Paket dan cara menjalankan program 1126
12.3.4 Seluruh rancangan sekilas 1127
12.4 Bagian Satu: Prototipe 1128
12.4.1 Soal 12.1 — Dari kata ke signature-nya 1128
12.4.2 Soal 12.2 — Dari signature ke kata-katanya, cara yang lambat 1137
12.4.3 Soal 12.3 — Dua program baris perintah 1138
12.5 Bagian Dua: *List* Terurut dan Antarmuka 1142
12.5.1 Soal 12.4 — *List* terurut berisi pasangan dan pencarian biner 1142
12.5.2 Soal 12.5 — Antarmuka `Dictionary` 1152
12.5.3 Soal 12.6 — Program *list*, dan pengukuran waktu pertama 1154
12.6 Bagian Tiga: *Map* dan Pohon 1163
12.6.1 Soal 12.7 — *Map* bernilai jamak 1163
12.6.2 Soal 12.8 — Pohon tombol yang juga mencocokkan prefiks 1169
12.6.3 Soal 12.9 — Satu baris yang diubah, dan adu cepat lengkap 1180
12.7 Bagian Empat: Papan Tombol dalam MVC 1184
12.7.1 Soal 12.10 — GUI teks prediktif 1184
12.8 Soal Baru yang Dikerjakan: Koleksi dan Generik 1201
12.8.1 Soal 12.11 — Memilih koleksi dengan mengukurnya 1201
12.8.2 Soal 12.12 — Kelas generik dengan parameter tipe berbatas 1208
12.8.3 Soal 12.13 — Rantai *comparator* 1215
12.9 Soal Baru yang Dikerjakan: Peristiwa, Observer, dan MVC 1221
12.9.1 Soal 12.14 — Penghitung berbasis peristiwa dengan model yang teruji 1221
12.9.2 Soal 12.15 — Dua *bug* observer 1230
12.9.3 Soal 12.16 — Tombol-tombol yang dibuat dalam gelung 1236
12.10 Perbandingan Soal-Soal dalam Bab Ini 1241
12.11 Profitina dalam Praktik 1242
12.12 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya 1243
12.13 Ringkasan Bab 1244
12.14 Soal Latihan 1245
12.15 Penyelesaian Soal Latihan 1245
Daftar Pustaka 1253

BAB 13 Iterator, Imutabilitas, dan Koleksi Lanjutan 1255
13.1 Di Mana Kita Berhenti 1255
13.2 Menelusuri Koleksi: Iterator 1256
13.2.1 Protokol Iterator di Java 1256
13.2.2 Protokol yang sama di Python 1259
13.2.3 Apa yang tidak dapat dilakukan iterator 1261
13.3 Iterable, Perulangan for yang Disempurnakan, dan ListIterator 1262
13.3.1 Menghapus duplikat yang bersebelahan 1262
13.3.2 ListIterator: mundur, mengganti, menyisipkan, menghapus 1266
13.4 Menulis Koleksi Iterable Sendiri 1268
13.4.1 Java: Iterable ditambah kelas iterator dalam 1268
13.4.2 Python: iter, next, dan generator 1271
13.4.3 Iterable bukanlah iterator 1274
13.5 Mengubah Koleksi Selagi Menelusurinya 1275
13.6 Mengurutkan Objek: Comparable dan Comparator 1281
13.6.1 Kontrak yang harus dipegang setiap perbandingan 1281
13.6.2 Comparator berdasarkan usia 1282
13.6.3 Nama dalam tiga urutan, dan perbandingan yang hanya tampak benar 1286
13.7 Salinan: Aliasing, Dangkal, dan Dalam 1291
13.8 Imutabilitas dari Nol 1296
13.8.1 Mengapa ada yang menginginkan objek yang tidak dapat berubah? 1296

13.8.2 final bukan berarti imutabel; tampilan dan salinan 1300
13.8.3 Menulis kelas imutabel 1304
13.9 Objek Nilai, Kesamaan, dan Hashing 1311
13.10 Stream, Comprehension, dan itertools 1317
13.11 Profitina Laundry: Uang, Riwayat Pesanan, dan Pipeline Laporan 1323
13.12 Profitina dalam Praktik 1338
13.13 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya 1338
13.14 Ringkasan Bab 1339
13.15 Soal Latihan 1340
13.16 Penyelesaian Soal Latihan 1342
Daftar Pustaka 1352

BAB 14 Thread, Race, Deadlock, dan Livelock 1354
14.1 Di Mana Kita Berhenti 1354
14.2 Apa Itu Thread? 1355
14.2.1 Konkurensi dan paralelisme 1356
14.2.2 Mengapa memakai thread? 1357
14.3 Induk yang Tak Dapat Merebut Kembali Kendali 1357
14.3.1 Anak-anak sebagai thread 1358
14.3.2 Berhenti dengan bendera, dan jebakan visibilitas 1363
14.4 Empat Cara Menciptakan Thread 1366
14.5 Kehidupan Sebuah Thread 1371
14.6 Global Interpreter Lock Python, dan Kapan Thread Membantu 1375
14.6.1 Global Interpreter Lock 1376
14.7 Race Condition: Ketika Dua Thread Berbagi Pencacah 1379
14.7.1 Mengapa `c++` bukan satu langkah 1382
14.7.2 Race yang sama di Python 1384
14.8 Saling Pengecualian: `synchronized`, Kunci, dan Atomik 1386
14.8.1 Mengapa kunci juga membereskan visibilitas 1391
14.8.2 Kunci di Python 1391
14.9 Menunggu Suatu Kondisi: `wait`, `notifyAll`, dan `Condition` 1393
14.10 Produsen dan Konsumen dengan Cara Mudah: Antrean Pemblokir 1401
14.11 Deadlock 1403
14.11.1 Para filsuf yang bersantap 1404
14.11.2 Dua petugas dan dua mesin 1410
14.12 Mencegah Deadlock 1414
14.13 Livelock dan Starvation 1420
14.13.1 Livelock 1420
14.13.2 Starvation dan keadilan 1427
14.14 Profitina Laundry: Mesin sebagai Thread Pekerja 1427
14.15 Profitina dalam Praktik 1436
14.16 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya 1437
14.17 Ringkasan Bab 1438
14.18 Soal Latihan 1439
14.19 Penyelesaian Soal Latihan 1441
Daftar Pustaka 1448

BAB 15 Socket, Jaringan, dan Proyek Tim 1449
15.1 Di Mana Kita Berhenti 1449
15.2 Jaringan dari Nol 1450
15.2.1 Host dan alamat IP 1450
15.2.2 Port 1450
15.2.3 Lapisan: bagaimana sebuah pesan berpindah 1451
15.2.4 TCP dan UDP 1452
15.2.5 Program jaringan pertama: datagram untuk diri sendiri 1453

15.2.6 Klien dan server 1456
15.3 Socket di Java dan Python 1457
15.4 Server dan Klien *Echo* Berbasis Baris 1458
15.4.1 Server 1459
15.4.2 Klien 1461
15.4.3 Menjalankan pasangan ini di dua terminal 1464
15.4.4 Baris, pembilasan, dan pembingkaian 1464
15.4.5 Ketika pihak lain tidak ada 1465
15.5 Merancang Protokol di Atas Kertas Terlebih Dahulu 1465
15.6 Melayani Banyak Klien Sekaligus 1472
15.6.1 Satu thread per klien, dari sebuah kumpulan 1472
15.6.2 Server knock-knock multiklien 1472
15.6.3 Klien, dan tiga terminal 1478
15.6.4 Membuktikan bahwa server benar-benar konkuren 1481
15.7 Profitina Laundry: Layanan Status Pesanan dengan JSON di atas TCP 1485
15.7.1 Spesifikasi, di atas kertas 1485
15.7.2 JSON di kedua bahasa 1487
15.7.3 Server 1495
15.7.4 Objek klien 1510
15.7.5 Java bercakap dengan Python, dan Python bercakap dengan Java 1517
15.7.6 Biaya sebuah percakapan 1521
15.8 HTTP: Protokol Web 1521
15.8.1 Seperti apa percakapan HTTP 1522
15.8.2 Server dan klien HTTP di setiap bahasa 1522
15.9 Keamanan dan Data Pribadi 1527
15.9.1 Tujuh kebiasaan untuk kode jaringan 1527
15.9.2 Apa yang diminta UU PDP dari program seperti ini 1527
15.10 Menguji Program Jaringan 1528
15.11 Proyek Tim 1535
15.11.1 Memilih topik 1535
15.11.2 Membagi pekerjaan 1536
15.11.3 Dari spesifikasi ke rancangan ke implementasi 1537
15.11.4 Kasus uji melintasi antarmuka 1537
15.11.5 Bekerja bersama dengan Git 1538
15.11.6 README 1539
15.11.7 Kegagalan: server mati, klien mati 1539
15.11.8 Dokumentasi yang sungguh dipakai 1540
15.11.9 Mempresentasikan proyek 1540
15.11.10 Contoh ringkasan proyek yang lengkap 1540
15.11.11 Daftar periksa untuk tim 1542
15.12 Profitina dalam Praktik 1542
15.13 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya 1543
15.14 Ringkasan Bab 1544
15.15 Soal Latihan 1544
15.16 Penyelesaian Soal Latihan 1546
Daftar Pustaka 1564

BAB 16 Evaluasi dan Pembahasan IV: Iterator, Konkurensi, Jaringan, dan Keseluruhan Buku 1566
16.1 Titik Tolak Kita 1566
16.2 Cara Menghadapi Proyek Satu Sistem Utuh 1568
16.3 Tugas Proyek, Dinyatakan Ulang 1570
16.4 Soal-Soal tentang Perencanaan dan Rancangan Protokol 1571
16.4.1 Soal 16.1 — Topik, pembagian kerja, dan kontrak pertama 1571
16.4.2 Soal 16.2 — Protokol sebagai diagram alir dan mesin keadaan 1581

16.5 Soal-Soal tentang Membangun dan Menguji Sistem Klien–Server ... 1590
16.5.1 Soal 16.3 — Server untuk banyak klien, dan klien konsol ... 1590
16.5.2 Soal 16.4 — Setiap pihak menguji pihak lain, dengan *stub* ... 1605
16.5.3 Soal 16.5 — Banyak klien sekaligus: uji *multithread* dan terdistribusi ... 1610
16.6 Soal-Soal tentang Ketangguhan, Antarmuka Grafis, dan Dokumentasi ... 1618
16.6.1 Soal 16.6 — Ketika server atau klien mati ... 1618
16.6.2 Soal 16.7 — Jendela yang tidak pernah membeku ... 1626
16.6.3 Soal 16.8 — Kriteria kinerja, dokumentasi produk, dan laporan ... 1639
16.7 Soal Baru yang Dikerjakan: Iterator dan Nilai Imutabel ... 1644
16.7.1 Soal 16.9 — Jadwal penjemputan dengan iteratornya sendiri ... 1644
16.7.2 Soal 16.10 — Uang sebagai objek nilai yang imutabel ... 1653
16.8 Soal Baru yang Dikerjakan: *Thread* ... 1663
16.8.1 Soal 16.11 — Pencacah bersama, *race* yang diukur, dan penawarnya ... 1663
16.8.2 Soal 16.12 — Lini cuci produsen–konsumen ... 1672
16.9 Soal Baru yang Dikerjakan: Protokol Mungil Buatan Kita Sendiri ... 1682
16.9.1 Soal 16.13 — LAUNDRY/1, protokol permintaan–balasan ... 1682
16.10 Proyek Puncak: Profitina Laundry dari Ujung ke Ujung ... 1692
16.10.1 Soal 16.14 — Sistem binatu yang kecil tetapi lengkap ... 1692
16.11 Perbandingan Soal-Soal dalam Bab Ini ... 1718
16.12 Seluruh Buku dalam Satu Tabel ... 1718
16.13 Ke Mana Setelah Ini ... 1719
16.13.1 Pola rancangan ... 1719
16.13.2 Perkakas *build* dan lingkungan ... 1720
16.13.3 *Framework* ... 1721
16.14 Profitina dalam Praktik ... 1721
16.15 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya ... 1722
16.16 Ringkasan Bab dan Kata Penutup ... 1723
16.17 Soal Latihan ... 1724
16.18 Penyelesaian Soal Latihan ... 1725
Daftar Pustaka ... 1734

Daftar Gambar

Gambar 1.1 Bagaimana kode sumber menjadi program yang berjalan: Java dikompilasi sekali menjadi *bytecode* yang dijalankan Java Virtual Machine; Python dikompilasi menjadi *bytecode* setiap kali dijalankan, di dalam interpreter. 4
Gambar 1.2 Kelas `Order` (atas) adalah formulir kosong: ia mendaftar keadaan yang dimiliki setiap pesanan dan perilaku yang ditawarkan setiap pesanan; objek `ani` dan `budi` (bawah) adalah dua nota yang sudah diisi. 7
Gambar 1.3 Suatu pagi di meja layanan sebagai percakapan antarobjek: panah utuh adalah pesan, panah putus-putus adalah jawaban, dan angkanya menunjukkan urutan kejadiannya. 8
Gambar 1.4 Tiga pesanan yang sama disusun dengan dua cara: di kiri, empat larik paralel yang dapat ditulisi baris mana pun dalam program; di kanan, satu objek per pesanan, yang keadaannya hanya dapat diubah oleh metode-metodenya sendiri. 12
Gambar 1.5 Bahasa yang paling banyak dipakai menurut Survei Pengembang Stack Overflow 2025, seluruh responden: Python di urutan keempat dan Java di urutan ketujuh dari lima belas bahasa yang ditampilkan. 15
Gambar 1.6 Peringkat PYPL untuk April 2026, berdasarkan seberapa sering tutorial setiap bahasa dicari: Python pertama, Java ketiga. 16
Gambar 1.7 Peringkat TIOBE untuk September 2026, berdasarkan insinyur, kursus, dan vendor per bahasa: Python pertama, Java keempat. 16
Gambar 1.8 Peringkat GitHub Octoverse 2025 menurut kontributor bulanan: TypeScript, Python, dan JavaScript memimpin, dan Java di urutan keempat. 17
Gambar 1.9 Folder `bin` sebuah JDK, ditampilkan di terminal Linux; kelima perkakas pada **Tabel 1.3** disorot, dan di Windows setiap berkasnya berakhiran `.exe`. 19
Gambar 1.10 Membuka Command Prompt di Windows: setelah mengetik `cmd` di kotak pencarian bilah tugas, pilih kecocokan terbaik (1). 20
Gambar 1.11 Menambahkan folder `bin` milik JDK ke bagian depan PATH untuk jendela Command Prompt saat ini dengan perintah `set path=`. 20
Gambar 1.12 Riwayat hidup sebuah pesanan: setiap pesan memajukannya satu langkah, dan pesan yang dikirim pada saat yang salah, seperti `pickUp()` ketika cucian masih dicuci, ditolak dan tidak mengubah apa pun. 34
Gambar 2.1 Dari kode sumber ke prosesor. Java mengompilasi sekali dengan javac dan JVM menjalankan bytecode-nya setiap kali; Python mengompilasi di dalam interpreter setiap kali program dimulai, lalu mesin virtualnya menjalankan bytecode itu. 59
Gambar 2.2 Daftar paket pada Eclipse Installer. Eclipse IDE for Java Developers adalah paket yang dipilih. 64
Gambar 2.3 Memilih runtime Java untuk Eclipse dan folder pemasangan, sesaat sebelum menekan INSTALL. 64
Gambar 2.4 Eclipse IDE Launcher meminta sebuah folder workspace sebelum meja kerja terbuka. 65
Gambar 2.5 File > Switch Workspace > Other… membuka kembali peluncur agar Anda dapat memilih workspace lain. 66
Gambar 2.6 Halaman preferensi Appearance, tempat tema warna dipilih. 67
Gambar 2.7 Window > Perspective > Open Perspective, dengan perspective Debug dan Java Browsing ditawarkan langsung dan sisanya di bawah Other. 68
Gambar 2.8 Dialog Open Perspective mendaftar setiap perspective pada instalasi; Java ditandai sebagai bawaan. 68
Gambar 2.9 Perspective Java pada workspace kosong: Package Explorer di kiri, area editor di tengah, Outline di kanan, serta Problems, Javadoc, dan Declaration di bawah. 69
Gambar 2.10 Dialog Show View, dikelompokkan per kategori; ketik sebagian nama view di kotak saring untuk menemukannya. 70
Gambar 2.11 View Problems dengan dua peringatan; menu konteksnya menawarkan Quick Fix. 71
Gambar 2.12 View Javadoc menampilkan dokumentasi println(String) ketika kursor berada pada pemanggilannya. 71

Gambar 2.13 Klik kanan pada ruler kiri editor: Show Line Numbers tercentang, dan Toggle Breakpoint berada paling atas. 72
Gambar 2.14 Wizard New Java Project, dengan folder terpisah untuk berkas sumber dan berkas kelas. 73
Gambar 2.15 Klik kanan pada src, lalu New > Package. 74
Gambar 2.16 Dialog New Java Package: folder sumber dan nama paket. 74
Gambar 2.17 Dialog New Java Class, dengan kerangka method main tercentang. 75
Gambar 2.18 Run As > Java Application dari menu konteks kelas. 76
Gambar 2.19 View Console setelah program dijalankan. Baris pertama mencatat program mana yang berjalan, di atas Java yang mana, dan kapan. 76
Gambar 2.20 Wizard Export: JAR file di bawah kategori Java. 77
Gambar 2.21 Halaman JAR File Specification: sumber daya mana yang diekspor dan ke mana JAR ditulis. 78
Gambar 2.22 Menjalankan kelas dari JAR di Command Prompt Windows. 78
Gambar 2.23 Peta jendela VS Code saat men-debug: Explorer dan Java Projects di bilah samping, editor dengan breakpoint di gutter, view Run and Debug, bilah alat debug, panel, dan pemilih interpreter di bilah status. 79
Gambar 2.24 Versi awal Counter.java: komentar TODO menguraikan apa yang harus dikerjakan count(x), dan method itu mengembalikan 0 sampai semuanya selesai. 84
Gambar 2.25 Counter cannot be resolved to a type: kotak quick fix menawarkan import util.Counter, atau org.w3c.dom.css.Counter yang tidak berkaitan. 87
Gambar 2.26 Organize Imports menanyakan Counter mana yang dimaksud ketika nama pendeknya ambigu. 87
Gambar 2.27 Mengekspor dua proyek ke berkas arsip, dengan struktur foldernya tetap terjaga. 89
Gambar 2.28 Mengimpor proyek yang sudah ada dari berkas arsip ke dalam workspace. 89
Gambar 2.29 Menu view Package Explorer, dengan Link with Editor yang akan dinyalakan. 90
Gambar 2.30 Toggle Breakpoint pada menu konteks ruler kiri editor (Ctrl+Shift+B). 93
Gambar 2.31 Sebuah breakpoint pada baris 8, Counter counter = new Counter(); titik di ruler menandainya. 93
Gambar 2.32 Debug As > Java Application menjalankan program di bawah debugger. 94
Gambar 2.33 Tombol Debug pada bilah alat menjalankan ulang peluncuran debug terakhir, di sini Tester. 94
Gambar 2.34 Eclipse menawarkan untuk membuka perspective Debug ketika program pertama kali tertangguhkan. 95
Gambar 2.35 Tombol eksekusi debugger Eclipse beserta tombol papan ketiknya: Resume (F8), Terminate (Ctrl+F2), Step Into (F5), Step Over (F6), dan Step Return (F7). 95
Gambar 2.36 Langkah-langkah debugger dilihat pada tumpukan pemanggilan. Step Into turun ke method yang dipanggil, Step Out kembali ke pemanggil, dan Step Over menjalankan seluruh pemanggilan dan tetap pada tingkat yang sama. 96
Gambar 2.37 View Debug menampilkan thread main yang tertangguhkan di Tester.main pada baris 11; setiap baris di bawah thread adalah satu frame tumpukan pemanggilan. 97
Gambar 2.38 View Variables sesudah count(5) mengembalikan nilainya: args, counter, dan result = 15. 97
Gambar 2.39 Mengubah nilai selama debugging: variabel gelung i di dalam count(int) sedang disunting di view Variables. 98
Gambar 2.40 Tanpa method toString(), panel detail hanya menampilkan util.Counter@71bbf57e. 98
Gambar 2.41 Detail Formatter untuk util.Counter yang menampilkan getResult() sebagai gantinya. 99
Gambar 2.42 View Breakpoints mendaftar setiap breakpoint; kotak centangnya menyalakan atau mematikan breakpoint, dan bagian bawahnya memuat properti Hit count dan Conditional. 100
Gambar 2.43 Skip All Breakpoints (Ctrl+Alt+B) menonaktifkan setiap breakpoint tanpa menghapus satu pun. 100
Gambar 2.44 Siklus debugging. Bila perbandingan pada langkah 5 membantah hipotesis, kembalilah ke langkah 3 dengan membawa apa yang sudah Anda pelajari, bukan kembali menebak-nebak. 114
Gambar 3.1 Kelas `Hello` terbuka di editor Eclipse: satu kelas, satu metode `main`, satu pernyataan. 131

Gambar 3.2 `Hello2` di Eclipse: program membaca kata pertama dari baris perintah melalui `args[0]` bila kata itu diberikan. 132
Gambar 3.3 Kelas `Dog` sebagai cetak biru (kiri) dan dua objek yang dibangun darinya dengan `new` (kanan); setiap objek memiliki salinan sendiri dari setiap *field*. 136
Gambar 3.4 Bagian pertama `Dog.java` di Eclipse: *field*, konstruktor, serta *getter* dan *setter* yang pertama....138
Gambar 3.5 Sisa `Dog.java` di Eclipse: *getter* dan *setter* yang tersisa, dan `toString`. 138
Gambar 3.6 Kelas uji JUnit `DogTest` di Eclipse; metode bertanda `@Test` membuat seekor anjing dan memakai `assertEquals` untuk membandingkan umur harapan dengan umur sebenarnya. 145
Gambar 3.7 Package Explorer di Eclipse: `Dog.java` dan `DogTest.java` di dalam proyek `dog`, dengan berkas pustaka JUnit di bawah *Referenced Libraries*. 145
Gambar 3.8 Bilah merah: `test1` gagal dengan harapan 3 tetapi ternyata 0, karena konstruktor `Dog` versi ini menulis `age = age;` alih-alih `this.age = age;`. 146
Gambar 3.9 Bilah hijau: setelah diperbaiki, pengujian berjalan dan lulus....146
Gambar 3.10 Memori di dalam `main` pada **Kode 3.13**: `a` dan `b` menyimpan nilai, sedangkan `fido`, `alias`, dan `twin` menyimpan panah; `fido` dan `alias` menunjuk ke objek yang sama. 148
Gambar 3.11 Nama dan objek dalam **Kode 3.14**: setiap nama adalah panah; `fido` dan `alias` terikat ke objek `Dog` yang sama. 150
Gambar 3.12 Metode string tidak pernah mengubah string: `toUpperCase` dan `+` membangun string baru, dan penugasan hanya memindahkan panah. 155
Gambar 3.13 Dua objek `String` berbeda dengan karakter yang sama: `==` menjawab `false`, `equals` menjawab `true`....156
Gambar 3.14 Galat setelah 0,1 dijumlahkan berulang-ulang, untuk `float` dan `double`; lekukan sampai ke dasar grafik adalah saat-saat ketika galat pembulatan kebetulan saling meniadakan. 169
Gambar 3.15 Rantai `else if` sebagai diagram alir: kondisi dicoba dari atas, dan kondisi benar yang pertama yang memutuskan. 175
Gambar 3.16 Tanpa `break`, eksekusi jatuh terus dari *case* yang cocok ke *case* berikutnya; dengan `break`, eksekusi keluar dari `switch`. 177
Gambar 3.17 Empat bagian `for (int j = 0; j < 4; j++)`: inisialisasi sekali, lalu kondisi, badan, dan pembaruan dalam satu siklus sampai kondisinya salah. 180
Gambar 3.18 Saldo GBP 100 dengan bunga 2,3% per tahun selama 500 tahun, pada skala logaritmik; garis lurus adalah ciri khas pertumbuhan majemuk. 193
Gambar 3.19 Dua kelas pertama Profitina Laundry dan program yang memakainya. 195
Gambar 3.20 Perhitungan harga dalam `Service.price`: berat yang ditagihkan, harga dasar, lalu dua penyesuaian yang bersifat pilihan. 197
Gambar 4.1 Delapan langkah yang dipakai untuk setiap soal dalam bab ini; panah putus-putus adalah putaran antara menulis kode dan menjalankan uji. 220
Gambar 4.2 Jam pada pukul 3:00 (kiri) dan pukul 9:00 (kanan); busur jingga bergerak berlawanan arah jarum jam dari jarum jam (pendek, gelap) ke jarum menit (panjang, hijau kebiruan), menghasilkan 90 ◦ dan 270 ◦. 229
Gambar 4.3 Jam pada pukul 4:41: jarum jam telah merayap ke 140.5 ◦, jarum menit menunjuk 246 ◦, dan sudut berlawanan arah jarum jam di antara keduanya 254.5 ◦, yang dibulatkan setengah ke genap menjadi 254. 230
Gambar 4.4 Kotak bawah: `getSum` yang benar membiarkan `f1` dan `f2` dan mengembalikan objek baru. Kotak atas: versi menggoda dari Subbab 4.8.7 menimpa `f2` lalu mengembalikannya, sehingga `f2` dan hasilnya adalah objek yang sama yang telah berubah. 257
Gambar 4.5 Diagram kelas Soal 4.10: sebuah `LaundryReceipt` memiliki total bertipe `Rupiah`; belah ketupat terisi menandai bahwa total itu milik nota. 285
Gambar 5.1 Sebuah larik berisi lima string di dalam memori. Variabel menyimpan referensi ke objek larik; slot-slotnya bernomor 0 sampai 4, dan indeks 5 tidak ada. 325
Gambar 5.2 Yang dilakukan `add` ketika larik tersembunyinya penuh: larik yang lebih besar dibuat, referensi-referensi lama disalin ke dalamnya, dan larik lama dibuang. 339
Gambar 5.3 Propagasi. Panggilan bergerak turun dari `main`; eksepsi berjalan kembali ke atas, meninggalkan setiap metode yang dilaluinya, sampai mencapai `catch` yang cocok. 348

Gambar 5.4 Sebagian hierarki eksepsi Java. Kelas berwarna biru adalah eksepsi terperiksa, yang kuning tak terperiksa, dan yang merah adalah *error* yang tidak semestinya dicoba ditangani oleh program. 350
Gambar 5.5 Sebagian hierarki eksepsi Python. Program menangkap subkelas-subkelas `Exception`; eksepsi-eksepsi di sampingnya meminta program berhenti. 351
Gambar 5.6 Jalur-jalur yang melewati pernyataan `try–catch–finally`. Setiap jalur melewati blok `finally`. 356
Gambar 5.7 Alur pemuatan pesanan. Baris yang buruk menjadi satu pesan dalam daftar kesalahan dan pemuatan berlanjut; berkas yang tidak ada atau *header* yang salah menghentikan pemuatan dan dilaporkan oleh aplikasi. 380
Gambar 6.1 Dari komentar dokumentasi menjadi dokumentasi: `javadoc` membangun halaman HTML yang saling bertaut, `pydoc` mencetak teks bantuan atau menulis halaman HTML, dan `doctest` menjalankan contoh yang ditulis di dalam *docstring*. 421
Gambar 6.2 *Arrange, act, assert*: bangun objeknya, panggil satu metode yang diuji, bandingkan nilai yang diharapkan dengan nilai sebenarnya. 430
Gambar 6.3 Siklus uji-lebih-dulu: uji yang gagal (merah), kode paling sederhana yang meluluskannya (hijau), lalu perapian dengan uji sebagai jaring pengaman (*refactor*). 438
Gambar 6.4 Dua referensi ke dua objek berbeda yang isinya sama: tidak identik, tetapi sama jika kelasnya berkata demikian — dan kalau begitu, kode *hash* keduanya juga harus sama. 439
Gambar 6.5 Klien hanya bergantung pada antarmuka `DBops`; `LinearDB` dan `TreeDB` mengimplementasikannya dan dapat dipertukarkan tanpa mengubah klien. 472
Gambar 6.6 Bergerak naik dalam hierarki tipe (dari `Name` ke `Comparable` atau `Object`) terjadi otomatis; bergerak turun memerlukan *cast* eksplisit, yang diperiksa ketika program berjalan. 480
Gambar 6.7 Rancangan penentuan harga: lima kelas mengimplementasikan `PriceRule`, dua di antaranya dengan membungkus `PriceRule` lain; program demo hanya bergantung pada antarmuka. 499
Gambar 7.1 Masalah duplikasi dan obatnya. Kiri: dua kelas yang berdiri sendiri, masing-masing membawa salinan `addSeveral` dan `findOneOf`. Kanan: metode bersama naik ke superkelas abstrak `DB`; setiap subkelas hanya menyimpan apa yang benar-benar miliknya. 540
Gambar 7.2 Perangkaian konstruktor untuk `new Shirt("blue", true)`. Panggilan-panggilan naik menyusuri hierarki sampai `Object` lebih dahulu (panah utuh); badan-badan konstruktor lalu selesai dalam urutan terbalik (panah putus-putus), sehingga bagian objek yang paling umum selalu siap sebelum bagian yang paling khusus. 558
Gambar 7.3 Diagram kelas UML program Animal. `Animal` memiliki empat konstruktor yang di-*overload* dan empat metode `getPrice` yang di-*overload*; `Cat`, `Cow`, dan `Dog` masing-masing meng-*override* `sound()`, dan `Cat` menambahkan satu metode miliknya sendiri. Segitiga kosong menunjuk dari subkelas ke superkelasnya; `-` menandai anggota private dan `+` anggota public. 562
Gambar 7.4 Memori selama **Kode 7.9** berjalan. Variabel lokal di tumpukan panggilan (*stack*) memegang referensi ke objek di *heap*; *header* setiap objek menunjuk ke kelasnya; setiap kelas memiliki satu tabel metode (*vtable*) yang slot ke-*k*-nya memuat kode metode ke-*k* yang dapat di-override. Subkelas menyalin slot-slot induknya dan menimpa slot yang di-override-nya. 579
Gambar 7.5 Dua hierarki dengan tingkat dalam yang abstrak (nama bercetak miring). Kiri: bangun geometri, terbagi menjadi bangun terbuka dan tertutup. Kanan: mahasiswa, terbagi menjadi mahasiswa perkuliahan dan mahasiswa riset. 587
Gambar 7.6 Sebagian hierarki koleksi Java. Garis utuh dengan segitiga kosong adalah `extends`; garis putus-putus adalah `implements`. Kelas-kelas abstrak menyediakan kode bersama, antarmuka-antarmuka menyatakan kontraknya, dan `ArrayList` serta `LinkedList` adalah kelas konkret yang benar-benar dibuat oleh program. 594
Gambar 7.7 Mewarisi kelas konkret. `TreeDB` menerima larik dan pencacah milik `LinearDB` walaupun pohon tidak pernah memakainya, dan setiap `TreeDB` kini mengaku sebagai `LinearDB`. 597
Gambar 7.8 Kelas dasar yang rapuh. Kiri: `addAll` milik subkelas memanggil `addAll` milik superkelas, yang memanggil `add`, yang oleh *dynamic dispatch* dikirim kembali ke subkelas: setiap tag terhitung dua kali. Kanan: kelas komposisi memanggil metode objek yang sekadar dipegangnya; `add` miliknya sendiri tidak pernah dimasuki ulang. 602

Gambar 7.9 Berlian. `WashAndIron` memiliki dua jalur naik ke `Service`. Python melinearkan hierarki itu menjadi satu urutan resolusi metode, dan setiap panggilan `super()` berpindah ke kelas berikutnya dalam urutan itu (panah bernomor), sehingga `Service.steps` berjalan tepat satu kali. 607

Gambar 7.10 Empat kelas dalam dua paket. `Beta` sepaket dengan `Alpha`; `AlphaSub` berada di paket lain tetapi memperluas `Alpha`; `Gamma` sama sekali tidak berhubungan dengan `Alpha`. 611

Gambar 7.11 Diagram kelas UML mesin harga Profitina Laundry. `Service` dan `WeightService` abstrak; lima layanan konkret memperluas keduanya; `ExpressService` sekaligus *adalah* `Service` dan *membungkus* sebuah `Service` (belah ketupat kosong); sebuah `Order` memegang layanan dalam jumlah berapa pun (belah ketupat penuh). 621

Gambar 8.1 Pohon biner Soal 8.1. Simpul 4 hanya memiliki anak kanan (8) dan simpul 7 hanya memiliki anak kiri (9); setiap simpul dalam lainnya memiliki dua anak. 653

Gambar 8.2 Objek-objek di balik `List a = cons(2, cons(-5, cons(8, nil())))`. Setiap sel memiliki *head* dan referensi *tail*; *tail* yang terakhir adalah satu-satunya *list* kosong yang dipakai bersama. 657

Gambar 8.3 Diagram kelas UML untuk kelas-kelas *list*. `ListException` bersifat *unchecked* karena memperluas `RuntimeException`; setiap kelas soal di Subbab 8.5 hanya memakai `List`, tidak memperluasnya. 658

Gambar 8.4 Pohon contoh sebelum dan sesudah `t.triple()`: bentuknya sama dan setiap nilai dikalikan tiga. 671

Gambar 8.5 Kiri: `t` yang asli. Kanan: salinan segar `u`, tujuh objek baru. Tengah: salinan keliru `w`, satu objek baru yang referensinya menunjuk kembali ke dalam `t`. 681

Gambar 8.6 Pohon pencarian biner yang dipakai pada Soal 8.6 dan 8.7. Lintasan tebal ditempuh oleh `find(8)` dan `find(10)`; daun putus-putus 9 dan 12 ditambahkan oleh `insert(9)` dan `insert(12)`. 693

Gambar 8.7 Diagram kelas UML untuk Soal 8.18. Garis putus-putus dengan segitiga berongga berarti "*implements*"; `quote` adalah *default method* dari antarmuka itu. 745

Gambar 8.8 Hierarki layanan pada Soal 8.19. `Express` memperluas `Wash`, meng-*override* `pricePerKg` dan `describe`, dan memakai `super` untuk menjangkau versi milik `Wash`. 753

Gambar 8.9 Sebelum (kiri): dua salinan dari *field* dan metode yang sama, masing-masing kelas memakai kelas yang lain. Sesudah (kanan): satu `Fighter` abstrak, dan subkelas-subkelas yang hanya menambahkan apa yang berbeda. 761

Gambar 8.10 *Aliasing*. Atas: `LeakyOrder` dan pemanggil berbagi satu *list*, sehingga keduanya dapat mengubahnya. Bawah: `SafeOrder` menyalin *list* itu, sehingga perubahan yang kemudian dilakukan pemanggil tidak sampai ke pesanan. 768

Gambar 9.1 Panggilan-panggilan yang dilakukan `fibonacci(5)` rekursif standar: lima belas panggilan, karena submasalah yang sama diselesaikan berulang-ulang — `fib(3)` dua kali dan `fib(2)` tiga kali. 794

Gambar 9.2 Inti Collections Framework milik Java. Antarmuka ditandai «interface»; panah putus-putus berarti *mengimplementasikan* dan panah utuh berarti *memperluas*; kelas-kelas abu-abu adalah peninggalan sejarah. `Map` bukan `Collection`; `Arrays` dan `Collections` memuat metode statis pembantu. 800

Gambar 9.3 Tipe-tipe koleksi Python dan kelas dasar abstrak `collections.abc`, padanan antarmuka koleksi Java. 802

Gambar 9.4 List yang sama disimpan dengan dua cara. `ArrayList` menyimpan referensi dalam satu blok slot yang bersebelahan, dengan kapasitas cadangan di ujung; `LinkedList` menyimpan rantai simpul, masing-masing tertaut ke tetangganya. 807

Gambar 9.5 `HashSet` berisi empat nama. Kode hash setiap nama memilih salah satu dari 16 ember, sehingga `contains` hanya melihat satu ember; iterasi menelusuri ember menurut urutan indeks, yang menjelaskan urutan tercetak `[Sari, Budi, Agus, Rina]`. 816

Gambar 9.6 `TreeSet` berisi empat nama yang sama: pohon pencarian merah-hitam. Penelusuran in-order mengunjungi nama-nama menurut urutan; pencarian mengikuti satu lintasan dari akar. 817

Gambar 9.7 Tiga aturan untuk mengambil elemen. Antrean melepaskan elemen tertua (FIFO), tumpukan elemen terbaru (LIFO), dan antrean prioritas elemen terkecil, yang disimpan *binary heap* di indeks 0 lariknya. 826

Gambar 9.8 Hubungan subtipe di antara kelas (kiri) dan di antara tipe list (kanan). `List<Cat>` bukan subtipe `List<Animal>`, tetapi keduanya subtipe `List<? extends Animal>`, dan `List<Cat>`, `List<Animal>`, serta `List<Object>` semuanya subtipe `List<? super Cat>`.....850
Gambar 9.9 Perbandingan yang dilakukan satu `contains` ketika n tumbuh (kedua sumbu logaritmik): sekitar $n/2$ untuk list, sekitar $\log 2n$ untuk pohon seimbang, dan konstanta kecil untuk tabel hash dengan faktor muatan $\alpha \leq 0.75$. 873
Gambar 9.10 Memilih koleksi dengan empat pertanyaan. Setiap daun menyebut kelas Java dan tipe Python yang menjawabnya. 874
Gambar 9.11 Buku pesanan: empat struktur, satu kumpulan objek `LaundryOrder`. Kelas ini menjaga sebuah invarian: setiap pesanan muncul di `byId`, di list milik pelanggannya, dan tepat di satu himpunan status. 875
Gambar 10.1 Lapisan di bawah sebuah jendela. Di Java, panel Anda dibangun dari komponen Swing, yang bertumpu pada AWT, yang meminta sebuah jendela asli kepada sistem operasi. Di Python, `tkinter` adalah lapisan tipis kelas Python di atas toolkit Tcl/Tk. 909
Gambar 10.2 Sebagian hierarki kelas Swing/AWT (kiri) dan Tkinter (kanan). Kelas yang diarsir adalah yang dipakai bab ini. Panah menunjuk dari subkelas ke superkelasnya. 910
Gambar 10.3 Pohon komponen jendela pencatatan pesanan Profitina Laundry. Setiap kontainer hanya menata anak-anaknya sendiri, dengan layout manager-nya sendiri. 917
Gambar 10.4 Riwayat sebuah peristiwa. Sistem operasi mengubah tindakan pengguna menjadi peristiwa dan mengantrekannya; gelung peristiwa mengambilnya satu per satu, memanggil listener yang terdaftar untuknya, dan menggambar ulang saat senggang. Listener yang mengubah apa yang harus ditampilkan hanya *meminta* penggambaran ulang. 918
Gambar 10.5 Sebuah tombol dan listener-nya. Tombol menyimpan daftar; pendaftaran menambah ke daftar itu; sebuah klik memanggil `actionPerformed` setiap listener secara bergiliran. Tombol tidak pernah tahu apa yang dikerjakan listener-nya. 923
Gambar 10.6 Jendela gema setelah pengguna mengetik sapaan di field teks dan menekan Enter: listener telah menyalin teksnya ke label. 925
Gambar 10.7 Jendela kotak centang setelah *Mango* dicentang: listener item telah menulis ulang label penjelasan. 929
Gambar 10.8 Lima tombol dalam `FlowLayout`: masing-masing mempertahankan ukuran yang disukainya, dan barisnya mengalir dari kiri ke kanan. 932
Gambar 10.9 Lima wilayah `BorderLayout`. Wilayah tengah mengambil seluruh ruang yang disisakan keempat tepi. 932
Gambar 10.10 Dua kotak tegak di dalam sebuah kotak mendatar. Tombol radio di kiri tergabung dalam satu `ButtonGroup`, sehingga hanya satu yang dapat dipilih; kotak centang di kanan saling bebas. 933
Gambar 10.11 Sebuah `GridBagLayout` tempat tombol-tombol merentang satu sel, sisa satu baris, atau dua baris, sesuai `gridwidth`, `gridheight`, dan bobotnya. 934
Gambar 10.12 Sistem koordinat Swing: titik asal adalah pojok kiri atas dan y membesar ke bawah. Oval ditentukan oleh kotak pembatasnya, bukan oleh pusatnya. 943
Gambar 10.13 Jendela penggelap setelah cukup banyak klik pada *Darken*: kotaknya hitam, dan tombol *Darken* nonaktif karena tingkatnya tidak bisa lebih rendah lagi. 948
Gambar 10.14 Varian yang menggelapkan dengan tombol dan menerangkan dengan klik tetikus, lalu mengatakannya di sebuah label; di sini kotak sudah mencapai hitam dan label melaporkan bahwa ia tidak bisa digelapkan lagi. 949
Gambar 10.15 Apa yang diingat listener. Objek dalam dan objek anonim Java menyimpan referensi tersembunyi ke objek luarnya; objek lambda memegang salinan nilai yang ditangkapnya. Closure Python menyimpan *sel* yang merujuk ke variabel fungsi yang menciptakannya. (Nama-nama field adalah rincian implementasi kompilator.) 957
Gambar 10.16 Gambar hati yang dibuat dengan menyeret tetikus, dengan koordinat penunjuk ditampilkan di label. 964
Gambar 10.17 Aturan thread pada garis waktu. Atas: penghitungan berjalan di EDT, yang tidak dapat menangani klik atau penggambaran ulang selama 600 ms. Bawah: thread pekerja menjalankan penghitungan sementara EDT terus menangani peristiwa; `done()` kembali ke EDT untuk menampilkan hasilnya. 971

Gambar 10.18 Jendela pencatatan pesanan Swing dengan tiga pesanan, pesanan keempat sedang diketik (harga langsungnya sudah menunjukkan Rp68,250), dan baris kedua terpilih. 975
Gambar 10.19 Jendela pencatatan pesanan Tkinter dalam keadaan yang sama. (Tangkapan layar memperlihatkan isi jendela tanpa bilah judul yang ditambahkan sistem operasi.) 976
Gambar 10.20 Rancangan program pencatatan pesanan. View memiliki sebuah `OrderBook` dan memakai `OrderForm` untuk setiap perhitungan dan pemeriksaan; kelas-kelas model tidak pernah menyebut jendela. 977
Gambar 10.21 Program oval setelah sebuah klik: lingkaran kecil digambar di sekeliling titik yang diklik. 1001
Gambar 11.1 Meja depan yang sama sebelum dan sesudah MVC. Kiri: satu kelas jendela mengerjakan enam tugas. Kanan: model menyimpan pesanan dan aturannya, view menggambar, dan controller menerjemahkan gestur menjadi panggilan ke model; panah putus-putus adalah pemberitahuan Observer, cara model memberi tahu view bahwa sesuatu telah berubah. 1016
Gambar 11.2 Diagram MVC klasik. Model membungkus keadaan aplikasi, menjawab pertanyaan tentang keadaan itu, menyediakan fungsi aplikasi, dan memberi tahu view tentang perubahan; view menampilkan model dan mengirim gestur pengguna ke controller; controller memetakan tindakan pengguna menjadi perubahan model dan memilih view untuk tanggapannya. 1017
Gambar 11.3 Model, view, dan controller sebuah penyunting dokumen kecil. Modelnya lima nilai biasa; view-nya jendela properti yang menampilkan nilai-nilai itu; controller-nya menyediakan satu metode untuk setiap properti, yang dipanggil view ketika pengguna menyunting sebuah kolom. 1018
Gambar 11.4 Satu model, dua view. Jendela *Display* menggambar teks sebagaimana akan tampak di halaman, dan jendela *Properties* menampilkan nilai-nilai yang sama sebagai angka; keduanya dijaga tetap mutakhir oleh pemberitahuan perubahan dari model. 1019
Gambar 11.5 Controller sebagai mediator. Tindakan pengguna bergerak dari view ke controller, yang memperbarui properti model; perubahan properti model bergerak kembali ke controller, yang memperbarui komponen view. Model dan view tidak pernah saling merujuk. 1020
Gambar 11.6 Diagram kelas contoh mahasiswa. `StudentController` memegang sebuah model `Student` dan sebuah `StudentView`; ia memperbarui model melalui *setter* dan memperbarui view dengan `updateView()`; kelas demo `StudentMvcDemo` memakai controller dan mengambil data pertama dari basis data (tiruan). 1022
Gambar 11.7 Pola Observer sebagaimana dipakai `ObserverDemo.java`. Subjek `WashCounter` mendelegasikan daftar pendengarnya kepada sebuah `PropertyChangeSupport`, yang memberi tahu sebanyak apa pun objek yang mengimplementasikan `PropertyChangeListener`; kedua pengamatnya adalah lambda. Di Python peran yang sama dimainkan oleh list biasa berisi *callable*. 1028
Gambar 11.8 Diagram sekuens satu `increment()`. Pencacah memanggil `setLoads(1)` miliknya sendiri, yang meminta objek pendukung menembakkan satu peristiwa; objek pendukung memanggil setiap pendengar sesuai urutan pendaftaran, dan baru setelah itu `increment()` kembali ke `main`. 1034
Gambar 11.9 Pencacah Up/Down sedang bekerja. Baris atas, dimulai dari 0: setelah klik pada *Up* kolomnya menampilkan 1, 2, dan 10; setelah klik pada *Down* ia menampilkan 5 dan −15. Baris bawah, dimulai dari 100: 101, lalu 74. Penunjuk tetikus memperlihatkan tombol yang terakhir diklik. 1035
Gambar 11.10 Kelas-kelas papan tombol prediktif. Panel tombol melaporkan setiap tekanan kepada controller, yang memanggil salah satu dari empat operasi model; model memakai kamus dan memberi tahu ketiga view-nya, yang membaca teks dan kecocokan saat ini dari model. 1047
Gambar 11.11 Riwayat hidup sebuah pesanan binatu. Setiap `advance()` menggeser pesanan satu keadaan ke kanan; pesanan yang sudah diambil menolak untuk maju. 1070
Gambar 11.12 Diagram kelas meja depan. Model (kiri) tidak mengimpor apa pun dari GUI. Controller mengimplementasikan `OrderBookListener`, memanggil model, dan memperbarui tiga kontrak view; view Swing atau Tkinter sungguhan dan view teks uji mandiri sama-sama mengimplementasikan kontrak-kontrak itu. 1071
Gambar 11.13 Diagram sekuens sebuah klik pada *Add order*. Formulir melaporkan klik kepada controller, yang membaca formulir, mengurai berat, dan memanggil model; model memvalidasi,

menyimpan, dan memberi tahu controller, yang menyegarkan daftar dan ringkasan; setelah model kembali, controller mengosongkan formulir dan menampilkan pesan. 1087

Gambar 11.14 MVC dan dua keturunannya. Dalam MVC, view mengamati model dan mengirim gestur ke controller. Dalam MVP, view pasif melaporkan peristiwa kepada *presenter*, yang mengisi setiap kolom view. Dalam MVVM, view diikat, dua arah, pada sebuah *view model* yang memegang keadaan dan perintah layar. 1106

Gambar 12.1 Papan tombol telepon. Delapan tombol memuat huruf; tombol 7 dan 9 (diarsir) masing-masing memuat empat huruf, enam tombol lainnya tiga huruf; tombol 0 adalah spasi. 1125

Gambar 12.2 Diagram kelas UML proyek teks prediktif. Garis putus-putus dengan segitiga kosong berarti "mengimplementasikan"; panah padat berarti "memiliki" atau "memakai"; `DictionaryTreeImpl` merujuk objek dari kelasnya sendiri. 1127

Gambar 12.3 Kamus pohon untuk delapan kata. Setiap kata disimpan di setiap simpul pada lintasan signature-nya, sehingga simpul `46` memuat keempat kata yang diawali tombol 4 dan 6. 1170

Gambar 12.4 Mikrodetik per pencarian terhadap ukuran kamus, pada sumbu logaritmik, diukur dengan `TimingComparison` (kiri) dan `timing_comparison.py` (kanan). Prototipe naik bersama ukurannya; *map* tetap datar; *list* dan pohon berada di antaranya. 1182

Gambar 12.5 Pola Model–View–Controller. Controller menjadi perantara: ia mengubah tindakan pengguna menjadi pembaruan model, dan perubahan model sampai ke view. 1185

Gambar 12.6 Jendela versi awal program ini, dibangun ulang dengan label tombol yang diketik tangan, sebelum tombol apa pun ditekan. Label tombol 9 keliru: tertulis `vwxy`, bukan `wxyz`. 1186

Gambar 12.7 Jendela Swing dari **Kode 12.38** pada awal (kiri), sesudah menekan 4, 6, 6, 3 (tengah), dan sesudah `* * * 0 9 6 7 5 3 #` (kanan). Tombol 1 dinonaktifkan karena tidak memuat huruf. 1194

Gambar 12.8 Jendela Tkinter dari **Kode 12.43** pada ketiga saat yang sama seperti pada **Gambar 12.7**. 1200

Gambar 13.1 Kursor iterator berada di antara elemen. Setiap `next()` melompati satu elemen dan mengembalikannya (kotak abu-abu sudah dikembalikan); setelah lompatan terakhir, `hasNext()` bernilai `false`. 1259

Gambar 13.2 Mengapa Java melemparkan `ConcurrentModificationException`. Setelah `remove("a")` elemen-elemen bergeser ke kiri, tetapi kursor iterator tidak; tidak ada jawaban yang benar untuk "apa berikutnya?", maka iterator menolak menjawab. 1275

Gambar 13.3 Tiga jenis "salinan" dari list berisi orang. Aliasing: dua variabel, satu list. Salinan dangkal: dua list, tetapi slot-slotnya merujuk ke objek `Person` yang sama, sehingga penggantian nama lewat satu list terlihat lewat list yang lain. Salinan dalam: dua list dan objek `Person` yang terpisah. 1292

Gambar 13.4 Kunci yang diubah menjadi hilang. Entri disimpan di bucket 994 mod 16 = 2; hash kunci yang berubah mengirim pencarian ke bucket 1056 mod 16 = 0, yang kosong, sedangkan pencarian dengan nilai lama menemukan bucket yang benar tetapi gagal dalam uji `equals` 1315

Gambar 13.5 Pipeline filter–map–reduce. Muatan sumber mengalir melalui sebuah filter dan sebuah map menuju reduksi yang melipatnya menjadi satu bilangan; pipeline yang sama dalam satu baris Java dan satu baris Python. 1317

Gambar 13.6 Diagram kelas UML buku besar binatu. `Money`, `Price`, `OrderLine`, dan `Order` adalah record (nilai imutabel); `OrderHistory` memegang list pesanan yang tak dapat diubah, mengimplementasikan `Iterable<Order>`, dan memberikan dua jenis iterator. 1324

Gambar 14.1 Profitina Laundry sebagai sistem konkuren. Meja depan memasukkan pesanan ke satu antrean bersama; tiga mesin cuci, masing-masing sebuah thread pekerja, mengambil pesanan darinya; ketiganya membayar ke satu buku kas yang menerima satu mesin setiap saat. 1355

Gambar 14.2 Konkurensi dan paralelisme. Atas: satu inti; penjadwal memberi setiap thread irisan waktu pendek secara bergiliran, sehingga ketiganya maju tetapi pada setiap saat hanya satu yang berjalan. Bawah: dua inti; dua thread sungguh berjalan bersamaan, dan penjadwal memindahkan thread antarinti sesukanya. 1356

Gambar 14.3 Siklus hidup klasik sebuah thread Java. Thread lahir, menjadi siap ketika dimulai, berjalan ketika penjadwal memberinya prosesor, dan dapat meninggalkan keadaan berjalan untuk menunggu, tidur, atau terblokir oleh masukan dan keluaran sebelum siap lagi atau mati. 1371

Gambar 14.4 Enam nilai `Thread.State` dan peristiwa yang memindahkan thread di antaranya. `RUNNABLE` mencakup "siap" dan "berjalan" pada **Gambar 14.3**; `BLOCKED` berarti menunggu

kunci, `WAITING` berarti menunggu tanpa batas waktu, dan `TIMED_WAITING` berarti menunggu dengan batas waktu. 1371

Gambar 14.5 GIL dalam gambar. Atas: dua thread yang menghitung bergantian memegang GIL, sehingga keduanya tidak pernah menghitung pada saat yang sama. Bawah: thread yang lebih banyak menunggu melepas GIL selama menunggu, sehingga penantian mereka bertumpang tindih. 1378

Gambar 14.6 Pembaruan yang hilang, langkah demi langkah. Anak 1 membaca 0; Anak 2 membaca 0 dan menulis 1; Anak 1 menulis angka 1 miliknya sendiri di atasnya. Dua kenaikan sudah terjadi dan pencacah menunjukkan 1. Dua langkah kemudian sebuah penurunan juga hilang, dan pencacah berakhir di -1, bukan 0. 1383

Gambar 14.7 Penyangga melingkar lima sel ketika penuh. Kedua indeks menunjuk ke sel 2: nilai berikutnya bagi konsumen adalah angka 3 yang tersimpan di sana, dan nilai berikutnya dari produsen akan masuk ke sana begitu angka 3 itu diambil. 1394

Gambar 14.8 Keluaran `ProCon.java` yang ditampilkan di area teks Swing: produsen menulis 1 sampai 10 ke lima sel melingkar, menunggu ketika penyangga penuh, dan total milik konsumen adalah 55. 1398

Gambar 14.9 Para filsuf yang bersantap: lima filsuf, lima piring spageti, dan satu garpu di antara setiap pasangan tetangga. Filsuf Pi memerlukan garpu Fi dan $F(i + 1) \bmod 5$, seperti dalam `Philosopher.java`. 1404

Gambar 14.10 Garis waktu deadlock pada **Kode 14.21**. Pada $t1$ Ani mengambil mesin cuci, pada $t2$ Budi mengambil pengering, pada $t3$ dan $t4$ masing-masing meminta mesin milik yang lain dan terblokir selamanya. Graf tunggu di sebelah kanan memiliki siklus, dan siklus dalam graf tunggu tepat merupakan sebuah deadlock. 1412

Gambar 14.11 Livelock dan obatnya. Atas: dua pekerja yang selalu sopan mengoper sumber daya bolak-balik dan tidak pernah bekerja. Bawah: setiap pekerja mengalah paling banyak sekali, sehingga setelah dua penyerahan keduanya bekerja. 1423

Gambar 14.12 Diagram kelas binatu konkuren. Meja depan (`main`) memasukkan record `Order` ke sebuah `BlockingQueue`; tiga objek `WashingMachine`, masing-masing dijalankan oleh thread-nya sendiri, mengambil pesanan darinya dan mencatat harganya di satu-satunya `Ledger`, yang seluruhnya tersinkronisasi. 1428

Gambar 15.1 Satu permintaan yang berjalan melalui empat lapisan Internet. Lapisan-lapisan klien membungkus baris JSON dalam segmen TCP, paket IP, dan bingkai lapisan tautan; lapisan-lapisan server membukanya dalam urutan terbalik. Setiap lapisan tampak berbicara langsung dengan kembarannya di host lain. 1452

Gambar 15.2 Klien mengajukan permintaan koneksi ke port tempat server mendengarkan (di sini port 5050; port milik klien sendiri, misalnya 52100, adalah port efemeral yang dipilih sistemnya). 1456

Gambar 15.3 Setelah koneksi diterima, klien dan server terhubung melalui socket baru pada port 5050 milik server; socket pendengar milik server tetap terbuka untuk klien lain. 1456

Gambar 15.4 Riwayat hidup satu koneksi TCP. Server membuat socket pendengar dan terblokir di `accept()`; penyambungan klien memicu jabat tangan tiga langkah; `accept()` mengembalikan socket terhubung yang baru; baris-baris mengalir dua arah; `close()` klien mengirim FIN, yang dilihat server sebagai akhir aliran masukannya. 1458

Gambar 15.5 Protokol knock-knock sebagai mesin keadaan. Setiap panah berlabel "apa kata klien / apa jawaban server"; panah kelabu menangani masukan tak terduga; panah hijau memulai lelucon berikutnya. 1467

Gambar 15.6 Server multiklien. Thread penerima hanya menerima koneksi dan menyerahkan setiap socket baru kepada seorang pekerja dari kumpulan thread berukuran tetap; setiap pekerja memiliki satu socket dan satu objek protokol; satu-satunya keadaan yang dibagi para pekerja dijaga oleh pencacah atomik atau kunci. 1472

Gambar 15.7 Percakapan dengan layanan status pesanan. Permintaan berjalan sebagai baris JSON; server berkonsultasi dengan `OrderBook` bersama yang dijaga oleh kunci; baris yang rusak mendapat pesan galat, bukan kebisuan; `bye` mengakhiri percakapan. Klien dapat berupa Java dan server Python, atau sebaliknya. 1487

Gambar 15.8 Lima komponen proyek tim, pemiliknya, dan tiga dokumen antarmuka di antaranya. Panah putus-putus menunjukkan pengujian silang: setiap pemilik menulis uji untuk komponen yang memakai komponennya, atau yang dipakai oleh komponennya. 1536

Gambar 15.9 Cabang fitur. Tim memulai `main` dengan README dan dokumen protokol; setiap anggota bekerja di cabangnya; setiap cabang kembali ke `main` melalui *pull request* yang ditinjau; versi yang didemonstrasikan diberi *tag*.....1538
Gambar 16.1 Lima bagian Knock-Knock Club dan kontrak di antara mereka. Panah utuh berarti "memakai"; panah putus-putus menunjukkan bahwa anggota pengujian menguji setiap kontrak. ..1572
Gambar 16.2 Protokol *knock-knock* sebagai diagram alir: percakapan bagi manusia, hanya dengan jawaban-jawaban yang diharapkan.....1582
Gambar 16.3 Protokol yang sama sebagai mesin keadaan. Setiap keadaan memiliki panah untuk setiap masukan yang mungkin, termasuk "masukan lain".....1582
Gambar 16.4 *Thread-thread* server *knock-knock*. Setiap klien memiliki *thread* pekerja dan objek protokolnya sendiri; hanya objek imutabel atau aman-*thread* yang dipakai bersama.....1591
Gambar 16.5 Uji silang melalui antarmuka. Anggota klien menulis uji untuk kode server; anggota server menulis uji untuk kode klien. Masing-masing memakai *stub* dari sisinya sendiri.....1605
Gambar 16.6 Jendela Knock-Knock Club (Swing) selama percakapan. Baris status berbunyi "Connected"; kolom masukan aktif karena server sedang menunggu jawaban.....1627
Gambar 16.7 Pembaruan yang hilang. Kedua mesin membaca 41 sebelum salah satunya menulis, sehingga keduanya menulis 42: dua muatan dicuci, dan pencacah hanya naik satu.....1664
Gambar 16.8 Lini cuci: satu produsen, penyangga terbatas berkapasitas 2, dan dua konsumen. Dua pil `STOP`, satu per mesin, mengakhiri hari kerja.....1672
Gambar 16.9 Diagram kelas UML proyek puncak. Segitiga berongga berarti "mewarisi" (garis utuh) atau "mengimplementasikan" (garis putus-putus); belah ketupat berarti toko menyimpan pesanan-pesanan.....1693

Daftar Tabel

Tabel 1.1 Benda-benda di kios *laundry*, masing-masing digambarkan oleh apa yang diketahuinya dan apa yang dapat dilakukannya. 6
Tabel 1.2 Java dan Python berdampingan: perbedaan-perbedaan yang pertama kali dijumpai pemula.....18
Tabel 1.3 Perkakas Java Development Kit yang dipakai buku ini. 19
Tabel 1.4 Empat lingkungan pengembangan untuk Java dan Python. 21
Tabel 1.5 Meja kerja di tiga sistem operasi, dari komputer kosong hingga program berjalan dalam kedua bahasa. Di Fedora gunakan `sudo dnf install java-25-openjdk-devel python3.14`; di Arch, `sudo pacman -S jdk-openjdk python`.....23
Tabel 1.6 Masalah umum saat menyiapkan Java, Python, dan VS Code, serta cara memperbaikinya. 32
Tabel 1.7 Keenam belas bab buku ini dan apa yang ditambahkan masing-masing. 42
Tabel 2.1 Lima tugas setiap IDE, dan subbab yang membahas masing-masing. 57
Tabel 2.2 Gagasan yang sama dengan dua nama: kamus antara Eclipse dan VS Code. 58
Tabel 2.3 Pintasan papan ketik yang layak dipelajari pada minggu pertama, di Eclipse dan di VS Code.....91
Tabel 2.4 Kendali eksekusi debugger beserta tombol dan perintahnya di Eclipse, VS Code, jdb, dan pdb. 95
Tabel 2.5 Tagihan Bu Sari yang dihitung secara manual. Inilah nilai-nilai yang harus dicapai program, butir demi butir. 104
Tabel 3.1 Kelas dipandang sebagai lembar kerja: baris judul adalah kelas `Dog`, dan setiap baris di bawahnya adalah satu objek `Dog`.....136
Tabel 3.2 Bagian-bagian kelas dalam Java dan dalam Python. 139
Tabel 3.3 Identitas, kesetaraan, dan "tidak ada objek" dalam kedua bahasa.....150
Tabel 3.4 Operasi string sehari-hari dalam Java dan Python. 154
Tabel 3.5 Penampung format yang dipakai dalam buku ini. 158
Tabel 3.6 Delapan tipe primitif Java, dan tipe Python yang memainkan peran masing-masing.....161

Tabel 3.7 Operator yang umum, dari prioritas tertinggi sampai terendah. Di Python, `**` mengikat lebih erat daripada minus uner, sehingga `-2 ** 2` bernilai `-4`. 166
Tabel 3.8 Pembagian dan pembulatan dalam Java dan Python, berdampingan. 172
Tabel 3.9 Menelusuri rantai `else if` pada **Kode 3.33** untuk $k = 2, \ldots, 6$ 175
Tabel 3.10 Bentuk-bentuk perulangan Java dan Python, dan kapan memakai masing-masing. 184
Tabel 3.11 Menelusuri **Kode 3.43** untuk $n = 4071$; perulangan berakhir ketika n mencapai 0. 186
Tabel 3.12 Satuan massa imperial dan nilainya dalam pon. 188
Tabel 3.13 Menghitung Persamaan (3.8) dengan tangan untuk lima tahun pertama. 193
Tabel 3.14 Memeriksa keempat harga pada **Kode 3.55** dengan Persamaan (3.9) dan Persamaan (3.10). 205
Tabel 3.15 Nilai ungkapan-ungkapan pada Soal Latihan 3.5. 212
Tabel 3.16 Menelusuri perulangan Soal Latihan 3.6 untuk $x = 48$, $y = 18$ 212
Tabel 4.1 Keempat belas soal dalam bab ini dan perkakas dari Bab 1 sampai 3 yang paling banyak dilatih oleh masing-masing soal. 219
Tabel 4.2 Keenam waktu uji yang dikerjakan dengan tangan: jarum jam, jarum menit, selisih keduanya, selisih yang dibawa ke 0,360, dan hasil yang dibulatkan. 230
Tabel 4.3 Kata sandi dua akun setelah setiap langkah uji; baris terakhir memperlihatkan bahwa setiap objek memiliki *field*-nya sendiri. 237
Tabel 4.4 Pertarungan Soal 4.7 yang dikerjakan dengan tangan: setiap serangan merugikan pihak bertahan sebesar (kekuatan penyerang − pertahanan pihak bertahan) 264
Tabel 4.5 Mengelompokkan angka-angka 1250000 dari kanan: sebuah titik ditambahkan setelah setiap angka ketiga, kecuali setelah angka paling kiri. 272
Tabel 4.6 Jejak perulangan jumlah angka untuk $n = 4071$: empat lintasan, satu per angka, berakhir ketika n mencapai 0. 304
Tabel 4.7 Jejak algoritma Euclid untuk $a = 84$, $b = 36$: setelah dua lintasan $b = 0$ dan jawabannya $a = 12$ 304
Tabel 4.8 Keempat belas soal dalam bab ini, gagasan rancangan yang diajarkan masing-masing, dan jebakan yang diperagakan penyelesaian kelirunya. 316
Tabel 5.1 Operasi sehari-hari pada larik Java, `ArrayList` Java, dan `list` Python. 338
Tabel 5.2 Memilih antara larik dan list. 341
Tabel 6.1 Tag Javadoc yang paling sering dipakai, dan tempat keterangan yang sama dalam *docstring* Python. 419
Tabel 6.2 Gagasan pengujian yang sama dalam JUnit 6, `unittest`, dan pytest. 437
Tabel 6.3 Kelas ekuivalensi dan nilai batas untuk harga per kilogram dengan minimum 3 kg. 437
Tabel 6.4 *Overloading* di Java dan perkakas Python yang mengerjakan tugas yang sama. 457
Tabel 6.5 Antarmuka dari pustaka Java yang muncul paling awal, dan apa yang memainkan peran yang sama di Python. 475
Tabel 6.6 Empat cara menyatakan "setiap kelas jenis ini wajib menyediakan metode-metode ini". 487
Tabel 6.7 Kelima aturan harga Profitina Laundry. 501
Tabel 7.1 Perbandingan overloading dan overriding. 570
Tabel 7.2 Apa yang diputuskan kapan: hanya metode instans yang dapat di-override yang di-dispatch berdasarkan tipe dinamis. 578
Tabel 7.3 Kelas abstrak dan antarmuka di Java. 605
Tabel 7.4 Tingkat akses Java. *Kelas*: kode di kelas yang sama; *Paket*: kode di kelas mana pun dalam paket yang sama; *Subkelas*: kode di subkelas yang dideklarasikan di paket lain; *Dunia*: kode lain mana pun. 610
Tabel 7.5 Keterlihatan anggota `Alpha` dari setiap kelas pada **Gambar 7.10**. 611
Tabel 7.6 Kontrol akses di Java dan padanan konvensionalnya di Python. 615
Tabel 8.1 Metode delapan langkah untuk soal perancangan dengan banyak kelas. Langkah 1 sampai 6 adalah perancangan, Langkah 7 adalah spesifikasi, dan Langkah 8 adalah implementasi. 652
Tabel 8.2 Kesembilan pemanggilan konstruktor pada Soal 8.1 dalam urutan dari bawah ke atas. Sebuah simpul baru dapat dibangun setelah kedua subpohonnya ada. 666
Tabel 8.3 Resep delegasi yang diterapkan pada `triple()` 671

Tabel 8.4 Ketujuh pemanggilan `printDepthFirst()` pada pohon contoh, dalam urutan terjadinya. Membaca kolom tengah dari atas ke bawah menghasilkan keluarannya. 676
Tabel 8.5 Resep delegasi untuk salinan dalam. Berbeda dengan `triple()`, hasil dari pemanggilan yang didelegasikan justru menjadi inti persoalannya. 681
Tabel 8.6 Pemanggilan-pemanggilan `saveDepthFirst` pada pohon contoh yang dimulai dari posisi 4. Setiap pemanggilan mengembalikan posisi terakhir yang ditulisnya; pemanggil menambahkan satu untuk menemukan sel bebas berikutnya. 688
Tabel 8.7 Kasus-kasus `find(n)` pada BST. Hanya satu dari kedua subpohon yang pernah ditanya. 693
Tabel 8.8 `insert` adalah `find` dengan kasus terakhir yang berbeda: di tempat pencarian akan menyerah, daun baru dipasangkan. 698
Tabel 8.9 Rancangan `negateAll`: sebuah *map* atas *list*, satu elemen keluaran untuk setiap elemen masukan. 705
Tabel 8.10 Rancangan `find`: posisi yang ditemukan di *tail* digeser satu; sinyal "tidak ditemukan" tidak 708
Tabel 8.11 Rancangan `allPositive`: operasi AND logis atas semua elemen, dengan `true` sebagai nilai awal yang netral. 711
Tabel 8.12 Rancangan `positives`: sebuah *filter* menjaga urutan karena setiap *head* yang dipertahankan diletakkan di depan *tail* yang sudah disaring. 714
Tabel 8.13 Rancangan `sorted`: setiap pemanggilan membandingkan satu pasangan bersebelahan dan mendelegasikan sisanya. 717
Tabel 8.14 Rancangan `merge`: setiap pemanggilan memindahkan *head* yang lebih kecil ke hasil, sehingga setiap pemanggilan membuang satu elemen dari salah satu *list*. 720
Tabel 8.15 Rancangan `skip`: sebuah *filter* dengan uji "tidak sama dengan `x`" 723
Tabel 8.16 Rancangan `removeDuplicates`: *head* bertahan, salinan-salinannya yang datang kemudian dilewati, dan rekursi menangani sisanya. 726
Tabel 8.17 Rancangan pengurai pesanan: setiap jenis kegagalan dilaporkan oleh metode yang mendeteksinya dan ditangani oleh metode yang dapat berbuat sesuatu terhadapnya. 730
Tabel 8.18 Rancangan `GrowableIntArray`. Prasyarat `get` diperiksa terhadap `size`, bukan terhadap panjang larik. 738
Tabel 8.19 Bagaimana program-program yang memeriksa dirinya sendiri dalam bab ini dipetakan ke JUnit 6 dan `unittest`. 752
Tabel 8.20 Penelusuran `ServiceDispatch.main` dengan tangan. *Overload* dipilih dari tipe yang dideklarasikan; metode yang di-*override* dipilih dari kelas objeknya. 756
Tabel 8.21 Masalah-masalah rancangan pada kelas `Player` dan `Enemy` yang berulang, dan bagaimana *refactoring* memperbaiki masing-masing. 760
Tabel 8.22 Keempat *bug* enkapsulasi pada `LeakyOrder`. Setiap *bug* adalah jalan menuju keadaan objek yang melewati pemeriksaan. 768
Tabel 8.23 Kedua puluh satu soal dalam bab ini: apa yang dilatih masing-masing dan satu kasus yang membuat penyelesaian keliru yang menggoda gagal. 775
Tabel 9.1 Perbandingan tiga keluarga koleksi. 801
Tabel 9.2 Koleksi-koleksi bab ini dalam Java dan dalam Python. 803
Tabel 9.3 Apa yang boleh dipegang setiap jenis deklarasi list, dan apa yang boleh Anda baca dan tulis melaluinya. 850
Tabel 9.4 Pemrograman generik di Java dan di Python. 860
Tabel 9.5 Biaya operasi-operasi utama kelas koleksi Java untuk n elemen. 872
Tabel 9.6 Biaya operasi-operasi utama koleksi Python untuk n elemen. 873
Tabel 10.1 Komponen Swing yang umum dan padanannya di Tkinter. 911
Tabel 10.2 Empat jenis peristiwa utama: cara mendengarkannya di Swing dan di Tkinter. 931
Tabel 10.3 Layout manager Swing dan geometry manager Tkinter untuk kebutuhan umum. 942
Tabel 10.4 Cara-cara menulis listener atau callback, dan apa yang diingat masing-masing. 961
Tabel 10.5 Perkakas aturan thread di Swing dan di Tkinter. 974
Tabel 11.1 Tanggung jawab dan ketergantungan yang diizinkan bagi ketiga peran. 1020
Tabel 11.2 Pola Observer dalam kedua bahasa 1034
Tabel 11.3 Apa yang dilakukan setiap tombol papan tombol terhadap model pesan. 1047

Tabel 12.1 Enam langkah untuk soal yang memadukan koleksi, antarmuka, dan antarmuka pengguna. Langkah 1 sampai 3 memilih struktur data, Langkah 4 dan 5 membentuk GUI, dan Langkah 6 memeriksa pilihannya. 1125
Tabel 12.2 Empat bagian proyek teks prediktif dan bab-bab yang gagasannya dipakai setiap bagian. 1127
Tabel 12.3 Gelung `wordToSignature("Home")`, satu karakter per langkah. *Buffer* bertambah satu angka pada setiap langkah dan diubah menjadi `String` sekali saja, di akhir. 1128
Tabel 12.4 Pencarian biner untuk 4663 di antara 93 kartu terurut dari `words.txt`. Empat tengokan menemukan satu kecocokan; sisanya terletak di indeks 38 sampai 45, di sekitarnya. 1143
Tabel 12.5 Waktu berlalu setiap program baris perintah untuk 53 signature dari `sample_text.txt` dengan kamus 200.000 kata, sebagaimana dilaporkan `time`. Setiap eksekusi mencakup memulai program dan membangun kamusnya. 1162
Tabel 12.6 Resep delegasi dari Bab 8 yang diterapkan pada dua metode rekursif kamus pohon. 1171
Tabel 12.7 Keempat kamus dibandingkan. n adalah banyaknya kata, m panjang rata-ratanya, k banyaknya kecocokan, ℓ panjang signature, dan Ws banyaknya kata yang tersimpan di simpulnya. 1183
Tabel 12.8 Tiga *field* model (kata-kata yang selesai, kandidat, indeks kandidat yang ditampilkan) dan pengaruh setiap tombol terhadapnya. Tidak ada signature yang pernah disimpan. 1185
Tabel 12.9 Tiga tugas loket pengambilan dan koleksi-koleksi yang dapat mengerjakannya. Biayanya per operasi kecuali disebutkan lain. 1202
Tabel 12.10 Keenam contoh pesanan dalam urutan papan, beserta kunci pengurutan masing-masing. Pesanan 105 dan 102 seri pada dua kunci pertama dan dipisahkan oleh nama pelanggan. 1216
Tabel 12.11 Keenam belas soal dalam bab ini: apa yang dilatih masing-masing dan kasus yang keliru ditangani penyelesaian keliru yang menggoda. 1242
Tabel 13.1 Protokol iterasi di Java dan Python. 1261
Tabel 13.2 Cara-cara aman untuk mengubah koleksi yang sedang ditelusuri. 1281
Tabel 13.3 Alat-alat imutabilitas di Java dan Python. 1304
Tabel 13.4 Operasi stream di Java dan padanan lazimnya di Python. 1323
Tabel 14.1 Kosakata dasar thread di Java dan Python. 1370
Tabel 14.2 Alat-alat sinkronisasi Java dan Python yang dipakai dalam bab ini. 1393
Tabel 14.3 Cara-cara mencegah deadlock, dan syarat Coffman yang dipatahkan masing-masing. 1417
Tabel 15.1 Perbandingan TCP dan UDP. 1453
Tabel 15.2 Panggilan socket TCP di Java dan Python, dalam urutan pemakaiannya oleh program. 1457
Tabel 15.3 Protokol knock-knock sebagai tabel keadaan dan pesan. Pencocokan mengabaikan huruf besar dan kecil. 1466
Tabel 15.4 Protokol status pesanan. Satu objek JSON per baris ke setiap arah; klien berbicara lebih dahulu; setiap permintaan menerima tepat satu tanggapan. 1486
Tabel 15.5 Kode status HTTP yang umum. Kode 2xx berarti berhasil, 4xx masalah pada permintaan, 5xx masalah di server. 1522
Tabel 15.6 Kewajiban utama UU PDP No. 27/2022, dan bagaimana masing-masing tampak dalam sebuah layanan jaringan kecil. 1528
Tabel 15.7 Perintah Git yang diperlukan anggota tim pada minggu pertama. 1538
Tabel 15.8 Rencana tonggak contoh proyek, satu hasil kerja per minggu, masing-masing diperiksa oleh orang lain selain pembuatnya. 1541
Tabel 16.1 Perkakas yang ditambahkan Bab 13 sampai Bab 15, dalam kedua bahasa. Setiap soal dalam bab ini memakai beberapa di antaranya. 1567
Tabel 16.2 Metode sepuluh langkah untuk proyek tim, dan soal dalam bab ini tempat setiap langkah dilaksanakan. 1569
Tabel 16.3 Pembagian tuntutan tugas proyek ke dalam Soal 16.1 sampai 16.8. 1571
Tabel 16.4 Spesifikasi dan uraian tugas setiap bagian Knock-Knock Club. Dengan tiga anggota, anggota 2 dan 3 masing-masing memiliki dua bagian. 1572
Tabel 16.5 Tabel transisi protokol *knock-knock*. Setiap keadaan memiliki baris untuk "masukan lain", yang tidak tampak pada diagram alir. 1583
Tabel 16.6 Apa yang diamati setiap sisi ketika sisi lain gagal, di Java dan di Python, dan bagaimana Knock-Knock Club bereaksi. 1619

Tabel 16.7 Dokumentasi produk Knock-Knock Club, satu baris per dokumen. 1643
Tabel 16.8 Pemecahan masalah Knock-Knock Club: gejala seperti yang dilihat pengguna, penyebab, dan obatnya. 1643
Tabel 16.9 Protokol LAUNDRY/1. Kata perintah tidak membedakan huruf besar dan kecil; setiap baris permintaan mendapat tepat satu baris balasan. 1682
Tabel 16.10 Tempat gagasan-gagasan seluruh buku muncul di proyek puncak. 1693
Tabel 16.11 Keempat belas soal dalam bab ini: apa yang dilatih masing-masing dan satu-satunya kasus yang disalahkan penyelesaian keliru yang menggoda. 1718
Tabel 16.12 Seluruh buku dalam satu tabel: pertanyaan setiap bab dan tempat jawabannya dipakai dalam bab ini. 1719
Tabel 16.13 Pola-pola rancangan yang dipakai dalam buku ini, menurut namanya. 1720

Daftar Kode

Kode 1.1 `LaundryLedger.java`: buku catatan *laundry* sebagai empat larik paralel dan dua fungsi yang berdiri sendiri, serta dua baris ceroboh yang merusaknya. 10
Kode 1.2 `laundry_ledger.py`: buku catatan yang sama sebagai empat *list* Python paralel. 10
Kode 1.3 `Hello.java`: program Java pertama, yang menyapa pembaca dan melaporkan versi Java yang menjalankannya. 27
Kode 1.4 `hello.py`: program Python pertama, yang melakukan hal yang sama dengan **Kode 1.3**. 28
Kode 1.5 `Loads.java`: menghitung muatan mesin untuk pesanan-pesanan hari ini, program kecil dengan perulangan untuk diamati di *debugger*. 30
Kode 1.6 `loads.py`: perhitungan yang sama dalam Python; `//` adalah pembagian bilangan bulat Python. 30
Kode 1.7 `Receipt.java`: struk yang dihitung dengan variabel-variabel dari empat tipe. 33
Kode 1.8 `receipt.py`: struk yang sama dalam Python, tempat nilai-nilai menentukan tipenya sendiri. 33
Kode 1.9 `Order.java`: pesanan *laundry* yang menyimpan keadaannya sendiri, menghitung harganya sendiri, dan memajukan statusnya hanya bila hal itu masuk akal. 35
Kode 1.10 `order.py`: kelas `Order` yang sama dalam Python. 37
Kode 1.11 `OrderDemo.java`: menciptakan dua pesanan dan mengirimi mereka pesan, termasuk pengambilan yang terlalu dini dan ditolak. 39
Kode 1.12 `order_demo.py`: program yang sama dalam Python, yang mengimpor kelasnya dari `order.py`. 39
Kode 1.13 `ShopDay.java`: sehari di Profitina Laundry, memakai kelas `Order` tanpa perubahan; kompilasi bersama `Order.java`. 44
Kode 1.14 `shop_day.py`: hari yang sama dalam Python; program ini mengimpor `Order` dari `order.py` di folder yang sama. 44
Kode 1.15 `WashingMachine.java`: mesin cuci yang menolak muatan berlebih, pengisian ketika sedang berputar, dan penyalaan ketika kosong. 50
Kode 1.16 `washing_machine.py`: kelas yang sama dalam Python; blok di bawah `if __name__ == "__main__":` hanya berjalan ketika berkas itu dijalankan langsung, bukan ketika diimpor. 51
Kode 2.1 `Add.java`: metode dua baris, cukup kecil untuk dibaca bytecode-nya. 59
Kode 2.2 `add_bytecode.py`: menjalankan sebuah fungsi, lalu mencetak bytecode yang dibuat Python untuknya. 61
Kode 2.3 `MyFirstClass.java` dalam paket `myfirstproject`: program pertama proyek Eclipse. 75
Kode 2.4 `my_first_module.py`: program pertama yang sama dalam Python. 76
Kode 2.5 `Counter.java` dalam paket `util`: menjumlahkan $1 + 2 + \cdots + x$ untuk $1 \leq x \leq 255$. 83
Kode 2.6 `Tester.java` dalam paket `main`: pengujian kecil buatan sendiri untuk `Counter`. 84
Kode 2.7 `util/counter.py`: kembaran Python bagi `Counter.java`. 85

Kode 2.8 `main/tester.py`: kembaran Python bagi `Tester.java` 85
Kode 2.9 `LaundryPrice.java`: kalkulator harga kasir, lengkap dengan sebuah bug. 101
Kode 2.10 `laundry_price.py`: kalkulator yang sama dalam Python, dengan bug yang sama. 102
Kode 2.11 `LaundryPriceFixed.java`: biaya tambahan ekspres kini dikenakan pada harga butir itu sendiri 111
Kode 2.12 `laundry_price_fixed.py`: perbaikan yang sama dalam Python. 112
Kode 2.13 `DailyOrders.java`: pesanan sehari, kurang Rp 6.000. 115
Kode 2.14 `daily_orders.py`: pesanan hari yang sama dalam Python, dengan kekurangan yang sama. 115
Kode 2.15 `UnknownService.java`: pesanan yang menyebut layanan yang tidak dikenal daftar harga. 118
Kode 2.16 `unknown_service.py`: pesanan yang sama dalam Python. 118
Kode 2.17 `CounterTwice.java`: sebuah field mempertahankan nilainya di antara pemanggilan; variabel lokal dimulai dari awal. 125
Kode 2.18 `counter_twice.py`: perbandingan yang sama dalam Python, dengan sebuah atribut dan sebuah variabel lokal. 125
Kode 2.19 `LaundryReceipt.java`: struk terperinci dengan tambahan layanan dry cleaning. 126
Kode 2.20 `laundry_receipt.py`: struk yang sama dalam Python; pemformatan `%` memakai kode-kode yang sama dengan `printf` di Java. 127
Kode 3.1 `Hello.java`: program Java lengkap yang paling kecil. 131
Kode 3.2 `hello.py`: program yang sama dalam Python, dengan fungsi `main` yang lazim. 131
Kode 3.3 `Hello2.java`: mencetak kata pertama yang diketik di baris perintah. 132
Kode 3.4 `hello2.py`: kata-kata baris perintah tersimpan dalam `sys.argv`, sesudah nama skrip. 133
Kode 3.5 `CommentDemo.java`: komentar Javadoc, komentar baris, dan komentar blok. 134
Kode 3.6 `comment_demo.py`: docstring dan komentar baris dalam Python. 134
Kode 3.7 `Dog.java`: empat *field* privat, satu konstruktor, sebuah *getter* dan *setter* untuk setiap *field*, serta `toString`. 136
Kode 3.8 `dog.py`: kelas `Dog` dalam Python; *field* dibuat di dalam `__init__`. 139
Kode 3.9 `DogDemo.java`: membuat dua objek `Dog`, mencetaknya, dan mengubahnya melalui *setter*. 142
Kode 3.10 `dog_demo.py`: peragaan yang sama dalam Python. 142
Kode 3.11 `DogCheck.java`: pengujian buatan tangan untuk `getAge` dan `setWeight`. 143
Kode 3.12 `dog_check.py`: pemeriksaan yang sama dalam Python. 144
Kode 3.13 `ReferenceDemo.java`: menyalin primitif berarti menyalin nilai; menyalin referensi berarti menyalin panah. 147
Kode 3.14 `reference_demo.py`: dalam Python, setiap variabel adalah nama yang terikat ke sebuah objek. 149
Kode 3.15 `StringBasics.java`: `startsWith`, `substring`, `charAt`, `length`, dan metode `String` sehari-hari lainnya. 151
Kode 3.16 `string_basics.py`: operasi yang sama dengan `str` milik Python. 153
Kode 3.17 `StringImmutability.java`: metode string mengembalikan string baru; `==` membandingkan panah, sedangkan `equals` membandingkan karakter. 154
Kode 3.18 `string_immutability.py`: string Python juga imutabel; `==` membandingkan isi dan `is` membandingkan identitas 156
Kode 3.19 `StringFormatting.java`: `printf` dan `String.format` dengan lebar, desimal, dan pemisah ribuan 157
Kode 3.20 `string_formatting.py`: pemformatan yang sama dengan f-string dan `str.format` milik Python 157
Kode 3.21 `ParseNumbers.java`: mengubah teks baris perintah menjadi `int` dan `double`. 159
Kode 3.22 `parse_numbers.py`: `int(...)` dan `float(...)` mengurai teks di Python 159
Kode 3.23 `StaticTyping.java`: di Java setiap variabel memiliki satu tipe seumur hidupnya; `var` membiarkan kompilator menyimpulkan tipenya. 162
Kode 3.24 `dynamic_typing.py`: di Python sebuah nama dapat merujuk ke nilai-nilai bertipe berbeda; tipe diperiksa ketika operasi dijalankan. 162

Kode 3.25 `CharBoolean.java`: di balik layar, `char` adalah bilangan kecil; `&&`, `||`, dan `!` menggabungkan nilai kebenaran. 163
Kode 3.26 `char_boolean.py`: `ord` dan `chr` mengonversi antara karakter dan kodenya; `and`, `or`, dan `not` menggabungkan nilai kebenaran. 164
Kode 3.27 `IntegerTypes.java`: rentang tipe-tipe bilangan bulat, dan luapan di tengah perhitungan. 166
Kode 3.28 `integer_types.py`: bilangan bulat Python tidak pernah meluap; `wrap_int32` meniru `int` 32-bit milik Java. 168
Kode 3.29 `FloatingPoint.java`: menjumlahkan 0,1 sepuluh kali, dan mengapa bilangan desimal tidak boleh dibandingkan dengan `==`. 168
Kode 3.30 `floating_point.py`: efek yang sama di Python, dengan `math.isclose` dan modul `decimal`. 170
Kode 3.31 `IntegerDivision.java`: pembagian bilangan bulat, operator sisa bagi, dan konversi tipe. 171
Kode 3.32 `integer_division.py`: Python memisahkan pembagian sejati `/` dari pembagian lantai `//`. 171
Kode 3.33 `IfDemo.java`: sebuah `if`/`else`, dan rantai `else if` yang dicoba untuk beberapa nilai. 173
Kode 3.34 `if_demo.py`: `if`, `elif`, dan `else` dalam Python. 174
Kode 3.35 `SwitchDemo.java`: *fall-through* tanpa `break`, perbaikannya dengan `break` dan `default`, serta bentuk panah. 176
Kode 3.36 `switch_demo.py`: `match` (Python 3.10 ke atas), rantai `elif`, dan pencarian dalam *dictionary*. 178
Kode 3.37 `LoopBasics.java`: perulangan `while` yang sudah diperbaiki, perulangan `for` yang setara, dan perulangan `do-while`. 179
Kode 3.38 `loop_basics.py`: perulangan `while` Python, perulangan `for` atas `range`, dan pola `while True` yang menggantikan `do-while`. 180
Kode 3.39 `LoopSequences.java`: empat perulangan `for`, satu untuk setiap deret. 181
Kode 3.40 `loop_sequences.py`: deret-deret yang sama dengan `range`. 182
Kode 3.41 `ArgsEcho.java`: mengunjungi setiap kata baris perintah melalui indeks dan dengan perulangan for-each. 183
Kode 3.42 `args_echo.py`: hal yang sama dalam Python, dengan `enumerate` untuk indeks dan nilai sekaligus. 183
Kode 3.43 `TraceDemo.java`: perulangan `while` yang membongkar sebuah bilangan digit demi digit. 185
Kode 3.44 `trace_demo.py`: perulangan yang sama dalam Python, dengan `//` untuk pembagian bilangan bulat. 185
Kode 3.45 `CircleArea.java`: luas lingkaran berjari-jari 5. 187
Kode 3.46 `circle_area.py`: perhitungan yang sama dalam Python. 187
Kode 3.47 `ImperialToKg.java`: mengubah 11 *stone* dan 6 pon menjadi kilogram. 188
Kode 3.48 `imperial_to_kg.py`: konversi yang sama dalam Python. 189
Kode 3.49 `Investment.java`: Persamaan (3.8) dihitung untuk enam lama investasi, satu demi satu. 191
Kode 3.50 `investment.py`: penyelesaian berulang yang sama dalam Python. 192
Kode 3.51 `InvestmentLoop.java`: saldo dihitung tahun demi tahun dalam sebuah perulangan. 193
Kode 3.52 `investment_loop.py`: versi perulangan dalam Python. 194
Kode 3.53 `Customer.java`: pelanggan laundry dengan nomor telepon tersamar dan penghitung kunjungan. 197
Kode 3.54 `Service.java`: layanan laundry yang menghitung berat yang ditagihkan, harga, dan lama pengerjaan. 198
Kode 3.55 `LaundryDemo.java`: menghitung harga empat kantong cucian untuk dua pelanggan. 200
Kode 3.56 `customer.py`: kelas `Customer` dalam Python. 201
Kode 3.57 `service.py`: kelas `Service` dalam Python. 202
Kode 3.58 `laundry_demo.py`: program peragaan dalam Python. 203
Kode 3.59 `WashingMachine.java`: penyelesaian Soal Latihan 3.3. 209

Kode 3.60 `washing_machine.py`: penyelesaian Soal Latihan 3.3 dalam Python. 210
Kode 3.61 `Initials.java`: penyelesaian Soal Latihan 3.4. 210
Kode 3.62 `initials.py`: penyelesaian Soal Latihan 3.4 dalam Python. 211
Kode 3.63 `MultiplicationTable.java`: penyelesaian Soal Latihan 3.7. 212
Kode 3.64 `multiplication_table.py`: penyelesaian Soal Latihan 3.7 dalam Python. 213
Kode 3.65 `DoublingYears.java`: penyelesaian Soal Latihan 3.8. 213
Kode 3.66 `doubling_years.py`: penyelesaian Soal Latihan 3.8 dalam Python. 214
Kode 3.67 `PriceTable.java`: penyelesaian Soal Latihan 3.9. 214
Kode 3.68 `price_table.py`: penyelesaian Soal Latihan 3.9 dalam Python. 215
Kode 3.69 `BanknoteBreakdown.java`: penyelesaian Soal Latihan 3.10. 216
Kode 3.70 `banknote_breakdown.py`: penyelesaian Soal Latihan 3.10 dalam Python. 216
Kode 4.1 `Q11Fraction.java`: jumlah dan hasil kali dua pecahan hanya dengan bilangan bulat. 224
Kode 4.2 `Q11FractionTest.java`: driver uji sederhana untuk Soal 4.1. 224
Kode 4.3 `q11_fraction.py`: Soal 4.1 dalam Python, beserta uji `unittest`-nya. 225
Kode 4.4 `WrongFractionSum.java`: tiga cara menggoda untuk menjumlahkan 12 dan 13. 227
Kode 4.5 `wrong_fraction_sum.py`: tiga kesalahan yang sama dalam Python. 228
Kode 4.6 `Q12Time.java`: sudut berlawanan arah jarum jam dari jarum jam ke jarum menit. 231
Kode 4.7 `Q12TimeTest.java`: keenam waktu yang diberikan dan tiga kasus tepi. 232
Kode 4.8 `q12_time.py`: Soal 4.2 dalam Python, tempat `%` dan `round` sudah berperilaku seperti yang dituntut Persamaan (4.5). 233
Kode 4.9 `WrongClockAngle.java`: tiga kekeliruan menggoda pada sudut jam pukul 4:41. 235
Kode 4.10 `wrong_clock_angle.py`: di Python, membulatkan sebelum menormalisasi kebetulan memberi jawaban yang benar. 236
Kode 4.11 `Q13ComputerAccount.java`: akun dengan *field* privat dan cara terkendali untuk memeriksa kata sandi. 238
Kode 4.12 `Q13ComputerAccountTest.java`: jalan cerita **Tabel 4.3** sebagai pemeriksaan otomatis. 239
Kode 4.13 `q13_computer_account.py`: Soal 4.3 dalam Python. 241
Kode 4.14 `WrongPasswordCheck.java`: `==` membandingkan identitas objek, `equals` membandingkan karakter. 243
Kode 4.15 `wrong_password_check.py`: di Python, `is` adalah uji identitas dan `==` membandingkan isi. 243
Kode 4.16 `Q14ComplexNumber.java`: bilangan kompleks yang operasi-operasinya mengembalikan objek baru. 245
Kode 4.17 `Q14ComplexNumberTest.java`: keempat contoh tangan dan pemeriksaan bahwa operan-operannya tidak berubah. 246
Kode 4.18 `q14_complex_number.py`: Soal 4.4 dalam Python, tempat `__str__` memainkan peran `toString`. 247
Kode 4.19 `WrongComplex.java`: mengalikan bagian demi bagian, dan mencetak bagian negatif secara naif. 250
Kode 4.20 `wrong_complex.py`: dua kekeliruan yang sama dalam Python. 250
Kode 4.21 `Q15Weight.java`: berat yang disimpan dalam pon, dengan kilogram yang dihitung saat diminta. 251
Kode 4.22 `Q15WeightTest.java`: hasil titik-mengambang dibandingkan dengan toleransi. 252
Kode 4.23 `q15_weight.py`: Soal 4.5 dalam Python, dengan `assertAlmostEqual` untuk pemeriksaan titik-mengambang. 253
Kode 4.24 `WrongWeight.java`: dua *field* yang bisa saling tertinggal. 254
Kode 4.25 `wrong_weight.py`: nilai kedua yang basi, kali ini dalam Python. 255
Kode 4.26 `Q16Fraction.java`: pecahan *immutable* yang operasi-operasinya mengembalikan pecahan baru. 257
Kode 4.27 `Q16FractionTest.java`: ketiga keluaran yang diminta, imutabilitas, dan perhitungan berantai. 258
Kode 4.28 `q16_fraction.py`: Soal 4.6 dalam Python. 259
Kode 4.29 `WrongFraction.java`: `getSum` yang mengubah objek tempat ia dipanggil. 261

Kode 4.30 `wrong_fraction.py`: penjumlahan di tempat yang sama dalam Python. 262
Kode 4.31 `Player.java`: satu pihak dalam pertarungan; `Enemy.java` adalah bayangan cerminnya. 265
Kode 4.32 `EnemyPlayerTest.java`: pertarungan **Tabel 4.4**, dengan perulangan `while` untuk ronde-ronde terakhir. 266
Kode 4.33 `player_enemy.py`: kedua kelas Soal 4.7 dalam satu modul Python. 267
Kode 4.34 `Cheater.java`: kode di luar `Enemy` mencoba menulis *field* privatnya. 271
Kode 4.35 `cheater.py`: Python tidak memaksakan sifat privat, sehingga aturannya bergantung pada disiplin pemrogram. 271
Kode 4.36 `Rupiah.java`: uang yang tepat dalam rupiah utuh, diformat dengan titik dari kanan. 273
Kode 4.37 `RupiahTest.java`: pemformatan di batas-batas kelompok, dan aritmetika yang tepat. 274
Kode 4.38 `rupiah.py`: Soal 4.8 dalam Python. 275
Kode 4.39 `WrongRupiahFormat.java`: mengelompokkan dari kiri, dan uang dalam `double`. 277
Kode 4.40 `wrong_rupiah_format.py`: dua kesalahan yang sama dalam Python. 278
Kode 4.41 `TimeOfDay.java`: waktu dalam sehari yang disimpan sebagai menit sejak tengah malam. 279
Kode 4.42 `TimeOfDayTest.java`: penyimpanan ke jam, putaran balik ke dua arah, pengisian nol, dan perbandingan. 280
Kode 4.43 `time_of_day.py`: Soal 4.9 dalam Python. 281
Kode 4.44 `WrongTime.java`: menambah menit tanpa penyimpanan ke jam, tanpa putaran balik, dan tanpa pengisian nol. 283
Kode 4.45 `wrong_time.py`: tiga kekeliruan yang sama dalam Python. 283
Kode 4.46 `LaundryReceipt.java`: nota yang menetapkan harga layanan dengan `switch` dan menjumlahkannya dalam `Rupiah`. 285
Kode 4.47 `LaundryReceiptTest.java`: nota Sari dan Budi serta penolakan-penolakannya. 287
Kode 4.48 `laundry_receipt.py`: Soal 4.10 dalam Python, yang mengimpor `Rupiah` dari `rupiah.py`. 288
Kode 4.49 `WrongReceipt.java`: memotong atau membulatkan berat, dan memformat dengan *locale* Indonesia. 291
Kode 4.50 `wrong_receipt.py`: pemotongan, pembulatan, dan pembulatan setengah ke genap milik Python. 291
Kode 4.51 `PersonName.java`: awal kata ditemukan karakter demi karakter. 293
Kode 4.52 `PersonNameTest.java`: nama yang berantakan dan tiga kasus tepi. 294
Kode 4.53 `person_name.py`: Soal 4.11 dalam Python. 295
Kode 4.54 `WrongInitials.java`: `split(" ")` menghasilkan kata-kata kosong, dan `charAt(0)` dari kata kosong membuat program macet. 296
Kode 4.55 `wrong_initials.py`: kata-kata kosong yang sama dalam Python, dan sebuah `IndexError`. 297
Kode 4.56 `PasswordChecker.java`: menghitung dalam satu perulangan, memutuskan sesudahnya. 299
Kode 4.57 `PasswordCheckerTest.java`: angka di salah satu ujung, batas panjang, dan kata sandi kosong. 300
Kode 4.58 `password_checker.py`: Soal 4.12 dalam Python. 300
Kode 4.59 `WrongPassword.java`: perulangan yang mengembalikan putusan setelah karakter pertama. 302
Kode 4.60 `wrong_password.py`: pengembalian terlalu dini yang sama dalam Python. 303
Kode 4.61 `LoopTrace.java`: dua perulangan yang mencetak tabel jejaknya sendiri. 305
Kode 4.62 `LoopTraceTest.java`: nol, satu angka, nol di belakang, dan kedua urutan untuk Euclid. 306
Kode 4.63 `loop_trace.py`: Soal 4.13 dalam Python, dengan `//` untuk pembagian bilangan bulat. 306
Kode 4.64 `WrongLoop.java`: kondisi `n > 9` berhenti satu lintasan terlalu awal. 308
Kode 4.65 `wrong_loop.py`: kondisi yang meleset satu, dan `/` padahal yang dimaksud `//`. 309
Kode 4.66 `SimpleDate.java`: tahun kabisat menurut Persamaan (4.9), panjang bulan dengan `switch`, dan hari berikutnya dalam tiga kasus. 310
Kode 4.67 `SimpleDateTest.java`: keempat jenis tahun, akhir bulan, akhir tahun, dan tanggal yang tidak sah. 311
Kode 4.68 `simple_date.py`: Soal 4.14 dalam Python, dengan `match` sebagai ganti `switch`. 312

Kode 4.69 `WrongLeapYear.java`: "habis dibagi 4" saja. 314
Kode 4.70 `wrong_leap_year.py`: aturan Julian yang sama dalam Python. 315
Kode 5.1 `FruitArray.java`: membuat, mengisi, dan menelusuri larik Java, ditambah bentuk inisialisasi singkat. 325
Kode 5.2 `fruit_array.py`: program yang sama dengan list Python yang panjangnya tetap. 327
Kode 5.3 `ArrayBounds.java`: perulangan yang berjalan satu langkah terlalu jauh dan menulis melewati ujung larik berisi lima elemen. 328
Kode 5.4 `array_bounds.py`: kesalahan selisih-satu yang sama di Python. 329
Kode 5.5 `LoadStats.java`: total, maksimum, dan rata-rata sebuah `double[]`, serta jumlah per baris sebuah `int[][]`. 330
Kode 5.6 `load_stats.py`: statistik yang sama dengan list Python dan list berisi list. 331
Kode 5.7 `ArrayCopy.java`: *aliasing*, penyalinan dengan `Arrays.copyOf` dan `System.arraycopy`, serta pembandingan dengan `==` dan `Arrays.equals`. 331
Kode 5.8 `array_copy.py`: *aliasing* dan penyalinan list Python, serta `is` lawan `==`. 333
Kode 5.9 `Hello2.java`: mencetak argumen baris perintah. 333
Kode 5.10 `hello2.py`: mencetak argumen baris perintah dari `sys.argv`. 334
Kode 5.11 `FruitList.java`: operasi `ArrayList` sehari-hari. 335
Kode 5.12 `fruit_list.py`: operasi yang sama pada list Python. 336
Kode 5.13 `BoxingDemo.java`: *boxing*, *unboxing*, dan dua makna `remove` pada list berisi `Integer`. 339
Kode 5.14 `boxing_demo.py`: menghapus menurut posisi dan menurut nilai di Python. 340
Kode 5.15 `SquareInput.java`: membaca satu baris, satu bilangan bulat, satu bilangan desimal, dan satu baris lagi dengan `Scanner`. 341
Kode 5.16 `square_input.py`: dialog yang sama dengan `input()`, `int()`, dan `float()`. 343
Kode 5.17 `FruitCollector.java`: membaca baris-baris ke dalam `ArrayList` sampai bertemu *sentinel* `#stop#`. 343
Kode 5.18 `fruit_collector.py`: perulangan *sentinel* yang sama di Python. 344
Kode 5.19 `InputTest.java`: membaca bilangan bulat sampai bertemu 99, tanpa perlindungan terhadap masukan yang buruk. 345
Kode 5.20 `input_test.py`: perulangan tanpa perlindungan yang sama di Python. 346
Kode 5.21 `PropagationDemo.java`: eksepsi yang dilempar tiga panggilan di bawah dan ditangkap di `main`. 347
Kode 5.22 `propagation_demo.py`: propagasi yang sama di Python. 348
Kode 5.23 `SafeInput.java`: menangkap `InputMismatchException` sehingga masukan yang buruk dilaporkan lalu dilewati. 351
Kode 5.24 `safe_input.py`: menangkap `ValueError` dari `int()` di Python. 353
Kode 5.25 `CatchOrder.java`: tiga klausa `catch`, dari yang paling spesifik sampai yang paling umum. 354
Kode 5.26 `catch_order.py`: tiga klausa `except`, dari yang paling spesifik sampai yang paling umum. 355
Kode 5.27 `FinallyDemo.java`: `finally` dijalankan sesudah `return` yang normal dan ketika sebuah eksepsi lolos. 356
Kode 5.28 `finally_demo.py`: `finally` di Python, dan klausa `else` 357
Kode 5.29 `IllegalArgumentExceptionExample.java`: melempar `IllegalArgumentException` untuk nilai *grayscale* di luar rentang, lalu memulihkan diri dengan memangkasnya menjadi 255. 358
Kode 5.30 `grayscale.py`: melempar `ValueError` untuk nilai *grayscale* di luar rentang. 359
Kode 5.31 `UncheckedDemo.java`: tiga eksepsi tak terperiksa yang disebabkan oleh kesalahan pemrograman. 360
Kode 5.32 `unchecked_demo.py`: tiga jenis kesalahan yang sama di Python. 362
Kode 5.33 `OpenFile.java`: `firstLine` mendeklarasikan `IOException` dengan `throws`; `main` menangkapnya. 363
Kode 5.34 `open_file.py`: `first_line` mendokumentasikan eksepsinya di dalam *docstring*; `main` menangkapnya. 365

Kode 5.35 `WeightCheck.java`: eksepsi terperiksa buatan sendiri yang membawa teks penyebab masalah dan, bila ada, penyebab (*cause*) aslinya. 366
Kode 5.36 `weight_check.py`: kelas eksepsi buatan sendiri di Python, dirantai dengan `raise ... from` 367
Kode 5.37 `ReadUntilStop.java`: membaca baris-baris dari papan ketik dengan `BufferedReader` sampai bertemu `stop` atau akhir masukan. 369
Kode 5.38 `read_until_stop.py`: membaca baris-baris dari masukan standar sampai bertemu `stop` atau akhir masukan. 370
Kode 5.39 `CopyFileClassic.java`: menyalin berkas dan menutup kedua *stream* di dalam `finally`, cara sebelum Java 7. 371
Kode 5.40 `copy_file_classic.py`: penyalinan yang sama dengan panggilan `close()` eksplisit di dalam `finally`. 372
Kode 5.41 `CopyFile.java`: menyalin dengan `try`-with-resources, yang menutup kedua *stream* secara otomatis. 373
Kode 5.42 `copy_file.py`: menyalin dengan pernyataan `with`. 374
Kode 5.43 `ReceiptWriter.java`: menyusun teks dengan `StringBuilder`, menulis berkas, menambahkan isi ke dalamnya, lalu membacanya kembali. 374
Kode 5.44 `receipt_writer.py`: struk yang sama dengan `"".join`, mode `open` `"w"` dan `"a"`, serta perulangan `for` atas berkas. 376
Kode 5.45 `CsvSplit.java`: dua jebakan dalam memecah baris CSV di Java. 377
Kode 5.46 `csv_split.py`: memecah dengan tangan, dan memecah dengan benar memakai modul `csv`. 378
Kode 5.47 `Order.java`: satu pesanan cucian, beserta harganya dan bentuk CSV-nya. 381
Kode 5.48 `order.py`: kelas yang sama di Python, dengan sebuah *dictionary* berisi tarif. 383
Kode 5.49 `OrderFormatException.java`: eksepsi terperiksa untuk satu baris yang tidak sah. 383
Kode 5.50 `order_errors.py`: eksepsi yang sama di Python. 384
Kode 5.51 `OrderCsv.java`: mengurai dan memvalidasi satu baris, memuat seluruh berkas, dan menyimpan. 384
Kode 5.52 `order_csv.py`: pemuatan, validasi, dan penyimpanan yang sama di Python. 387
Kode 5.53 `LaundryApp.java`: memuat, melaporkan, menghitung harga, menyimpan, dan memuat ulang pesanan-pesanan. 389
Kode 5.54 `laundry_app.py`: aplikasi Python-nya. 391
Kode 5.55 `TenthFruit.java`: buah kesepuluh, jika ada. 398
Kode 5.56 `tenth_fruit.py`: fungsi yang sama di Python. 399
Kode 5.57 `GrowArray.java`: tumbuh satu slot lawan penggandaan. 399
Kode 5.58 `grow_array.py`: percobaan yang sama di Python. 400
Kode 5.59 `SomeNumbers.java`: melempar `IllegalArgumentException` dan menangkapnya di pemanggil. 402
Kode 5.60 `some_numbers.py`: hal yang sama dengan `ValueError`. 402
Kode 5.61 `AverageInput.java`: perata-rataan bilangan bulat yang diketik secara tangguh. 403
Kode 5.62 `average_input.py`: hal yang sama di Python. 403
Kode 5.63 `PrepaidCard.java`: eksepsi terperiksa buatan sendiri untuk saldo yang tidak mencukupi. 404
Kode 5.64 `prepaid_card.py`: kartu yang sama di Python. 405
Kode 5.65 `ServiceSummary.java`: cacah pesanan, kilogram, dan pendapatan per layanan. 406
Kode 5.66 `service_summary.py`: ringkasan yang sama di Python. 407
Kode 5.67 `WordCount.java`: menghitung baris, kata, dan karakter sebuah berkas yang namanya diberikan di baris perintah. 409
Kode 5.68 `word_count.py`: hitungan yang sama di Python. 409
Kode 6.1 `BankAccount.java`: kelas dengan dua konstruktor dan komentar dokumentasi pada setiap anggotanya. 413
Kode 6.2 `bank_account.py`: satu `__init__` dengan argumen bawaan mengerjakan tugas dua konstruktor Java. 416

Kode 6.3 `RandomDemo.java`: metode-metode utama `java.util.Random`, dengan benih tetap. 423
Kode 6.4 `random_demo.py`: gagasan yang sama dengan modul `random` Python. 424
Kode 6.5 `Dog.java`: kelas yang akan kita uji. Kelas ini tidak memiliki `main`; program-program uji di bawah yang memakainya. 425
Kode 6.6 `dog.py`: kelas yang sama dalam Python, dengan *property* yang hanya dapat dibaca. 426
Kode 6.7 `DogCheck.java`: menguji `Dog` dengan metode `check` buatan sendiri, tanpa pustaka uji. 427
Kode 6.8 `dog_check.py`: program pemeriksa-diri yang sama dalam Python. 429
Kode 6.9 `DoubleEquality.java`: mengapa `==` adalah uji yang keliru untuk dua `double`. 431
Kode 6.10 `double_equality.py`: percobaan yang sama, ditambah `math.isclose`. 431
Kode 6.11 `DogTest.java`: uji-uji `Dog` yang ditulis untuk JUnit 6. 433
Kode 6.12 `test_dog.py`: uji-uji `Dog` yang ditulis untuk `unittest`. 435
Kode 6.13 `IdentityDemo.java`: kelas tanpa `equals`, dan kelas dengan `equals` tetapi tanpa `hashCode`. 439
Kode 6.14 `identity_demo.py`: dua kesalahan yang sama dalam Python; kesalahan kedua tertangkap. 442
Kode 6.15 `Customer.java`: kesamaan berdasarkan nomor telepon, `hashCode` yang serasi, dan `toString` yang mudah dibaca. 444
Kode 6.16 `customer.py`: `__eq__`, `__hash__` yang serasi, serta `__repr__` dan `__str__`. 447
Kode 6.17 `AverageTwice.java`: dua metode dengan nama yang sama dan tubuh yang nyaris sama. 450
Kode 6.18 `OverloadDemo.java`: versi mana dari metode yang di-*overload* yang dipilih setiap pemanggilan? 453
Kode 6.19 `average_twice.py`: di Python, `def average` kedua menggantikan yang pertama. 455
Kode 6.20 `overload_demo.py`: argumen bawaan, `*args`, dan `functools.singledispatch`. 456
Kode 6.21 `Measurable.java`: antarmuka dengan satu metode dan tanpa kode. 457
Kode 6.22 `MeasurableDemo.java`: dua kelas mengimplementasikan `Measurable`, dan satu metode `average` melayani keduanya. 458
Kode 6.23 `measurable_demo.py`: *protocol* `Measurable`; perhatikan bahwa `Account` dan `Country` tidak pernah menyebutnya. 461
Kode 6.24 `dbrep/DBClient.java`: klien yang menyimpan satu variabel per representasi dan menguji sebuah penanda (*flag*) di setiap operasi. 463
Kode 6.25 `db_client_v1.py`: klien penguji penanda yang sama dalam Python. 467
Kode 6.26 `dbiface/DBClient.java`: satu variabel bertipe antarmuka, tanpa penanda, dan tanpa pengujian penanda. 469
Kode 6.27 `db_client_v2.py`: antarmuka sebagai kelas dasar abstrak, dengan dua metode bersama. 473
Kode 6.28 `NameSortDemo.java`: `compareTo` yang benar, dan satu yang percaya bahwa `String.compareTo` hanya mengembalikan −1, 0, atau 1. 476
Kode 6.29 `name_sort.py`: pengurutan dengan kunci *tuple*, *bug* `switch` yang sama, dan pengurutan dengan fungsi `key`. 480
Kode 6.30 `StopButtonDemo.java`: tombol yang memberi tahu setiap pendengar terdaftar melalui sebuah antarmuka. 483
Kode 6.31 `stop_button_demo.py`: pola pendengar yang sama dengan sebuah *protocol* Python. 485
Kode 6.32 `Order.java`: pesanan laundry, divalidasi ketika dibangun dan dikenali dari nomor tiketnya. 488
Kode 6.33 `order.py`: pesanan yang sama dalam Python, dengan *property* yang hanya dapat dibaca. 490
Kode 6.34 `PriceRule.java`: antarmuka yang diimplementasikan setiap aturan harga, dengan metode pembantu statis untuk memformat rupiah. 492
Kode 6.35 `price_rule.py`: `PriceRule` sebagai kelas dasar abstrak, dan pembantu `rupiah`. 493
Kode 6.36 `PerKiloRule.java`: harga per kilogram yang sudah dimulai, dengan minimum. 494
Kode 6.37 `PerItemRule.java`: harga per potong. 495
Kode 6.38 `FlatRule.java`: satu harga tetap. 496
Kode 6.39 `ExpressRule.java`: dekorator yang menambahkan 50%, dibulatkan ke bawah. 497
Kode 6.40 `MemberDiscountRule.java`: dekorator yang memotong sekian persen untuk anggota, dan menolak persentase yang mustahil. 498

Kode 6.41 `rules.py`: kelima aturan harga dalam Python. 500
Kode 6.42 `LaundryDemo.java`: memilih aturan sekali, lalu memakainya hanya melalui antarmuka. 502
Kode 6.43 `laundry_demo.py`: pagi yang sama dalam Python. 504
Kode 6.44 `PriceRuleTest.java`: uji JUnit 6 untuk setiap aturan harga. 505
Kode 6.45 `PriceRuleCheck.java`: uji-uji aturan harga sebagai program biasa. 508
Kode 6.46 `test_price_rules.py`: uji pytest untuk setiap aturan harga, dengan batas-batas berparameter. 509
Kode 6.47 `price_rule_check.py`: uji-uji aturan harga sebagai program Python biasa. 512
Kode 6.48 `KilosCharged.java`: pembantu yang terdokumentasi penuh, dengan contoh-contoh terdokumentasinya di `main`. 518
Kode 6.49 `kilos_charged.py`: *docstring* gaya Google yang contoh-contohnya berupa *doctest*, salah satunya eksepsi yang diharapkan. 519
Kode 6.50 `ChainedAccount.java`: dua konstruktor publik yang mendelegasikan kepada satu konstruktor privat. 520
Kode 6.51 `chained_account.py`: konstruktor alternatif yang ditulis sebagai metode kelas. 521
Kode 6.52 `BankAccountTest.java`: uji JUnit 6 untuk `interestOn`, membandingkan `double` dengan sebuah delta. 522
Kode 6.53 `test_bank_account.py`: uji `unittest` untuk `interest_on` dengan `assertAlmostEqual`. 523
Kode 6.54 `MemberCardCheck.java`: uji-uji kartu loyalitas, ditulis sebelum kode yang diujinya. 524
Kode 6.55 `MemberCard.java`: aturan kartu loyalitas, ditulis untuk meluluskan uji-ujinya. 525
Kode 6.56 `test_member_card.py`: versi pytest dari uji-uji kartu loyalitas. 526
Kode 6.57 `member_card.py`: aturannya, dengan pembagian yang membulatkan ke bawah. 526
Kode 6.58 `Machine.java`: kesamaan dan kode *hash* yang dibangun dari dua *field*. 527
Kode 6.59 `machine.py`: `__eq__` dan `__hash__` yang dibangun dari *tuple* dua *field*. 527
Kode 6.60 `CustomerRecordDemo.java`: sebuah *record* membandingkan semua komponennya. 528
Kode 6.61 `customer_record.py`: *dataclass* beku membandingkan semua *field*-nya. 529
Kode 6.62 `OverloadPuzzle.java`: pemilihan *overload* dengan pelebaran dan *boxing*. 529
Kode 6.63 `overload_puzzle.py`: `singledispatch` dengan `bool`, `Fraction`, dan bilangan bulat raksasa. 530
Kode 6.64 `AverageLoad.java`: sebuah adapter membuat `Order` dapat ikut serta dalam `average`. 530
Kode 6.65 `average_load.py`: adapter yang sama dalam Python. 531
Kode 6.66 `BulkDiscountRule.java`: dekorator yang memberi potongan untuk pesanan berat. 532
Kode 6.67 `BulkDiscountCheck.java`: uji-uji batas untuk potongan borongan. 532
Kode 6.68 `bulk_discount.py`: potongan borongan dalam Python. 533
Kode 6.69 `test_bulk_discount.py`: uji-uji batas pytest untuk potongan borongan. 534
Kode 6.70 `DecoratorOrder.java`: menumpuk dua dekorator yang sama dalam kedua urutan. 534
Kode 6.71 `decorator_order.py`: percobaan yang sama dalam Python. 535
Kode 6.72 `OrderSortDemo.java`: `Comparator` yang mengurutkan dari yang terberat, lalu menurut nomor tiket. 536
Kode 6.73 `order_sort_demo.py`: pengurutan dengan kunci *tuple*. 536
Kode 7.1 `db/DBTest.java`: proyek basis data dengan superkelas abstrak `DB` dan dua subkelas, `LinearDB` dan `TreeDB`, yang metode-metodenya hanya melaporkan bahwa mereka dipanggil. 541
Kode 7.2 `db_test.py`: proyek basis data dalam Python. `DB` mewarisi `ABC` dan menandai `add_key` dan `search` sebagai abstrak. 544
Kode 7.3 `IntDBDemo.java`: basis data kunci yang berfungsi. `LinearIntDB` meng-*override* `addSeveral` warisan dengan penyalinan borongan; `TreeIntDB` mewarisinya tanpa perubahan. 546
Kode 7.4 `int_db_demo.py`: basis data yang sama dalam Python; `LinearIntDB` meng-*override* `add_several` dengan satu panggilan `list.extend`. 551
Kode 7.5 `ConstructorChain.java`: rantai tiga tingkat `LaundryItem`, `Garment`, `Shirt`; setiap konstruktor mengumumkan dirinya, dan setiap `describe` memperpanjang teks induknya dengan `super.describe()`. 556

Kode 7.6 `constructor_chain.py`: rantai yang sama dalam Python, ditambah sebuah kelas yang lupa memanggil `super().__init__()`. 559
Kode 7.7 `animal/AnimalTest.java`: `Animal` dengan konstruktor dan metode `getPrice` yang di-*overload*, tiga subkelas yang meng-*override* `sound()`, dan program uji yang menjalankan *overloading* maupun *overriding*. 562
Kode 7.8 `animal_test.py`: program Animal dalam Python; satu `__init__` dan satu `get_price` dengan parameter kata kunci opsional masing-masing menggantikan empat *overload* Java. 570
Kode 7.9 `DispatchDemo.java`: *dynamic dispatch* pada `start()`, tidak adanya *dispatch* untuk field yang tersembunyi dan metode statis yang tersembunyi, *downcast* yang aman dengan `instanceof`, dan *downcast* yang tidak aman. 575
Kode 7.10 `dispatch_demo.py`: pencarian dinamis di Python, dengan fungsi pembantu yang menelusuri urutan resolusi metode persis seperti yang dilakukan Python. 580
Kode 7.11 `ObjectRoot.java`: kelas yang mewarisi `toString`, `equals`, dan `hashCode` dari `Object`, dan kelas yang meng-*override* ketiganya. 582
Kode 7.12 `object_root.py`: ketiga metode yang sama dalam Python, dan sebuah kelas yang mendefinisikan `__eq__` tanpa `__hash__`. 584
Kode 7.13 `FigureDemo.java`: `Figure` abstrak dengan subkelas abstrak `OpenFigure` dan `ClosedFigure` serta empat bangun konkret. 587
Kode 7.14 `figure_demo.py`: hierarki bangun dalam Python; `OpenFigure` dan `ClosedFigure` mencantumkan `ABC` sekali lagi untuk menegaskan bahwa keduanya pun abstrak. 591
Kode 7.15 `dbconcrete/DBTest.java`: `TreeDB` mewarisi kelas konkret `LinearDB`. 595
Kode 7.16 `db_concrete.py`: versi superkelas konkret dalam Python. 598
Kode 7.17 `FragileBase.java`: himpunan tag yang menghitung, dibangun dengan pewarisan, yang menghitung setiap tag dua kali, dan yang dibangun dengan komposisi, yang menghitung dengan benar. 600
Kode 7.18 `fragile_base.py`: kelas dasar yang rapuh dalam Python; `UserDict.update` memanggil `__setitem__` untuk setiap pasangan. 602
Kode 7.19 `DiamondDemo.java`: dua antarmuka dengan metode default yang bertentangan; kelas pengimplementasinya wajib menyelesaikan pertentangan itu, dan metode yang diwarisi dari kelas mengalahkan default mana pun. 605
Kode 7.20 `diamond_mro.py`: hierarki berlian dengan panggilan `super()` kooperatif, MRO yang dihasilkannya, pertentangan yang diselesaikan oleh urutan, dan hierarki yang tidak memiliki urutan yang konsisten. 607
Kode 7.21 `one/Alpha.java`: satu field untuk setiap tingkat akses. 611
Kode 7.22 `one/Beta.java`: sepaket dengan `Alpha`, bukan subkelas. 611
Kode 7.23 `two/AlphaSub.java`: paket lain, tetapi subkelas `Alpha`. 612
Kode 7.24 `two/Gamma.java`: paket lain, tidak berhubungan; memuat `main`. 612
Kode 7.25 `access_demo.py`: tiga konvensi penamaan Python, dan apa yang sebenarnya dilakukan *name mangling*. 614
Kode 7.26 `EncapsulationDemo.java`: `WashingMachine` yang field-field private-nya hanya dapat berubah melalui metode yang menjaga invarian $0 \le \ell \le c$. 615
Kode 7.27 `encapsulation_demo.py`: mesin yang sama dalam Python; `capacity_kg` adalah property hanya-baca dan `load_kg` property yang setter-nya menegakkan invarian. 618
Kode 7.28 `LaundryPricing.java`: hierarki layanan Profitina Laundry, *decorator* kilat, dan pesanan yang menghitung semua harga secara polimorfis. 621
Kode 7.29 `laundry_pricing.py`: mesin harga yang sama dalam Python. 627
Kode 7.30 `StudentFees.java`: hierarki mahasiswa pada **Gambar 7.5** dengan biaya polimorfis. 637
Kode 7.31 `student_fees.py`: hierarki mahasiswa dalam Python. 639
Kode 7.32 `ShoePromoDemo.java`: cuci sepatu dan promosi ditambahkan ke mesin harga tanpa mengubahnya. 641
Kode 7.33 `shoe_promo_demo.py`: perluasan yang sama dalam Python. 643
Kode 7.34 `ConstructorTrap.java`: konstruktor superkelas yang memanggil metode yang dapat di-override. 646
Kode 7.35 `constructor_trap.py`: jebakan yang sama dalam Python. 646

Kode 8.1 `TreeStart.java`: kelas `Tree` awal dengan dua konstruktor, `mirror()` melalui delegasi, `toString()` yang ringkas, dan gambar menyamping. 653
Kode 8.2 `tree_start.py`: kelas `Tree` yang sama dalam Python. Garis bawah di depan menandai sebuah atribut sebagai *private* menurut konvensi. 656
Kode 8.3 `List.java`: *list* bilangan bulat yang imutabel. Konstruktornya *private*, sehingga `nil()`, `cons()`, dan `of()` adalah satu-satunya cara membangun sebuah *list*. 658
Kode 8.4 `ListException.java`: eksepsi *unchecked* untuk operasi yang memerlukan *list* tak kosong. 660
Kode 8.5 `ListOps.java`: operasi bantu yang ditulis secara rekursif. Konstruktor *private* mencegah siapa pun membuat objek `ListOps`. 660
Kode 8.6 `ListStart.java`: perkenalan pertama dengan `List`, `ListOps`, dan `ListException`. 662
Kode 8.7 `cons_list.py`: kelas *list*, eksepsinya, dan operasi-operasi bantunya dalam satu modul Python. 662
Kode 8.8 `list_start.py`: perkenalan pertama dengan modul *list* Python. 664
Kode 8.9 `BuildTree.java`: kesembilan pernyataan `new Tree`, `size()` yang rekursif, dan pemeriksaan diri. 666
Kode 8.10 `build_tree.py`: pohon yang sama dalam Python, diuji dengan `unittest`. 668
Kode 8.11 `TripleTree.java`: melipattigakan setiap simpul melalui delegasi, dengan varian keliru yang berakhir *crash*. 671
Kode 8.12 `triple_tree.py`: melipattigakan setiap simpul dalam Python. 673
Kode 8.13 `DepthFirstPrint.java`: pencetakan *pre-order*, dengan metode bantu yang menangkap keluaran konsol untuk pengujian. 676
Kode 8.14 `depth_first_print.py`: pencetakan *pre-order* dalam Python, diuji dengan menangkap keluaran standar. 678
Kode 8.15 `FreshCopy.java`: salinan dalam, salinan dangkal yang keliru, dan uji untuk objek simpul yang dipakai bersama. 681
Kode 8.16 `fresh_copy.py`: salinan dalam di Python, dengan uji berbagi simpul yang sama. 684
Kode 8.17 `SaveDepthFirst.java`: menulis penelusuran *pre-order* ke dalam larik dan mengembalikan posisi terakhir yang dipakai. 688
Kode 8.18 `save_depth_first.py`: penelusuran yang sama disimpan ke dalam *list* Python. 690
Kode 8.19 `BstFind.java`: mencari di BST melalui satu lintasan, dengan penghitung kunjungan dan varian keliru yang berakhir *crash*. 693
Kode 8.20 `bst_find.py`: pencarian yang sama dalam Python. 695
Kode 8.21 `BstInsert.java`: menyisipkan ke BST dengan memasang daun baru, disertai pemeriksaan *in-order*. 698
Kode 8.22 `bst_insert.py`: penyisipan BST dalam Python. 701
Kode 8.23 `NegateAll.java`: menegasikan setiap elemen secara rekursif. 705
Kode 8.24 `negate_all.py`: fungsi yang sama dalam Python. 706
Kode 8.25 `ListFind.java`: posisi kemunculan pertama, atau `-1`. 708
Kode 8.26 `list_find.py`: pencarian yang sama dalam Python. 709
Kode 8.27 `AllPositive.java`: memeriksa bahwa tidak ada elemen yang di bawah nol. 711
Kode 8.28 `all_positive.py`: pemeriksaan yang sama dalam Python. 712
Kode 8.29 `Positives.java`: menyaring elemen yang paling sedikit nol. 714
Kode 8.30 `positives.py`: *filter* yang sama dalam Python. 715
Kode 8.31 `SortedCheck.java`: memeriksa urutan tidak turun pasangan demi pasangan. 717
Kode 8.32 `sorted_check.py`: pemeriksaan yang sama dalam Python, dinamai `is_sorted` agar tidak menutupi fungsi bawaan `sorted`. 718
Kode 8.33 `MergeLists.java`: menggabungkan dua *list* terurut dengan kedua kasus dasarnya. 720
Kode 8.34 `merge_lists.py`: penggabungan yang sama dalam Python. 722
Kode 8.35 `SkipElement.java`: membuang setiap salinan satu nilai. 724
Kode 8.36 `skip_element.py`: *filter* yang sama dalam Python. 724
Kode 8.37 `RemoveDuplicates.java`: menghapus duplikat dengan bantuan `skip`. 726
Kode 8.38 `remove_duplicates.py`: metode yang sama dalam Python, dengan mengimpor `skip` dari Soal 8.14. 727

Kode 8.39 `LaundryOrderParser.java`: pengurai baris dengan eksepsi *checked* buatan sendiri, pembaca berkas dengan *try-with-resources*, dan pembaca keliru sebagai pembanding. 730
Kode 8.40 `laundry_order_parser.py`: pengurai yang sama dalam Python, dengan uji `unittest`. 734
Kode 8.41 `GrowableIntArray.java`: larik yang menggandakan diri, dengan Javadoc lengkap dan pemeriksaan batas. 738
Kode 8.42 `growable_int_array.py`: kelas yang sama dalam Python, dengan *docstring* dan metode `setUp` yang dipakai bersama oleh uji-ujinya. 742
Kode 8.43 `PriceRules.java`: antarmuka dengan *default method*, dua implementasi, dan uji yang memperlakukan keduanya melalui antarmuka. 745
Kode 8.44 `price_rules.py`: rancangan yang sama dengan *abstract base class*, dan uji kontrak yang dapat dipakai ulang dan dijalankan terhadap kedua implementasi. 748
Kode 8.45 `ServiceDispatch.java`: kelas abstrak, tiga subkelas, *overriding* dengan `super`, dan dua *overload* dari `label`. 753
Kode 8.46 `service_dispatch.py`: hierarki yang sama dalam Python, dengan `singledispatch` yang memilih berdasarkan kelas saat berjalan. 757
Kode 8.47 `FighterArena.java`: hierarki hasil *refactoring* dengan *template method* yang *final*, *hook* yang *protected*, dan tanpa *setter* publik. 761
Kode 8.48 `fighter_arena.py`: hierarki yang sama dalam Python, dengan *property* `health` yang hanya-baca. 765
Kode 8.49 `OrderBugHunt.java`: kelas dengan empat *bug* enkapsulasi yang diperagakan, dan kelas perbaikannya yang diuji. 769
Kode 8.50 `order_bug_hunt.py`: perburuan yang sama dalam Python, diperbaiki dengan *property*, *setter* yang memvalidasi, dan *tuple*. 771
Kode 8.51 `TraceFinally.java`: `try` dengan `catch` dan `finally`, serta sebuah eksepsi yang tidak ditangani oleh `catch`. 778
Kode 8.52 `trace_finally.py`: alur yang sama dalam Python, dengan kedua galat berupa `ValueError`. 779
Kode 8.53 `ExerciseSolutions.java`: tinggi, jumlah, daun, minimum BST, jumlah *list*, *take*, dan penyisipan persisten, beserta pemeriksaannya. Kompilasikan bersama `List.java` dan `ListException.java`. 780
Kode 8.54 `exercise_solutions.py`: penyelesaian yang sama dalam Python. 782
Kode 9.1 `InterfaceConstants.java`: konstanta di dalam antarmuka `Measurable` dan `Units`, dipakai dengan dan tanpa awalan, serta sebuah `enum` sebagai alternatif modern. 789
Kode 9.2 `interface_constants.py`: atribut kelas yang ditandai `Final`, konstanta modul, dan sebuah `Enum`, yang anggotanya tidak dapat ditugasi ulang. 792
Kode 9.3 `Fibonacci.java`: hitung mundur, Fibonacci standar dengan dua panggilan dan penghitung panggilan, versi rekursif ekor di balik metode pembantu, dan versi linear yang mengembalikan dua nilai sekaligus. 796
Kode 9.4 `fibonacci.py`: keempat fungsi yang sama dalam Python, ditambah versi standar yang dibuat cepat oleh `functools.lru_cache`. 797
Kode 9.5 `ListDemo.java`: menambah, menguji, menghitung, operasi borongan, posisi, dan satu metode yang melayani dua kelas list. 803
Kode 9.6 `list_demo.py`: operasi yang sama dengan `list` Python, dan satu fungsi yang melayani `list` maupun `deque`. 805
Kode 9.7 `IterationDemo.java`: tiga cara mengunjungi list; menghapus di dalam perulangan for-each gagal, sedangkan `Iterator.remove` dan `removeIf` berhasil. 808
Kode 9.8 `iteration_demo.py`: penelusuran yang sama dalam Python. Menghapus dari list di dalam perulangan `for` diam-diam melewatkan elemen; mengubah ukuran kamus di dalam perulangan memunculkan `RuntimeError`. 809
Kode 9.9 `SetDemo.java`: `TreeSet` bilangan dengan gabungan, selisih, dan irisan; nama-nama yang sama dalam `HashSet`, `LinkedHashSet`, dan `TreeSet`; kueri elemen terdekat; dan himpunan huruf. 811
Kode 9.10 `set_demo.py`: `set` Python dengan operator-operatornya, urutan-urutan yang diberikan tiga himpunan Java, pencarian biner pada list terurut, dan sebuah `frozenset`. 813

Kode 9.11 `MapDemo.java`: menelusuri entri sebuah map, mula-mula dengan tipe mentah dan cast, lalu dengan generik; metode-metode utama `Map` pada daftar harga; menghitung dengan `merge`; dan tiga jenis map. 818
Kode 9.12 `map_demo.py`: `dict` Python, `get`-nya dengan nilai bawaan, `KeyError`, dan tiga cara menghitung: `get`, `Counter`, dan `defaultdict`. 820
Kode 9.13 `WordFile.java`: sebuah berkas dibaca ke dalam `List` kata dan `List` baris, kata-katanya dihitung dalam `TreeMap`, dan yang tersering ditemukan dengan mengurutkan entri-entrinya. 822
Kode 9.14 `word_file.py`: laporan yang sama dalam Python, dengan kunci pengurutan yang mengurutkan menurut hitungan menurun dan kata menaik, serta `Counter.most_common` 824
Kode 9.15 `QueueDemo.java`: `ArrayDeque` sebagai antrean FIFO, sebagai tumpukan LIFO, dan sebagai deque; `PriorityQueue` dengan urutan alami dan dengan `Comparator.reverseOrder()`. 826
Kode 9.16 `queue_demo.py`: `deque` sebagai antrean dan deque, `list` sebagai tumpukan, dan `heapq` sebagai antrean prioritas. 828
Kode 9.17 `RawTypes.java`: `List` mentah menerima sebuah `Integer` di antara nama-nama dan baru gagal ketika di-cast; `List<String>` generik tidak memerlukan cast dan akan menolak `Integer` itu saat kompilasi. 830
Kode 9.18 `raw_types.py`: list Python menerima apa saja; petunjuk `list[str]` memungkinkan mypy menolak `append(42)` sebelum program berjalan. 832
Kode 9.19 `GListDemo.java`: `GList<T>`, list imutabel generik dengan pembantu rekursif untuk `toString`, dipakai dengan `Integer` dan dengan `String`. 833
Kode 9.20 `glist_demo.py`: list generik dengan `TypeVar` dan `Generic[T]`, diperiksa oleh mypy. 835
Kode 9.21 `GenericMethods.java`: kelas generik dengan dua parameter tipe, `Pair<A, B>`, dan empat metode generik, di antaranya `removeDuplicatesFromSortedList` 837
Kode 9.22 `generic_methods.py`: `Pair(Generic[A, B])` dan fungsi-fungsi generik yang bertipe `TypeVar`. 840
Kode 9.23 `GList2Demo.java`: list yang tipe elemennya dibatasi oleh `Comparable`, dengan metode rekursif `sorted`, dan metode generik terbatas `max`. 843
Kode 9.24 `glist2_demo.py`: `TypeVar` yang dibatasi protokol `SupportsLessThan`; mypy menolak list berisi `object` biasa, dan Python gagal ketika mencoba membandingkannya. 846
Kode 9.25 `WildcardDemo.java`: `Animal`, `Cat`, dan `Dog`; metode pembaca dengan `? extends Animal`; metode `addAll(src, dest)` dengan `? super T`; dan larik kovarian yang gagal saat eksekusi. 851
Kode 9.26 `wildcard_demo.py`: `Sequence[Animal]` menerima list kucing (baca-saja, kovarian); `list[Animal]` tidak (invarian), dan bila aturan itu diabaikan, seekor anjing berakhir di antara kucing-kucing. 854
Kode 9.27 `ErasureDemo.java`: saat eksekusi `List<String>` dan `List<Integer>` adalah kelas yang sama; referensi mentah dapat menyelundupkan `Integer` ke dalam `List<String>`, dan `Collections.checkedList` menghentikannya di pintu. 857
Kode 9.28 `erasure_demo.py`: `list[str]` hanyalah alias bagi `list`; tidak ada yang memeriksa argumen tipe saat eksekusi, tetapi petunjuk tipe tetap tersedia bagi perkakas. 858
Kode 9.29 `MyCollection.java`: sebuah iterator, fakta bahwa iterator hanya membaca, menghapus duplikat dari list terurut, sebuah `Iterable` dalam perulangan for-each, dan mengurutkan mahasiswa dengan `AgeComparator`. 860
Kode 9.30 `my_collection.py`: kelima bagian dalam Python; fungsi pembanding bergaya Java yang diadaptasi dengan `functools.cmp_to_key`, dan fungsi `key=` yang Pythonic. 865
Kode 9.31 `SortingDemo.java`: urutan alami, `comparingInt(...).reversed()`, `comparing(...).thenComparing(...)`, comparator lambda, `TreeSet` dengan comparator, dan limpahan pada comparator pengurangan. 867
Kode 9.32 `sorting_demo.py`: urutan alami dengan `__lt__` dan `total_ordering`, fungsi kunci, dua pengurutan stabil untuk arah campuran, dan bilangan bulat Python yang tak terbatas. 870
Kode 9.33 `LaundryOrderBook.java`: buku pesanan Profitina Laundry dengan `TreeMap`, `MultiIndex` generik, `EnumMap` berisi `LinkedHashSet`, dan `PriorityQueue`, ditambah laporan pendapatan yang diurutkan dengan comparator. 875

Kode 9.34 `laundry_order_book.py`: buku pesanan yang sama dalam Python, dengan `MultiIndex` generik, kamus yang dipakai sebagai himpunan berurutan, dan `heapq` untuk antrean cucian. 883
Kode 9.35 `Anagrams.java`: kelompok-kelompok anagram dalam `TreeMap` yang berkunci huruf-huruf terurut setiap kata. 895
Kode 9.36 `anagrams.py`: pengelompokan yang sama dengan `defaultdict`. 896
Kode 9.37 `StackDemo.java`: `ArrayStack<T>` generik yang puncaknya adalah ujung sebuah `ArrayList` privat, dan pemeriksa kurung yang dibangun di atasnya. 896
Kode 9.38 `stack_demo.py`: tumpukan generik dalam Python, dan pemeriksa kurung yang sama dengan tabel pasangan. 898
Kode 9.39 `LruCache.java`: cache LRU dalam selusin baris, dengan memperluas `LinkedHashMap` dalam urutan akses. 901
Kode 9.40 `lru_cache.py`: cache yang sama dengan `OrderedDict` dan `move_to_end`. 902
Kode 10.1 `FirstWindow.java`: frame dengan satu label, dibangun dalam enam langkah, dengan mode `--selftest` yang berjalan tanpa layar. 911
Kode 10.2 `first_window.py`: jendela yang sama dalam Tkinter, dengan mode `--selftest` yang sama. 914
Kode 10.3 `ToyEventLoop.java`: gelung peristiwa tulisan tangan dengan komponen tiruan, listener, dan sebuah antrean. 918
Kode 10.4 `toy_event_loop.py`: gelung peristiwa yang sama dalam Python, dengan callback berupa fungsi biasa. 920
Kode 10.5 `EchoPanel.java`: panel yang mendengarkan Enter di field teksnya dan menyalin teksnya ke sebuah label. 924
Kode 10.6 `echo_panel.py`: panel gema yang sama dalam Tkinter, memakai `bind` untuk memasang callback pada tombol Enter. 926
Kode 10.7 `FruitChoice.java`: dua kotak centang dan satu listener item yang melaporkan kotak mana yang berubah dan bagaimana. 927
Kode 10.8 `fruit_choice.py`: pilihan yang sama dalam Tkinter, dengan satu `BooleanVar` per tombol centang dan sebuah callback `command`. 929
Kode 10.9 `LayoutTour.java`: empat layout manager utama; uji mandirinya mencetak batas-batas yang ditetapkan setiap pengelola. 935
Kode 10.10 `layout_tour.py`: `pack` dan `grid` sedang bekerja; uji mandirinya mencetak geometri yang ditetapkan Tk. 938
Kode 10.11 `ShadeBox.java`: pelukisan di `paintComponent`, dua tombol yang dibedakan dengan `getSource()`, dan klik tetikus pada kotak yang dilukis. 944
Kode 10.12 `shade_box.py`: kotak penggelap di atas `Canvas` Tkinter, tempat kotak itu adalah item yang diingat dan digambar ulang oleh Tk. 949
Kode 10.13 `ListenerStyles.java`: lima cara menulis action listener di Java, dan lambda yang diciptakan di dalam perulangan. 953
Kode 10.14 `listener_styles.py`: lima jenis callback Tkinter, jebakan *late binding* dalam perulangan, dan obatnya. 958
Kode 10.15 `Doodle.java`: menggambar dengan tetikus, dengan `MouseAdapter` anonim dan metode lukis yang menggambar ulang setiap goresan yang tersimpan. 961
Kode 10.16 `doodle.py`: menggambar dengan tetikus di kanvas Tkinter; setiap ruas adalah item garis yang disimpan Tk. 964
Kode 10.17 `SlowWork.java`: pekerjaan lambat di EDT membekukan jendela; sebuah `SwingWorker` menjalankannya di thread pekerja dan menampilkan hasilnya di EDT. 967
Kode 10.18 `slow_work.py`: thread pekerja menghitung, sebuah `queue.Queue` membawa hasilnya, dan `after` memeriksa antrean di thread GUI. 971
Kode 10.19 `LaundryOrderWindow.java`: jendela pencatatan pesanan Profitina Laundry dalam Swing, dengan kelas-kelas modelnya dan mode `--selftest`. 978
Kode 10.20 `laundry_order_window.py`: jendela pencatatan pesanan yang sama dalam Tkinter, dengan model yang sama dan uji mandiri yang sama. 990
Kode 10.21 `MachineBoard.java`: empat mesin dilukis sebagai lingkaran; klik di dalam lingkaran, diuji dengan Persamaan (10.8), membalik keadaan mesin itu. 1007

Kode 10.22 `machine_board.py`: papan yang sama di kanvas Tkinter, tempat `tag_bind` pada setiap oval melakukan uji kenanya. 1009
Kode 11.1 `StudentMvcDemo.java`: program MVC terkecil. `Student` adalah model, `StudentView` view, dan `StudentController` controller. 1022
Kode 11.2 `student_mvc_demo.py`: data mahasiswa dalam Python, dengan *field* model disajikan sebagai *property*. 1025
Kode 11.3 `ObserverDemo.java`: sebuah subjek, `WashCounter`, yang mengumumkan setiap perubahan properti `loads`-nya melalui `PropertyChangeSupport`, dan dua pengamat yang tidak saling mengenal. 1029
Kode 11.4 `observer_demo.py`: pola Observer yang ditulis sendiri. Subjek menyimpan list berisi *callable* dan memanggil masing-masing dengan nama properti serta nilai lama dan barunya. 1031
Kode 11.5 `CounterMvc.java`: pencacah Up/Down sebagai model, *interface* view dengan implementasi Swing dan implementasi teks, serta controller. Dengan `--selftest` ia berjalan tanpa jendela. 1036
Kode 11.6 `counter_mvc.py`: pencacah dalam Python. `TkCounterView` mengimpor Tkinter di dalam konstruktornya, sehingga uji mandiri tetap berjalan bahkan di tempat Tkinter tidak terpasang. 1042
Kode 11.7 `PredictiveKeypad.java`: papan tombol ponsel dengan teks prediktif. `T9Dictionary` dan `MessageModel` membentuk model, `KeypadController` controller, `SwingMessageView` dan `SwingMatchView` dua view dari model yang sama, dan `KeypadPanel` kedua belas tombol; `TextMessageView` menggantikan jendela dalam uji mandiri. 1048
Kode 11.8 `predictive_keypad.py`: papan tombol prediktif dalam Python. Model dan controller-nya sepadan dengan kelas-kelas Java; `TkKeypadWindow` membangun kotak pesan, baris kecocokan, dan kedua belas tombol dengan Tkinter. 1057
Kode 11.9 `MessageModelCheck.java`: sepuluh pemeriksaan terhadap model dan controller papan tombol, dijalankan tanpa jendela apa pun. 1064
Kode 11.10 `test_keypad_model.py`: *unit test* model dan controller papan tombol dengan `unittest`. 1067
Kode 11.11 `OrderBook.java`: model meja depan, dengan buku pesanan, satu pesanan, dua enumerasi, dan *interface* pendengar. Ia tidak mengimpor apa pun dari Swing. 1071
Kode 11.12 `order_book.py`: model meja depan dalam Python. Seperti kembarannya di Java, ia tidak mengimpor apa pun dari Tkinter. 1077
Kode 11.13 `LaundryDesk.java`: kontrak-kontrak view, controller perantara, view-view teks uji mandiri, dan `main`. 1079
Kode 11.14 `DeskWindow.java`: view-view Swing meja depan. `OrderFormPanel`, `OrderTablePanel`, dan `SummaryBar` mengimplementasikan ketiga kontrak view; `DeskWindow` menempatkannya dalam sebuah *frame*. 1087
Kode 11.15 `laundry_desk.py`: *protocol* view, controller, view-view teks uji mandiri, dan `main`. 1092
Kode 11.16 `tk_desk_views.py`: view-view Tkinter meja depan. 1099
Kode 11.17 `OrderBookCheck.java`: dua puluh pemeriksaan terhadap model meja depan, termasuk harga, penolakan, mesin keadaan, list hanya-baca, dan penghentian langganan. 1101
Kode 11.18 `test_order_book.py`: *unit test* model meja depan dengan `unittest`, memakai `subTest` untuk kelima penolakan. 1104
Kode 11.19 `StudentTwoViews.java`: controller dengan sebanyak apa pun view; `StudentView` yang asli dan sebuah view baru yang dipisahkan koma. Ia memakai `Student` dan `StudentView` dari `StudentMvcDemo.java`. 1111
Kode 11.20 `student_two_views.py`: kedua view yang sama dalam Python, mengimpor model dan view pertama dari `student_mvc_demo.py`. 1113
Kode 11.21 `PickupBoard.java`: view baru atas buku pesanan untuk layar ruang tunggu, ditambahkan tanpa mengubah model. Kompilasi bersama `OrderBook.java`. 1116
Kode 11.22 `pickup_board.py`: view ruang tunggu dalam Python; objek yang memiliki `__call__` adalah *callable*, sehingga dapat didaftarkan sebagai pendengar. 1116
Kode 11.23 `MiniBinding.java`: `Property` generik yang dapat diamati dan *view model* formulir pesanan. Kompilasi bersama `OrderBook.java`. 1118
Kode 11.24 `mini_binding.py`: properti dan *view model* yang sama dalam Python. 1120

Kode 12.1 `PredictivePrototype.java`: signature sebuah kata, uji keabsahan, dan pencarian yang memindai berkas, semuanya sebagai metode statis dari satu kelas utilitas. 1129
Kode 12.2 `PrototypeCheck.java`: pemeriksaan Soal 12.1 dan 12.2, dua varian keliru, dan pengukuran waktu `String` lawan `StringBuffer`. 1130
Kode 12.3 `prototype.py`: ketiga fungsi yang sama dalam Python. Tabel `_KEY_OF` dibangun sekali, saat modul pertama kali diimpor. 1133
Kode 12.4 `prototype_check.py`: uji Soal 12.1 dan 12.2 dalam Python, dengan varian keliru yang sama dan pengukuran waktu tiga cara membangun string. 1134
Kode 12.5 `Words2SigProto.java`: signature setiap kata di baris perintah. 1139
Kode 12.6 `Sigs2WordsProto.java`: kata-kata setiap signature di baris perintah, satu baris per signature, menurut abjad 1139
Kode 12.7 `words2sig_proto.py`: program yang sama dalam Python; kata-katanya adalah `sys.argv[1:]`. 1140
Kode 12.8 `sigs2words_proto.py`: kata-kata setiap signature di baris perintah, terurut. 1140
Kode 12.9 `WordSig.java`: signature dan kata, dibandingkan menurut urutan numerik Persamaan (12.4). 1143
Kode 12.10 `DictionaryListImpl.java`: kamus sebagai `ArrayList<WordSig>` terurut, dicari dengan `Collections.binarySearch` dan penelusuran ke kedua sisi. 1144
Kode 12.11 `ListCheck.java`: pemeriksaan Soal 12.4, kedua urutan signature berdampingan, dan dua varian keliru 1146
Kode 12.12 `word_sig.py`: kelas pasangan dalam Python; `__lt__` adalah satu-satunya yang diperlukan pengurutan dan `bisect` 1148
Kode 12.13 `dictionary_list_impl.py`: *list* terurut dalam Python, dicari dengan `bisect_left` dan penelusuran ke atas. 1148
Kode 12.14 `list_check.py`: uji Soal 12.4 dalam Python, dengan dua varian keliru yang sama. 1149
Kode 12.15 `Dictionary.java`: satu metode, didokumentasikan dengan Javadoc, termasuk keadaan kosong dan tidak sah. 1152
Kode 12.16 `dictionary.py`: kontrak yang sama sebagai kelas dasar abstrak dengan *docstring* 1153
Kode 12.17 `Sigs2WordsList.java`: program baris perintah di atas `Dictionary` apa pun; yang ini membuat `DictionaryListImpl` 1154
Kode 12.18 `MakeDictionary.java`: 200.000 kata rekaan yang dapat dilafalkan, sama di setiap komputer, ditulis ke `big_words.txt`. 1155
Kode 12.19 `TimingComparison.java`: waktu pembangunan dan waktu pencarian keempat kamus pada berkas yang sama dan 2.000 signature yang sama. 1157
Kode 12.20 `sigs2words_list.py`: program *list* dalam Python. 1158
Kode 12.21 `make_dictionary.py`: kamus rekaan yang sama dalam Python. 1159
Kode 12.22 `timing_comparison.py`: keempat kamus diukur dalam Python. 1160
Kode 12.23 `DictionaryMapImpl.java`: `HashMap` dari signature ke himpunan kata, dengan `addWord` publik untuk penyisipan cepat dan hasil yang hanya-baca. 1164
Kode 12.24 `MapCheck.java`: pemeriksaan Soal 12.7, termasuk upaya mengosongkan himpunan yang dikembalikan, dan kelas *map* dengan dua metode keliru. 1165
Kode 12.25 `dictionary_map_impl.py`: *map* bernilai jamak dalam Python, mengembalikan `frozenset`. 1166
Kode 12.26 `map_check.py`: uji Soal 12.7 dalam Python, dan dua metode keliru yang sama. 1167
Kode 12.27 `DictionaryTreeImpl.java`: *trie* dengan delapan anak per simpul, diisi dan dicari melalui delegasi; hasilnya dipotong sepanjang signature 1171
Kode 12.28 `TreeCheck.java`: pemeriksaan Soal 12.8 dan pohon yang menyimpan kata hanya di ujung lintasannya. 1174
Kode 12.29 `dictionary_tree_impl.py`: *trie* yang sama dalam Python; `_Node` menjalankan rekursinya dan `_trim` berupa *set comprehension*. 1176
Kode 12.30 `tree_check.py`: uji Soal 12.8 dalam Python, dan pohon yang hanya menyimpan di ujung. 1177
Kode 12.31 `Sigs2WordsMap.java`: program baris perintah di atas `DictionaryMapImpl`. 1180
Kode 12.32 `Sigs2WordsTree.java`: program baris perintah di atas `DictionaryTreeImpl` 1180

Kode 12.33 `sigs2words_map.py`: program *map* dalam Python. 1181
Kode 12.34 `sigs2words_tree.py`: program pohon dalam Python. 1182
Kode 12.35 `DictionaryModel.java`: modelnya. Ia menyimpan kata-kata yang selesai, kandidat, dan sebuah indeks, serta menghitung ulang signature dari kata yang ditampilkan. 1186
Kode 12.36 `MessageView.java`: view, area teks hanya-baca di dalam panel bergulir. 1189
Kode 12.37 `KeypadController.java`: controller, dua belas tombol dalam kisi 4 kali 3 yang dibangun dari satu tabel tombol dan label. 1190
Kode 12.38 `PredictiveGUI.java`: program utama, dengan mode `--selftest` yang mengeklik tombol dari kode. 1191
Kode 12.39 `ModelCheck.java`: uji model sendirian, dengan kamus tiruan yang ditulis sebagai lambda. 1193
Kode 12.40 `dictionary_model.py`: model dalam Python; `from_file` adalah konstruktor alternatif, ditulis sebagai *class method*. 1195
Kode 12.41 `message_view.py`: view, *widget* `Text` Tkinter yang hanya dibuka kuncinya selagi teks model ditulis ke dalamnya. 1196
Kode 12.42 `keypad_controller.py`: controller; `make_command` memberi setiap tombol sebuah fungsi yang mengingat tombolnya sendiri. 1197
Kode 12.43 `predictive_gui.py`: program utama; `--selftest` menekan tombol-tombol melalui fungsi `make_command` yang sama dengan yang dipakai tombol. 1197
Kode 12.44 `model_check.py`: uji model sendirian; kamus tiruannya adalah objek apa saja yang memiliki metode `signature_to_words`. 1199
Kode 12.45 `CollectionChoice.java`: uji keanggotaan pada tiga koleksi, urutan penelusuran tiga himpunan, dan dua cara membuang duplikat, semuanya diukur waktunya. 1202
Kode 12.46 `collection_choice.py`: percobaan yang sama dalam Python; *list* terurut dengan `bisect` menggantikan `TreeSet`. 1205
Kode 12.47 `GenericBin.java`: kelas generik berbatas `Bin<T extends Weighed>`, metode *wildcard* `totalKg`, dan metode generik `maxOf`, beserta pemeriksaannya. 1209
Kode 12.48 `generic_bin.py`: rancangan yang sama dengan `Protocol`, `TypeVar`, dan `Generic`, diperiksa mypy sebelum dijalankan dan oleh `isinstance` selagi berjalan. 1211
Kode 12.49 `OrderSorting.java`: papan pengambilan yang diurutkan dengan rantai `Comparator` empat kunci, beserta dua *comparator* keliru yang menggoda. 1216
Kode 12.50 `order_sorting.py`: papan yang sama dengan kunci tuple dan dengan dua pengurutan stabil, serta dua urutan keliru yang sama. 1218
Kode 12.51 `BagCounterApp.java`: penghitung dalam bentuk MVC dengan dua view, uji-mandiri yang memeriksa model sendirian lalu mengeklik tombol-tombol yang sesungguhnya, dan panel tanpa model sebagai pembanding. 1221
Kode 12.52 `bag_counter.py`: penghitung yang sama dalam Python; sebuah view menerima fungsi yang menampilkan teksnya, sehingga uji dapat mengganti labelnya. 1226
Kode 12.53 `ObserverBugs.java`: subjek yang memberi tahu sebelum berubah dan menelusuri daftar *listener* yang hidup, di samping versi yang sudah diperbaiki. 1231
Kode 12.54 `observer_bugs.py`: subjek yang sama dalam Python, tempat *bug* kedua gagal tanpa suara. 1233
Kode 12.55 `KeypadClosures.java`: *listener* yang dibuat dalam gelung dengan lambda, kelas dalam anonim, dan kelas bernama, serta satu versi yang *listener*-nya berbagi satu sel yang dapat berubah. 1237
Kode 12.56 `keypad_closures.py`: perintah-perintah yang dibuat dalam gelung, satu versi dengan *bug late binding* dan tiga versi yang benar. 1239
Kode 12.57 `ExerciseSolutions.java`: pasangan dengan `equals` dan `hashCode`, signature berbasis tabel, kamus `TreeMap` dengan rentang prefiks, `invert` generik, dan sebuah *comparator*, beserta pemeriksaannya. 1246
Kode 12.58 `exercise_solutions.py`: penyelesaian yang sama dalam Python; pencarian prefiksnya memakai dua panggilan `bisect_left` pada signature yang terurut. 1249
Kode 13.1 `IteratorBasics.java`: menelusuri koleksi dengan iterator eksplisit, mengubah salinan lokal sebuah elemen, perulangan `for` yang disempurnakan, dan meminta satu elemen terlalu banyak. 1257

Kode 13.2 `iterator_basics.py`: `iter()`, `next()`, dan `StopIteration`, perulangan `for`, serta meminta satu elemen terlalu banyak. 1260
Kode 13.3 `RemoveDuplicates.java`: menghapus duplikat bersebelahan dari list terurut dengan mengingat elemen sebelumnya, versi yang menangani list kosong dan `null`, serta versi di tempat dengan `ListIterator`. 1263
Kode 13.4 `remove_duplicates.py`: tugas yang sama dalam Python, dengan penanda, dengan `itertools.groupby`, di tempat dengan menelusuri indeks dari ujung, dan berjalan mundur dengan `reversed`. 1266
Kode 13.5 `LoadPlanDemo.java`: `LoadPlan` adalah `Iterable` yang menghitung muatannya saat diminta; setiap panggilan `iterator()` mengembalikan `LoadIterator` yang mandiri. 1268
Kode 13.6 `load_plan.py`: iterable dengan kelas iterator yang ditulis dengan tangan, dan iterable yang sama yang ditulis dengan fungsi generator. 1271
Kode 13.7 `ConcurrentModification.java`: `set` selama iterasi diizinkan; `add` dan `remove` melalui list melemparkan `ConcurrentModificationException`; satu kasus lolos diam-diam; tiga alternatif yang benar. 1276
Kode 13.8 `concurrent_modification.py`: list Python yang diubah selama perulangan `for` diam-diam melompati elemen; dictionary dan set membangkitkan `RuntimeError`; tiga alternatif yang benar. 1278
Kode 13.9 `AgeSort.java`: `AgeComparator` yang ditulis dengan tangan dan juga menangani `null`, serta urutan-urutan yang sama yang dirakit dengan `Comparator.comparingInt`, `reversed`, dan `nullsFirst`. 1282
Kode 13.10 `age_sort.py`: mengurutkan dengan fungsi kunci (`attrgetter`, `lambda`), `reverse=True`, dan comparator gaya Java yang diadaptasi dengan `functools.cmp_to_key`. 1285
Kode 13.11 `NameOrder.java`: `compareTo` yang hanya menangani -1, 0, dan 1, urutan alami yang benar, comparator nama-belakang-lalu-nama-depan, dan mengapa membandingkan string hasil rekatan itu rapuh. 1287
Kode 13.12 `name_order.py`: `@dataclass(order=True)` untuk urutan alami field demi field, kunci tuple untuk urutan lain, dan perbandingan rusak yang direproduksi dengan `cmp_to_key` 1290
Kode 13.13 `CopyDemo.java`: percobaan ganti-nama-dan-tambah yang sama pada alias, salinan dangkal, dan salinan dalam dari list berisi objek `Person` 1292
Kode 13.14 `copy_demo.py`: aliasing, `list(...)` sebagai salinan dangkal, `copy.deepcopy`, dan tiga tingkat yang sama untuk list bersarang. 1295
Kode 13.15 `MedianDemo.java`: metode median yang mengurutkan list milik pemanggilnya, metode yang mengurutkan salinan, dan tampilan tak dapat diubah yang mengubah efek samping tersembunyi menjadi eksepsi. 1297
Kode 13.16 `median_demo.py`: `list.sort()` mengubah list milik pemanggil, `sorted()` tidak, dan tuple sama sekali tidak dapat diurutkan di tempat. 1298
Kode 13.17 `ImmutableCollections.java`: `final` versus imutabel, tampilan tak dapat diubah yang memperlihatkan perubahan belakangan, salinan tak dapat diubah yang tidak, metode pabrik `of`, dan ketakterubahan yang dangkal. 1300
Kode 13.18 `immutable_collections.py`: `Final` sebagai petunjuk, `MappingProxyType` sebagai tampilan hanya-baca, `tuple` dan `frozenset` sebagai salinan imutabel, kemampuan di-*hash*, dan imutabilitas yang dangkal 1302
Kode 13.19 `ImmutableClasses.java`: `Point` imutabel yang ditulis dengan tangan, `Student` dengan id seumur hidup, kelas yang membocorkan list-nya, dan record dengan salinan defensif di konstruktor ringkasnya 1305
Kode 13.20 `immutable_classes.py`: dataclass beku dengan `replace`, id yang ditetapkan saat lahir, kelas yang bocor, salinan defensif di `__post_init__`, dan `NamedTuple`. 1309
Kode 13.21 `MutableKey.java`: `Spot` yang mutabel dipakai sebagai kunci di `HashMap` dan `HashSet`, diubah selagi tersimpan; lalu record imutabel yang dipakai dengan benar. 1312
Kode 13.22 `mutable_key.py`: kunci hilang yang sama di `dict` dan `set`, obat berupa dataclass beku, dan penolakan Python untuk meng-*hash* sebuah list. 1315
Kode 13.23 `StreamsDemo.java`: perulangan dan stream yang menghitung total yang sama, kemalasan yang diperlihatkan dengan `peek`, sekali pakai, collector, dan stream tak hingga yang dibatasi dengan `limit` 1318

Kode 13.24 `streams_demo.py`: comprehension dan ekspresi generator, `map`, `filter`, dan `reduce`, kemalasan, sekali pakai, pengelompokan, dan iterator tak hingga dari `itertools`. 1320
Kode 13.25 `LaundryLedger.java`: Profitina Laundry dengan record `Money` yang imutabel, aturan harga, pesanan beku, `OrderHistory` yang hanya dapat ditambah dengan dua iterator buatan sendiri, dan laporan yang dibangun dari stream. 1324
Kode 13.26 `laundry_ledger.py`: buku besar yang sama dengan dataclass beku, iterator terbaru-lebih-dulu berbasis generator, generator per pelanggan yang malas, dan laporan yang dibangun dari comprehension dan `itertools.groupby`. 1333
Kode 13.27 `PickupDays.java`: `Iterable<LocalDate>` yang melewati hari Minggu, dengan iterator dalam anonim. 1343
Kode 13.28 `pickup_days.py`: iterable yang sama sebagai generator. 1345
Kode 13.29 `LedgerQueries.java`: lima pertanyaan lanjutan yang dijawab dengan stream, termasuk `partitioningBy`, `flatMap`, `max`, `counting`, dan `flatMapping`. 1347
Kode 13.30 `ledger_queries.py`: kelima pertanyaan yang sama dengan comprehension, `max` dengan kunci, `Counter`, dan `next` dengan nilai bawaan. 1349
Kode 13.31 `MoneySplit.java`: membagi dan mengalokasikan `Money` sehingga bagian-bagiannya selalu berjumlah sama dengan total. 1350
Kode 13.32 `money_split.py`: hal yang sama dengan `divmod` dan tuple berisi nilai `Money`. 1351
Kode 14.1 `ParentChild.java`: induk memulai dua anak yang mencacah sebagai thread, tidur tiga detik, lalu menginterupsi mereka dan menunggu mereka berakhir. 1358
Kode 14.2 `parent_child.py`: induk dan anak-anak yang sama dalam Python; sebuah `threading.Event` bersama menggantikan interupsi Java. 1361
Kode 14.3 `StopFlag.java`: thread yang berputar pada bendera `boolean` biasa mungkin tidak pernah melihat bendera itu berubah; perulangan yang sama pada bendera `volatile` langsung berhenti. 1363
Kode 14.4 `stop_flag.py`: di CPython, atribut biasa yang dipakai sebagai bendera berhenti memang menjadi terlihat; `threading.Event` tetap pilihan yang idiomatis. 1365
Kode 14.5 `CreateThreads.java`: subkelas `Thread`, sebuah `Runnable`, sebuah lambda, dan `ExecutorService` yang mengembalikan hasil melalui objek `Future`; serta apa yang terjadi bila Anda memanggil `run()` alih-alih `start()`. 1366
Kode 14.6 `create_threads.py`: subkelas `threading.Thread`, fungsi target, sebuah lambda, dan `ThreadPoolExecutor` dengan `submit` serta `map`. 1368
Kode 14.7 `ThreadStates.java`: satu thread dalam setiap keadaan dari keenam keadaan `Thread.State`, diamati dengan `getState()`. 1372
Kode 14.8 `thread_states.py`: apa yang dapat kita amati di Python tentang kehidupan sebuah thread: `is_alive()`, `ident`, `Lock.locked()`, dan `threading.enumerate()`. 1373
Kode 14.9 `SpeedupDemo.java`: mencacah bilangan prima di bawah tiga juta dalam satu thread dan dalam dua thread, beserta waktunya. 1375
Kode 14.10 `gil_demo.py`: pekerjaan yang terikat CPU dalam satu thread, dua thread, dan dua proses, serta penantian dalam satu thread dan empat thread. 1376
Kode 14.11 `RaceDemo.java`: tiga anak berbagi satu pencacah tanpa kunci apa pun; lima putaran naik-dan-turun dan lima putaran naik saja. 1379
Kode 14.12 `race_demo.py`: pembaruan yang hilang pada pencacah bersama di Python, dengan `+=` biasa dan dengan baca-ubah-tulis yang sengaja memberi giliran. 1384
Kode 14.13 `SafeCounters.java`: lima pencacah yang benar: metode synchronized, blok synchronized di pemanggil, `ReentrantLock`, `AtomicInteger`, dan `LongAdder`. 1387
Kode 14.14 `safe_counters.py`: pencacah baca-ubah-tulis yang lambat dibuat benar dengan `threading.Lock`, ditulis dengan `with` dan dengan `acquire/release` eksplisit. 1391
Kode 14.15 `ProCon.java`: seorang produsen dan seorang konsumen yang lebih lambat berbagi penyangga melingkar lima sel; `wait()` dan `notifyAll()` membuat masing-masing menunggu yang lain. 1394
Kode 14.16 `procon.py`: penyangga melingkar yang sama dengan `threading.Condition`. 1399
Kode 14.17 `QueueDemo.java`: pertukaran produsen–konsumen dengan `ArrayBlockingQueue` berkapasitas lima dan sebuah pil racun. 1401

Kode 14.18 `queue_demo.py`: pertukaran yang sama dengan `queue.Queue(maxsize=5)` dan `None` sebagai pil racun. 1403
Kode 14.19 `Philosopher.java`: lima filsuf, masing-masing sebuah thread, masing-masing mengambil garpu kiri lalu garpu kanan; `main` menggambar meja sampai setiap filsuf memegang tepat satu garpu. 1404
Kode 14.20 `philosopher.py`: para filsuf yang bersantap di Python; setiap garpu adalah sebuah `threading.Lock`. 1408
Kode 14.21 `DeadlockDemo.java`: dua thread mengambil dua kunci dalam urutan berlawanan; setelah satu detik, pendeteksi deadlock milik JVM melaporkan siklusnya. 1410
Kode 14.22 `deadlock_demo.py`: kedua petugas di Python; setelah satu detik kedua thread masih hidup dan kedua kunci masih terpegang. 1412
Kode 14.23 `DeadlockFixes.java`: filsuf kiri-lalu-kanan terkena deadlock; pengurutan kunci, `tryLock` dengan batas waktu, dan semafor empat kursi semuanya menuntaskan lima belas kali makan. 1414
Kode 14.24 `deadlock_fixes.py`: filsuf-filsuf naif dan tiga obat di Python. 1418
Kode 14.25 `Livelock.java`: dua pekerja yang selalu mengalah mengoper sebuah sumber daya bolak-balik tanpa pernah memakainya; membatasi kesopanan mereka menjadi satu kali penyerahan mengakhiri livelock itu. 1420
Kode 14.26 `livelock.py`: dua pekerja yang terlalu sopan di Python. 1424
Kode 14.27 `LaundryShop.java`: tiga thread mesin cuci mengambil pesanan dari `LinkedBlockingQueue` dan membayar ke `Ledger` yang tersinkronisasi; pil racun mematikan mereka. 1428
Kode 14.28 `laundry_shop.py`: binatu konkuren dalam Python: `WashingMachine` adalah subkelas `threading.Thread`, buku kas menjaga totalnya dengan sebuah `Lock`, dan `None` adalah pil racunnya. 1433
Kode 14.29 `WalletTransfer.java`: transfer ke dua arah antara dua dompet, dibuat bebas deadlock dengan mengunci id yang lebih kecil lebih dahulu. 1442
Kode 14.30 `wallet_transfer.py`: transfer bebas deadlock yang sama dalam Python. 1444
Kode 14.31 `ExpressQueue.java`: `PriorityBlockingQueue` membiarkan pesanan kilat menyalip pesanan biasa. 1445
Kode 14.32 `express_queue.py`: pesanan kilat lebih dahulu dengan `queue.PriorityQueue` dan tupel. 1446
Kode 15.1 `UdpHello.java`: tiga datagram UDP dikirim melalui loopback dari satu socket ke socket lain. 1453
Kode 15.2 `udp_hello.py`: tiga datagram yang sama dalam Python. 1455
Kode 15.3 `EchoServer.java`: terima satu klien dan kirim balik setiap barisnya. 1459
Kode 15.4 `echo_server.py`: server *echo* dalam Python. 1460
Kode 15.5 `EchoClient.java`: kirim baris yang diketik pengguna dan cetak apa yang kembali. 1461
Kode 15.6 `echo_client.py`: klien *echo* dalam Python. 1463
Kode 15.7 `KnockKnockProtocol.java`: protokol **Tabel 15.3** sebagai mesin keadaan, dengan `main` yang memainkan percakapan berskenario. 1467
Kode 15.8 `knock_knock_protocol.py`: mesin keadaan yang sama dalam Python. 1470
Kode 15.9 `KnockKnockServer.java`: server knock-knock yang melayani banyak klien sekaligus, satu thread kumpulan per klien, dengan `stop()` yang bersih. 1473
Kode 15.10 `knock_knock_server.py`: server multiklien dalam Python. 1476
Kode 15.11 `KnockKnockClient.java`: bercakap dengan server knock-knock dari papan ketik. 1478
Kode 15.12 `knock_knock_client.py`: klien knock-knock dalam Python. 1479
Kode 15.13 `KnockKnockLoadTest.java`: jalankan server, jalankan tiga klien bersamaan terhadapnya, lalu hentikan. 1481
Kode 15.14 `knock_knock_load_test.py`: uji konkurensi yang sama dalam Python. 1484
Kode 15.15 `Json.java`: pembaca dan penghasil JSON minimal; objek menjadi `LinkedHashMap`, larik menjadi `ArrayList`, bilangan bulat menjadi `Long`. 1487
Kode 15.16 `json_roundtrip.py`: perjalanan bolak-balik yang sama dengan modul standar `json` Python. 1494

Kode 15.17 `LaundryStatusServer.java`: layanan status pesanan Profitina Laundry, dengan kelas-kelas domainnya `Status`, `LaundryOrder`, dan `OrderBook` 1495
Kode 15.18 `laundry_status_server.py`: layanan status pesanan dalam Python. 1503
Kode 15.19 `LaundryStatusClient.java`: objek klien yang menyembunyikan format kabel, dan skenario yang menguji setiap aturan **Tabel 15.4**. 1510
Kode 15.20 `laundry_status_client.py`: objek klien dalam Python. 1514
Kode 15.21 `InteropDemo.java`: jalankan server Python sebagai proses terpisah dan bercakap dengannya lewat klien Java. 1517
Kode 15.22 `interop_demo.py`: jalankan server Java sebagai proses terpisah dan bercakap dengannya dari Python. 1519
Kode 15.23 `HttpStatusDemo.java`: server HTTP di localhost, ditanya dengan `HttpClient` dan dengan socket polos. 1523
Kode 15.24 `http_status_demo.py`: server dan pertanyaan yang sama dalam Python. 1525
Kode 15.25 `LaundryStatusCheck.java`: sepuluh pemeriksaan otomatis, masing-masing terhadap servernya sendiri pada port efemeral 1529
Kode 15.26 `test_laundry_status.py`: sepuluh uji yang sama dengan `unittest` 1532
Kode 15.27 `RetryDemo.java`: menyambung dengan *backoff* eksponensial; server muncul setelah kegagalan ketiga. 1548
Kode 15.28 `retry_demo.py`: klien yang mencoba ulang yang sama dalam Python. 1550
Kode 15.29 `ChatRoom.java`: ruang obrolan yang menyiarkan setiap baris, dengan anggota berskenario untuk demonstrasi yang dapat diulang. 1553
Kode 15.30 `chat_room.py`: ruang obrolan dalam Python. 1558
Kode 16.1 `JokeStore.java`: kontrak penyimpanan, dengan *record* `Joke` dan *stub* di memori yang dibangun oleh `JokeStore.of`. 1572
Kode 16.2 `JokeStores.java`: `JokeStore` yang berbasis berkas, penyimpan bernomor mulai 1 yang melanggar kontrak, dan satu uji kontrak yang dijalankan terhadap ketiga penyimpan. 1573
Kode 16.3 `joke_store.py`: kontrak penyimpanan sebagai kelas dasar abstrak, *dataclass* beku `Joke`, dan *stub* di memori `ListJokeStore`. 1576
Kode 16.4 `joke_stores.py`: penyimpan berkas, penyimpan bernomor mulai 1, dan kontraknya, dengan uji `unittest`. 1577
Kode 16.5 `KnockKnockProtocol.java`: tabel transisi sebagai kelas, versi keliru tanpa keadaan, dan metode bantu `play` yang menjalankan percakapan tanpa jaringan. 1583
Kode 16.6 `knock_knock_protocol.py`: mesin keadaan yang sama dalam Python, dengan uji yang sama. 1586
Kode 16.7 `KnockKnockServer.java`: *thread* penerima, satu pekerja per klien, metode statis `converse` yang menjalankan satu percakapan, dan mode satu-per-satu yang sengaja dibuat keliru. 1591
Kode 16.8 `KnockKnockClient.java`: gelung klien, `PoliteUser` yang berskrip, mode interaktif, dan uji diri yang juga memperlihatkan server satu-per-satu menelantarkan klien kedua 1594
Kode 16.9 `knock_knock_server.py`: server yang sama dengan pekerja `threading.Thread` dan sebuah kunci di sekeliling pencacahnya. 1598
Kode 16.10 `knock_knock_client.py`: klien, `PoliteUser` sebagai kelas yang dapat dipanggil, dan tiga uji termasuk klien kedua yang ditelantarkan. 1600
Kode 16.11 `ProtocolContractTest.java`: uji yang ditulis setiap anggota untuk anggota lain, dengan *stub* yang memutar ulang baris tetap, dan klien mutan yang diloloskan oleh uji yang lemah. 1605
Kode 16.12 `protocol_contract_test.py`: uji silang dalam Python, satu kelas uji per anggota. 1607
Kode 16.13 `ConcurrentClientsTest.java`: dua puluh klien yang dilepas bersama oleh gerbang mulai, diperiksa terhadap orakel, dan server yang memakai satu objek protokol bersama. 1611
Kode 16.14 `concurrent_clients_test.py`: uji terdistribusi yang sama dalam Python 1614
Kode 16.15 `FailureDetection.java`: klien yang mengubah setiap kegagalan menjadi satu kalimat, klien yang naif, server-server yang sengaja berulah, dan server yang menyadari klien yang telah pergi. 1619
Kode 16.16 `failure_detection.py`: skenario-skenario kegagalan yang sama dalam Python. 1623

Kode 16.17 `KnockKnockGui.java`: MVC dengan antarmuka `View`, pengendali yang menjalankan jaringan pada *thread*-nya sendiri, *view* Swing, dan uji diri tanpa layar. 1627
Kode 16.18 `knock_knock_gui.py`: rancangan yang sama dengan Tkinter; *thread* jaringan mengirim pembaruan *view* ke sebuah antrean yang dikosongkan *thread* Tk. 1633
Kode 16.19 `PerformanceProbe.java`: latensi dan *throughput* server *knock-knock* dengan 1, 10, dan 50 klien serentak. 1640
Kode 16.20 `performance_probe.py`: pengukuran yang sama dalam Python. 1641
Kode 16.21 `PickupSchedule.java`: larik terurut dengan iterator berupa kelas dalam, tampilan tersaring yang malas, dan pemeriksaan *fail-fast*. 1645
Kode 16.22 `pickup_schedule.py`: jadwal yang sama dengan kelas iterator eksplisit dan generator untuk tampilan malas. 1649
Kode 16.23 `Money.java`: objek nilai imutabel dengan pembuatan yang tervalidasi, kesamaan nilai, kode *hash* yang konsisten, pengurutan, dan dua varian yang rusak. 1654
Kode 16.24 `money.py`: objek nilai yang sama sebagai *dataclass* beku, dan kedua varian yang rusak. 1659
Kode 16.25 `WashCounter.java`: pencacah yang tidak aman, *race* yang sama dibuat pasti, dan tiga pencacah yang aman-*thread*, semuanya dijalankan oleh empat *thread* yang dilepas bersama. 1664
Kode 16.26 `wash_counter.py`: percobaan yang sama dalam Python. 1668
Kode 16.27 `WashLine.java`: penyangga terbatas buatan tangan, lini yang sama dengan `ArrayBlockingQueue`, pil racun, dan penyangga yang memakai `if` alih-alih `while`. 1672
Kode 16.28 `wash_line.py`: lini cuci yang sama dalam Python. 1677
Kode 16.29 `LaundryProtocol.java`: protokol sebagai fungsi murni, server banyak-klien, klien dengan batas waktu balasan, dan klien yang lupa melakukan *flush*. 1682
Kode 16.30 `laundry_protocol.py`: protokol, server, dan klien yang sama dalam Python. 1687
Kode 16.31 `LaundryCore.java`: model proyek puncak: eksepsi *checked*, antarmuka `PriceRule`, hierarki layanan, *enum* status, *record* `Order` yang imutabel, repositori berkas, dan `Shop` yang aman-*thread*. 1694
Kode 16.32 `ProfitinaLaundry.java`: pengendali, *view*, API soket dengan balasan yang diakhiri titik, klien, dan uji diri. 1698
Kode 16.33 `laundry_core.py`: model dalam Python, dengan `RLock` yang melindungi toko. 1705
Kode 16.34 `profitina_laundry.py`: pengendali, API soket, klien, dan uji dalam Python. 1710
Kode 16.35 `ExerciseSolutions.java`: `Countdown`, penghapusan yang benar di dalam gelung, `split`, `TOTAL`, dan pencacah `READY` yang diuji melalui jaringan. 1725
Kode 16.36 `exercise_solutions.py`: penyelesaian yang sama dalam Python, dan penyimpan lelucon SQLite untuk Latihan 8. 1729

Daftar Pseudokode

Pseudokode 1.1 Prosedur menerima pesanan *laundry*, ditulis cukup cermat sehingga dapat diikuti oleh orang yang belum pernah bekerja di kios itu. 2
Pseudokode 10.1 Gelung peristiwa yang dijalankan setiap toolkit GUI: tunggu, ambil peristiwa berikutnya, panggil listener yang terdaftar untuknya, dan gambar ulang saat senggang. 917

Daftar Persamaan

(1.1) 3
(1.2) 3
(1.3) 29

(1.4) 29
(2.1) 83
(2.2) 101
(2.3) 101
(2.4) 108
(3.1) 161
(3.2) 167
(3.3) 170
(3.4) 171
(3.5) 186
(3.6) 188
(3.7) 188
(3.8) 191
(3.9) 196
(3.10)....196
(3.11)....208
(4.1) 222
(4.2) 222
(4.3) 229
(4.4) 229
(4.5) 229
(4.6) 244
(4.7) 251
(4.8) 304
(4.9) 309
(5.1) 325
(5.2) 339
(5.3) 339
(5.4) 381
(6.1) 419
(6.2) 432
(6.3) 442
(6.4) 461
(6.5) 476
(6.6) 494
(6.7) 497
(6.8) 497
(6.9) 499
(7.1) 579
(7.2) 587
(7.3) 587
(7.4) 604
(7.5) 609
(7.6) 615
(7.7) 620
(7.8) 620
(7.9) 620
(7.10)....620
(8.1) 657
(8.2) 688
(8.3) 693

(8.4) 704
(8.5) 720
(8.6) 729
(8.7) 738
(9.1) 794
(9.2) 795
(9.3) 807
(9.4) 816
(9.5) 816
(9.6) 817
(9.7) 849
(9.8) 875
(9.9) 875
(10.1) 943
(10.2) 947
(10.3) 948
(10.4) 967
(10.5) 976
(10.6) 976
(10.7) 976
(10.8) 1007
(11.1) 1019
(11.2) 1034
(11.3) 1046
(11.4) 1069
(11.5) 1069
(11.6) 1070
(12.1) 1126
(12.2) 1128
(12.3) 1138
(12.4) 1142
(12.5) 1143
(12.6) 1143
(12.7) 1155
(12.8) 1164
(12.9) 1171
(12.10) 1221
(13.1) 1259
(13.2) 1275
(13.3) 1282
(13.4) 1282
(13.5) 1282
(13.6) 1312
(13.7) 1312
(13.8) 1317
(13.9) 1324
(13.10) 1350
(14.1) 1376
(14.2) 1383
(14.3) 1384
(14.4) 1394

(14.5)....1414
(14.6)....1433
(15.1)....1451
(15.2)....1451
(15.3)....1472
(15.4)....1521
(15.5)....1521
(15.6)....1527
(16.1)....1582
(16.2)....1639
(16.3)....1639
(16.4)....1653
(16.5)....1663
(16.6)....1663
(16.7)....1732

Kata Pengantar

Setiap program berguna yang pernah Anda pakai — aplikasi yang memberi tahu kapan cucian Anda selesai, gim di ponsel Anda, situs tempat Anda membayar tagihan — dibangun dari *objek*: potongan perangkat lunak kecil yang mandiri, yang masing-masing mengetahui sesuatu dan dapat melakukan sesuatu, dan yang bekerja sama dengan saling mengirim pesan. Buku ini mengajari Anda berpikir dalam objek dan membangun dengan objek, dalam dua bahasa yang paling banyak dipakai dunia untuk tujuan itu: **Java** dan **Python**.

Buku ini tumbuh dari mata kuliah Pemrograman Berorientasi Objek (EF234302) di Departemen Teknik Informatika, Institut Teknologi Sepuluh Nopember (ITS), Surabaya, dan telah disusun ulang sehingga dapat berdiri sendiri sepenuhnya. Keenam belas babnya mengikuti irama satu semester yang berlangsung enam belas minggu, tetapi Anda tidak memerlukan kelas, dosen, atau dokumen lain untuk belajar darinya.

Untuk siapa buku ini ditulis

Anda dianggap belum tahu apa-apa. Jika Anda belum pernah menulis program, belum pernah memasang perkakas pemrograman, belum pernah mendengar kata *kelas* dipakai di luar gedung sekolah, Andalah pembaca yang dibayangkan saat buku ini ditulis. Bab 1 dimulai dari komputer kosong dan berakhir dengan dua program pertama Anda berjalan, satu dalam setiap bahasa. Sejak itu, setiap gagasan diperkenalkan dengan kata-kata sederhana, lalu ditunjukkan dalam Java, lalu dalam Python, lalu dijalankan, lalu dijelaskan baris demi baris — karena program yang sudah Anda jalankan dan pahami bernilai sepuluh kali program yang hanya Anda baca.

Mengapa dua bahasa

Java dan Python sama-sama berorientasi objek, tetapi keduanya membuat pilihan yang berbeda: Java memeriksa tipe sebelum program berjalan dan meminta Anda menyatakan segalanya secara tersurat; Python memeriksa saat program berjalan dan membiarkan Anda menulis jauh lebih sedikit. Melihat gagasan yang sama ditulis dengan dua cara adalah jalan tercepat untuk memahami gagasan itu sendiri, bukan sekadar kebiasaan satu bahasa. Karena itu, setiap contoh dalam buku ini berpasangan — program Java dan program Python yang mengerjakan hal yang sama dan mencetak keluaran yang sama — dan keduanya utuh: Anda dapat menyalin salah satunya ke sebuah berkas lalu menjalankannya.

Isi buku ini

Bab 1 sampai 3 menyiapkan meja kerja Anda (VS Code, Java Development Kit, dan Python, di Windows, macOS, atau Linux), mengajarkan lingkungan pengembangan dan *debugger*, serta membangun kelas-kelas pertama: objek, string, tipe, percabangan, dan perulangan. Bab 5 sampai 7 menambahkan larik dan list, masukan, eksepsi dan berkas, dokumentasi dan pengujian, antarmuka, pewarisan, polimorfisme, serta kontrol akses. Bab 9 sampai 11 membahas koleksi dan generik, antarmuka grafis dan peristiwa, serta pola Model–View–Controller. Bab 13 sampai 15 membahas iterator dan imutabilitas, *thread* beserta bahayanya, dan jaringan dengan *socket*, ditutup dengan sebuah proyek tim.

Bab 4, 8, 12, dan 16 adalah bab **Evaluasi dan Pembahasan**. Masing-masing menutup seperempat bagian buku dengan menyelesaikan persoalan utuh secara tuntas — setiap persoalan dinyatakan ulang, dirancang di kertas, diprogram dalam kedua bahasa, diuji, dijalankan, dan dijelaskan — lengkap dengan penyelesaian keliru yang menggoda beserta pembongkarannya, dan varian "uji diri" di akhir setiap persoalan.

Di sepanjang buku, satu usaha kecil tumbuh bab demi bab: **Profitina Laundry**, sebuah binatu yang pelanggan, pesanan, layanan, mesin, dan nota-notanya menjadi kelas, antarmuka, koleksi, jendela, *thread*, dan layanan jaringan seiring berjalannya buku. Setiap bab juga memiliki subbab *Profitina dalam Praktik* yang menunjukkan di mana gagasan yang sama muncul dalam produk nyata.

Setiap program dalam buku ini sudah dikompilasi dan dijalankan sebelum keluarannya dicetak. Program-program itu dikumpulkan, bab demi bab, dalam folder `Code` yang menyertai buku ini, dengan README di setiap folder bab yang menjelaskan cara menjalankan setiap program di terminal dan di VS Code, pada Windows, macOS, dan Linux.

Cara membacanya

Dengan komputer menyala. Ketikkan programnya, jangan hanya membacanya; jalankan; ubah satu hal lalu jalankan lagi; rusakkan dengan sengaja lalu baca pesan galatnya sampai masuk akal. Semua soal latihan di akhir tiap bab dilengkapi penyelesaian lengkap — gunakan sesudah usaha Anda sendiri, bukan sebagai penggantinya.

Gambar, tabel, kode, pseudokode, dan persamaan yang bernomor semuanya adalah tautan hidup: pada edisi elektronik, klik "Kode 5.3" atau "[4]" di bagian mana pun dan Anda langsung dibawa ke sana. Daftar Isi serta daftar-daftar di bagian depan buku bekerja dengan cara yang sama.

Tentang penulis

Irfan Subakti menggambarkan dirinya dalam satu baris — *Father | Educator | Entrepreneur | Bridging Tech & Nature: CS Lecturer × Crypto/Digital Business × AI/ML Specialist × Fruit Farmer | Storyteller* — ayah, pendidik, wirausahawan, penjembatan teknologi dan alam: dosen ilmu komputer, pelaku bisnis kripto dan digital, spesialis AI/ML, petani buah, dan pencerita. Setiap bagian dari baris itu ikut masuk ke dalam buku ini. Ia mengajar di Departemen Teknik Informatika ITS sejak 1998. Ia membimbing dan menguji tugas akhir mahasiswa sarjana, mengajar MLOps dalam program IBM SkillsBuild, dan membangun produk: ia mendirikan Profitina, platform visibilitas AI yang rekayasanya muncul di subbab "Profitina dalam Praktik" pada setiap bab. Jauh dari papan ketik, ia merawat kebun buah dan durian keluarga — di sanalah ia belajar bahwa perangkat lunak yang baik, seperti kebun yang baik, ditumbuhkan dengan sabar, satu bagian yang terawat demi satu bagian. Jiwa penceritanya adalah alasan buku ini memilih kisah ketimbang daftar, setiap kali sebuah kisah bisa mengantarkan maknanya.

Ucapan terima kasih

Bentuk buku ini berutang pada para mahasiswa yang mempelajari versi-versi awalnya dan mengajukan pertanyaan yang menyingkapkan di mana sebuah penjelasan masih tipis, serta pada rekan-rekan di ITS yang bersama-sama mengajarkan dan menyempurnakan mata kuliah ini dari tahun ke tahun. Standarnya berutang pada buku-buku klasik tentang perancangan berorientasi objek serta dokumentasi resmi Java dan Python, yang dikutip di setiap tempat penggunaannya.

Surabaya, September 2026



Irfan Subakti · yifana@gmail.com

Profitina

profitina.com

Bab 1

Objek, Program, dan Meja Kerja Pertama Anda

Di akhir bab ini Anda akan mengetahui apa itu program dan bagaimana sebuah komputer sampai dapat menjalankannya; Anda akan mampu memandang sebuah tempat biasa, misalnya kios *laundry* kecil, lalu melihat di dalamnya *objek-objek* yang akan menjadi bahan sebuah program berorientasi objek, masing-masing dengan *keadaan* (*state*) dan *perilaku* (*behavior*) sendiri, dan masing-masing menjawab *pesan* (*message*) yang dikirimkan objek lain kepadanya; Anda akan tahu mengapa buku ini mengajar dalam dua bahasa, Java dan Python, dan apa keunggulan masing-masing; Anda akan memiliki meja kerja pemrogram yang berfungsi di komputer Anda sendiri, baik yang menjalankan Windows, macOS, maupun Linux, lengkap dengan Java Development Kit, interpreter Python, dan Visual Studio Code yang sudah terpasang, diperiksa, dan saling terhubung; dan Anda akan sudah mengetik, menjalankan, dan men-*debug* program pertama Anda dalam kedua bahasa itu, diakhiri dengan kelas sungguhan pertama Anda, sebuah `Order` (pesanan) *laundry* yang mengetahui harganya sendiri dan menolak diambil sebelum cuciannya bersih.

1.1 Tempat Kita Memulai

Di sebuah sudut jalan di Surabaya ada kios *laundry* kecil. Di sana ada meja layanan dengan timbangan di atasnya, tiga mesin cuci berjajar di dinding belakang, dua mesin pengering, sebuah papan setrika, rak berisi kantong plastik yang diikat tali warna-warni, dan sebuah buku catatan tebal. Pemiliknya, Bu Sari, membuka kios pukul tujuh pagi. Seorang pelanggan masuk membawa sekantong pakaian. Bu Sari meletakkan kantong itu di atas timbangan — tiga setengah kilogram — bertanya apakah pelanggan menginginkan layanan biasa atau ekspres, menulis satu baris di buku catatan, menyobek selembar nota, dan menyebutkan harganya. Pakaian itu masuk ke mesin cuci, lalu ke pengering, lalu ke meja lipat, lalu ke dalam kantong di rak. Dua hari kemudian pelanggan kembali membawa notanya, membayar, dan membawa pulang pakaian yang sudah bersih.

Tidak ada satu kata pun tentang komputer dalam cerita itu, tetapi cerita itu sudah penuh dengan gagasan yang menjadi pokok buku ini. Ada sebuah **prosedur** — timbang, tanya, catat, hitung harga, cuci, keringkan, lipat, serahkan — yang diikuti dengan cara yang sama setiap kali. Ada **informasi** — nama pelanggan, berat, jenis layanan, harga, apakah pakaian masih di mesin atau sudah di rak. Dan ada **benda-benda**, masing-masing dengan informasi dan kemampuannya sendiri: nota mengetahui nomor dan harganya; mesin cuci mengetahui apakah ia sedang berputar dan berapa banyak yang dapat ditampungnya; buku catatan mengetahui semua pesanan hari itu. Ketika Bu Sari mempekerjakan pegawai kedua, ia tidak menjelaskan seluruh kios dalam satu tarikan napas. Ia berkata: *beginilah cara kerja nota, beginilah cara kerja mesin, beginilah yang kamu katakan kepada pelanggan*. Ia menjelaskan kiosnya satu jenis benda demi satu jenis benda.

Program komputer untuk kios ini dapat ditulis dengan dua cara yang sama seperti cara Bu Sari menjelaskannya. Program itu dapat ditulis sebagai satu daftar panjang instruksi yang memindah-mindahkan angka, atau sebagai sebuah masyarakat kecil **objek** — nota, mesin, pelanggan — yang

masing-masing menjaga informasinya sendiri dan mengerjakan tugasnya sendiri bila diminta. Cara kedua disebut **pemrograman berorientasi objek** (*object-oriented programming*), dan beginilah sebagian besar perangkat lunak yang Anda pakai sehari-hari disusun: aplikasi perbankan di ponsel Anda, toko daring tempat Anda membeli pengisi daya, gim yang Anda mainkan di bus, dan penyunting yang akan Anda pasang nanti di bab ini.

Buku ini mengajarkan pemrograman berorientasi objek sejak awal, dalam dua bahasa yang berdampingan: **Java** dan **Python**. Setiap gagasan dijelaskan lebih dahulu dengan kata-kata biasa, umumnya dengan kios *laundry* sebagai contoh. Kemudian gagasan itu ditunjukkan sebagai program Java yang lengkap, lalu sebagai program Python lengkap yang melakukan hal yang persis sama, lalu keduanya dijalankan dan keluarannya yang sesungguhnya dicetak, lalu setiap barisnya dijelaskan. Anda tidak akan pernah diminta menerima sebuah program yang tidak dapat Anda jalankan sendiri.

Buku ini tidak mengandaikan apa pun. Jika Anda pernah menulis program C atau C++ dan mengenal kata seperti *variabel*, *fungsi*, *larik* (*array*), dan *pointer*, Anda akan melaju cepat di halaman-halaman awal, dan buku ini akan menunjukkan di mana Java dan Python sejalan dengan C dan di mana keduanya berpisah jalan. Jika Anda belum pernah menulis satu baris kode pun atau membuka jendela terminal, Anda sama-sama disambut: setiap istilah dijelaskan saat pertama kali muncul, setiap langkah pemasangan dituliskan untuk Windows, macOS, dan Linux, dan setiap program lengkap serta mencetak persis apa yang dikatakan buku ini, karena keluaran yang Anda baca dihasilkan dengan menjalankannya.

Bab ini berjalan dari meja layanan ke papan ketik: apa itu program (Subbab 1.2), cara melihat objek (Subbab 1.3 dan 1.4), mengapa Java dan Python (Subbab 1.5), perkakas dan pemasangannya (Subbab 1.6 dan 1.7), kelas pertama Anda (Subbab 1.8), peta perjalanan ke depan (Subbab 1.9), dan gagasan-gagasan ini dalam sebuah produk nyata (Subbab 1.10).

1.2 Apa Itu Program

1.2.1 Resep untuk meja layanan

Andaikan Bu Sari pergi berlibur dan menyerahkan kios kepada keponakannya, Dimas. Dimas bersedia membantu, tetapi belum pernah bekerja di kios *laundry*. Maka Bu Sari menuliskan, pada sebuah kartu yang ditempel di meja layanan, apa persisnya yang harus dilakukan ketika pelanggan datang. **Pseudokode 1.1** adalah kartu itu, ditulis sedikit lebih cermat daripada yang biasa dilakukan orang.

Pseudokode 1.1 Prosedur menerima pesanan *laundry*, ditulis cukup cermat sehingga dapat diikuti oleh orang yang belum pernah bekerja di kios itu.

```
TAKE-ORDER(customer, laundry)
1  w = berat cucian di atas timbangan, dalam kilogram
2  tanyakan kepada pelanggan: biasa atau ekspres?
3  p = round(7000 * w)                    // Rp7.000 per kilogram
4  if pelanggan memilih ekspres
5      p = p + floor(p / 2)               // ekspres lebih mahal separuhnya
6  tulis nota: nomor, pelanggan, w, ekspres atau tidak, p, status "RECEIVED"
7  berikan potongan nota kepada pelanggan dan minta p rupiah
```

Perhatikan apa yang membuat kartu itu berguna. Setiap baris menyatakan satu hal. Tidak ada yang diserahkan pada selera: “round” (bulatkan) dan “floor” (bulatkan ke bawah) menyatakan dengan tepat apa yang dilakukan terhadap pecahan rupiah. Baris-baris diikuti berurutan dari atas

ke bawah, kecuali baris 5 yang dikerjakan hanya *jika* syarat pada baris 4 bernilai benar. Kartu itu menyebut dua keping informasi yang berubah dari pelanggan ke pelanggan — berat w dan harga p — dan memberi masing-masing nama pendek agar baris-baris berikutnya dapat merujuknya. Kartu seperti ini, sebuah daftar berhingga langkah-langkah yang tepat yang mengubah suatu masukan (pelanggan dan sekantong cucian) menjadi suatu keluaran (nota dan harga), disebut **algoritma**.

> **DEFINISI.** *Algoritma* adalah barisan berhingga langkah-langkah yang tepat yang mengubah suatu masukan menjadi suatu keluaran. *Program* adalah algoritma yang ditulis dalam bahasa pemrograman, sedemikian cermat sehingga komputer dapat melaksanakannya.

Harga pada kartu itu dapat ditulis sebagai dua rumus kecil. Untuk berat w kilogram, harga layanan biasa dalam rupiah adalah

$$p(w) = \text{round}(7000\, w) \tag{1.1}$$

dan harga ekspres menambahkan separuhnya, dibulatkan ke bawah menjadi rupiah utuh:

$$p_{\text{ekspres}}(w) = p(w) + \left\lfloor \frac{p(w)}{2} \right\rfloor \tag{1.2}$$

Lambang $\lfloor x \rfloor$, dibaca "*floor* dari x", berarti bilangan bulat terbesar yang tidak lebih besar daripada x; jadi $\lfloor 12250.0 \rfloor = 12250$ dan $\lfloor 7.9 \rfloor = 7$. Mari kita coba Persamaan (1.1) dan Persamaan (1.2) pada kantong Ani seberat 3.5 kilogram: $p(3.5) = \text{round}(7000 \times 3.5) = \text{round}(24500) = 24500$, sehingga cucian biasa berharga Rp24.500. Pada 5 kilogram milik Budi, layanan ekspres: $p(5) = 35000$, dan $p_{\text{ekspres}}(5) = 35000 + \lfloor 35000/2 \rfloor = 35000 + 17500 = 52500$. Ingatlah kedua angka itu; Anda akan melihat komputer mencetaknya beberapa kali di bab ini.

1.2.2 Bahasa untuk manusia, bahasa untuk mesin

Dimas dapat membaca kartu Bu Sari karena kartu itu ditulis dalam bahasa Indonesia, bahasa yang mereka berdua pahami. Komputer tidak dapat membacanya. Prosesor di dalam komputer hanya memahami **bahasa mesin**-nya sendiri: deretan panjang angka, yang masing-masing berarti sesuatu yang amat kecil, seperti "jumlahkan dua bilangan ini" atau "salin bilangan ini ke tempat itu di memori". Menulis program *laundry* langsung dalam bahasa mesin sama seperti menulis kartu Bu Sari satu gerakan otot demi satu gerakan otot.

Bahasa pemrograman berada di antara keduanya. Ia cukup tepat sehingga sebuah program komputer dapat menerjemahkannya ke bahasa mesin tanpa menebak-nebak, dan cukup mudah dibaca sehingga manusia dapat menulisnya, membacanya kembali minggu depan, dan menjelaskannya kepada rekan kerja. Java dan Python adalah dua bahasa semacam itu. Berikut baris 3 kartu itu dalam keduanya; jangan khawatir dulu tentang setiap lambangnya:

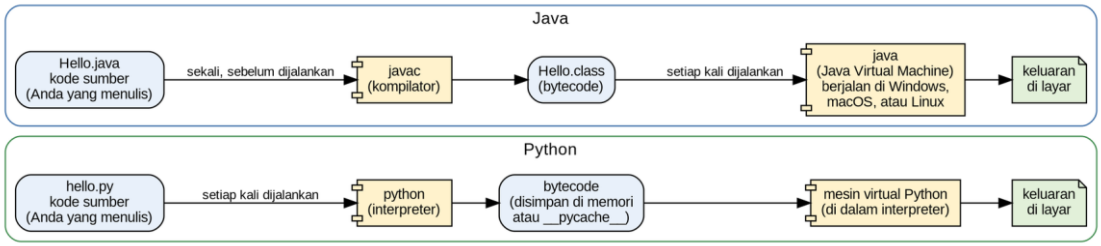
```
long p = Math.round(7000 * w);      Java
p = round(7000 * w)                 Python
```

Kedua baris itu menyatakan hal yang sama dengan tata bahasa yang sedikit berbeda. Java mengharuskan Anda menyebutkan *jenis* nilai yang akan disimpan `p` (`long`, bilangan bulat yang dapat sangat besar) dan mengakhiri setiap pernyataan dengan titik koma. Python membiarkan nilai itu sendiri menentukan jenisnya dan mengakhiri pernyataan di akhir baris. Bab 3 membahas perbedaan ini dengan cermat; untuk sekarang cukuplah melihat bahwa keduanya jauh lebih dekat ke kartu itu daripada ke bahasa mesin.

Teks yang Anda tulis dalam bahasa pemrograman disebut **kode sumber** (*source code*), dan berkas yang menyimpannya disebut **berkas sumber**. Berkas sumber Java berakhiran `.java`; berkas sumber Python berakhiran `.py`. Keduanya teks polos: Anda dapat menulisnya di Notepad, meskipun di Subbab 1.7 Anda akan memasang sesuatu yang jauh lebih nyaman.

1.2.3 Dari kode sumber menjadi program yang berjalan

Seseorang harus menerjemahkan kode sumber Anda menjadi instruksi yang dapat diikuti prosesor. Ada dua cara klasik untuk itu, dan Java maupun Python masing-masing memakai sedikit dari keduanya. **Gambar 1.1** memperlihatkan kedua perjalanan itu berdampingan.



Gambar 1.1 Bagaimana kode sumber menjadi program yang berjalan: Java dikompilasi sekali menjadi *bytecode* yang dijalankan Java Virtual Machine; Python dikompilasi menjadi *bytecode* setiap kali dijalankan, di dalam interpreter.

Kompilator (*compiler*) menerjemahkan seluruh berkas sumber terlebih dahulu dan menghasilkan berkas baru yang dapat dijalankan kemudian, berkali-kali, tanpa kode sumbernya. **Interpreter** membaca kode sumber dan langsung melaksanakannya, setiap kali Anda menjalankannya. Kompilator ibarat penerjemah yang menerjemahkan sebuah buku sekali lalu mencetaknya; interpreter ibarat penerjemah yang berdiri di samping Anda dan menerjemahkan secara lisan sambil jalan.

Java memakai kompilator bernama `javac`. Ia membaca `Hello.java` dan menulis `Hello.class`. Tetapi `Hello.class` tidak berisi bahasa mesin prosesor Anda. Isinya adalah ***bytecode***: instruksi untuk sebuah komputer khayalan yang dibakukan, yang disebut **Java Virtual Machine** (JVM, mesin virtual Java). Perintah `java` menyalakan sebuah JVM, yang membaca *bytecode* itu dan menjalankannya pada prosesor sungguhan di bawahnya. Karena setiap JVM memahami *bytecode* yang sama, `Hello.class` yang persis sama dapat berjalan di Windows, di macOS, di Linux, di server di pusat data, dan, dalam bentuk yang disesuaikan, di ponsel dan kartu pintar. Inilah janji Java yang terkenal, *write once, run anywhere* (tulis sekali, jalankan di mana saja), dan Subbab 1.5 kembali membahasnya.

Python, dilihat dari luar, tampak seperti interpreter murni: Anda mengetik `python hello.py` dan program pun berjalan. Di dalamnya, interpreter lebih dahulu mengompilasi kode sumber Anda menjadi *bytecode* Python — dengan sangat cepat, dan biasanya tanpa menyimpannya, meskipun untuk berkas yang diimpor oleh berkas lain ia menyimpan salinannya di folder bernama `__pycache__` — lalu sebuah mesin virtual Python di dalam interpreter melaksanakan *bytecode* itu. Anda tidak pernah menjalankan kompilator terpisah, dan itulah salah satu sebab Python terasa begitu ringan di awal.

Perbedaan praktisnya sederhana. Java memiliki dua langkah, **kompilasi** (`javac`) lalu **jalankan** (`java`), dan setelah mengubah kode sumber Anda harus mengompilasi lagi, kalau tidak Anda akan menjalankan versi lama. Python memiliki satu langkah: **jalankan** (`python`). Subbab 1.7 menunjukkan bagaimana VS Code menyembunyikan bahkan langkah-langkah ini di balik satu tombol.

1.2.4 Data dan operasi terhadapnya

Setiap program, sebesar apa pun, tersusun dari dua macam bahan. Ada **data**: bilangan, kata, dan fakta ya-atau-tidak yang diolah program — 3.5 kilogram, nama "Ani", fakta bahwa sebuah pesanan adalah ekspres. Dan ada **operasi**: hal-hal yang dilakukan terhadap data — mengalikan berat dengan harga per kilogram, membandingkan sebuah status dengan "READY", mencetak sebuah baris. Dalam **Pseudokode 1.1**, baris 1 dan 6 sebagian besar mengumpulkan dan mencatat data, sedangkan baris 3 dan 5 sebagian besar mengolahnya.

Sebagian besar sejarah bahasa pemrograman adalah kisah tentang cara menjaga data dan operasi tetap *teratur* ketika program tumbuh dari sepuluh baris menjadi sepuluh juta baris. Jawaban awal adalah memisahkan keduanya: data di satu tempat, operasi sebagai prosedur-prosedur terpisah yang menerima data itu. Jawaban berorientasi objek, yang diperkenalkan berikut ini, adalah menyatukan setiap keping data dengan operasi-operasi yang menjadi miliknya. Untuk program yang memodelkan dunia yang penuh benda — kios, bank, gim, rumah sakit — jawaban kedua terbukti lebih mudah dipahami, lebih mudah diubah, dan jauh lebih sulit dirusak secara tidak sengaja.

1.3 Berpikir dalam Objek

1.3.1 Melihat-lihat di dalam kios

Berdirilah lagi di meja layanan Bu Sari dan lihatlah sekeliling, kali ini sebagai pemrogram. Ajukan dua pertanyaan kepada setiap *benda* yang Anda lihat: *apa yang ia ketahui?* dan *apa yang dapat ia lakukan?*

Nota pesanan mengetahui nomornya, nama pelanggan, berat, apakah layanannya ekspres, dan di mana cucian itu berada sekarang: baru diterima, di dalam mesin cuci, siap di rak, atau sudah diambil. Ia dapat memberi tahu harganya. Ia dapat dipindahkan ke tahap berikutnya: dari diterima ke dicuci, dari dicuci ke siap, dari siap ke diambil.

Mesin cuci mengetahui namanya ("Mesin A"), berapa kilogram yang dapat ditampungnya, berapa kilogram yang ada di dalamnya sekarang, dan apakah ia sedang berputar. Ia dapat diisi, dinyalakan, dan dapat menyelesaikan satu siklus lalu dikosongkan. Dan ia punya pendirian: ia menolak muatan yang akan membuatnya terlalu berat, dan ia menolak diisi ketika sedang berputar.

Pelanggan mengetahui nama dan nomor teleponnya, dan dapat menyerahkan cucian, membayar, dan mengambil cucian.

Kasir — Bu Sari, atau Dimas — mengetahui daftar harga dan pendapatan hari itu, dan dapat menerima pesanan, menerima pembayaran, dan memberi uang kembalian.

Tabel 1.1 menghimpun pengamatan ini. Bacalah perlahan: tabel inilah, dan bukan kode apa pun, tempat sebuah program berorientasi objek bermula.

Tabel 1.1 Benda-benda di kios *laundry*, masing-masing digambarkan oleh apa yang diketahuinya dan apa yang dapat dilakukannya.

Benda di kios	Yang diketahuinya (keadaan)	Yang dapat dilakukannya (perilaku)
Nota pesanan	nomor, pelanggan, berat, ekspres atau tidak, status	memberi tahu harganya; mulai dicuci; menjadi siap; diambil
Mesin cuci	nama, kapasitas, kilogram di dalamnya, berputar atau tidak	menerima muatan; mulai; selesai dan dikosongkan
Pelanggan	nama, nomor telepon	menyerahkan cucian; membayar; mengambil
Kasir	daftar harga, uang kas hari ini	menerima pesanan; menerima pembayaran; memberi kembalian

1.3.2 Keadaan, perilaku, dan identitas

Kedua kolom **Tabel 1.1** memiliki nama. Apa yang diketahui sebuah objek adalah **keadaannya** (*state*). Apa yang dapat dilakukannya adalah **perilakunya** (*behavior*). Sifat ketiga begitu jelas di dunia nyata sehingga jarang kita sadari: setiap objek adalah *dirinya sendiri* dan bukan yang lain. Dua pelanggan mungkin membawa kantong dengan berat yang persis sama pada pagi yang sama untuk layanan yang sama, dan nota mereka mungkin mencantumkan berat, layanan, dan harga yang persis sama — tetapi keduanya tetap dua nota yang berbeda, dan menyerahkan pakaian bersih milik Ani kepada Budi adalah kesalahan serius. Sifat ini disebut **identitas** (*identity*).

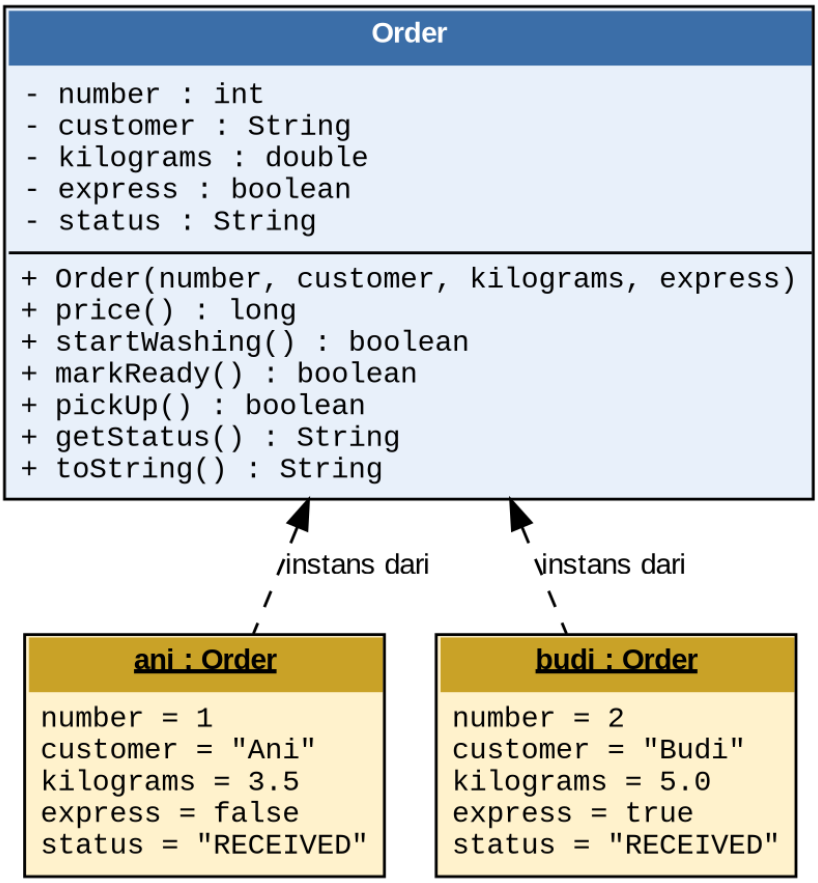
> **DEFINISI.** *Objek* adalah sesuatu di dalam program yang memiliki *keadaan* (informasi yang disimpannya pada suatu saat), *perilaku* (operasi yang dapat dilakukannya bila diminta), dan *identitas* (ia dapat dibedakan dari setiap objek lain, bahkan dari objek yang keadaannya identik).

Keadaan berubah seiring waktu; itulah yang membuat objek menarik. Pukul tujuh pagi nota Ani bertuliskan “RECEIVED” (diterima); pukul sepuluh “WASHING” (dicuci); pukul dua siang “READY” (siap). Nota itu tetap nota yang sama sepanjang waktu — identitasnya tidak berubah — tetapi keadaannya berubah. Perilaku adalah *cara* keadaan berubah: nota tidak melompat dari “RECEIVED” ke “PICKED UP” (diambil) secara ajaib, melainkan melalui serangkaian langkah yang terdefinisi dengan baik, dan sebagian langkah ditolak pada saat-saat tertentu. Anda tidak dapat mengambil cucian yang masih di dalam mesin.

1.3.3 Kelas: formulir kosong

Bu Sari tidak merancang nota baru untuk setiap pelanggan. Ia memiliki satu blok **formulir kosong**, semuanya dicetak sama: ruang untuk nomor, ruang untuk nama, ruang untuk berat, kotak centang untuk ekspres, ruang untuk harga. Setiap pagi ia mengisi satu formulir untuk setiap pelanggan. Formulir kosong menggambarkan apa yang akan diketahui setiap nota; setiap nota yang sudah diisi adalah satu nota tertentu dengan nilai-nilai tertentu.

Dalam pemrograman berorientasi objek, formulir kosong itu disebut **kelas** (*class*), dan setiap nota yang sudah diisi adalah sebuah **objek**, yang juga disebut ***instance*** (instans) dari kelas itu. Kelas `Order` menyatakan: *setiap pesanan memiliki nomor, pelanggan, berat, penanda ekspres, dan status; dan setiap pesanan dapat memberi tahu harganya, mulai dicuci, menjadi siap, dan diambil*. Objek `ani` dan `budi` adalah dua pesanan tertentu, masing-masing dengan nilainya sendiri. **Gambar 1.2** menggambar kelas itu dan dua objeknya dalam notasi yang dipakai para pemrogram, **Unified Modeling Language** (UML) [1].



Gambar 1.2 Kelas `Order` (atas) adalah formulir kosong: ia mendaftar keadaan yang dimiliki setiap pesanan dan perilaku yang ditawarkan setiap pesanan; objek `ani` dan `budi` (bawah) adalah dua nota yang sudah diisi.

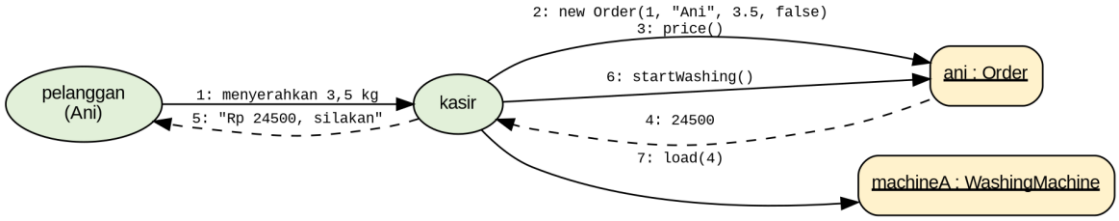
Bacalah kotak kelas itu dari atas ke bawah. Bagian pertama memuat nama kelas. Bagian kedua mendaftar ***field*** (atribut) — keping-keping keadaan yang akan dibawa setiap objek — masing-masing dengan tipenya: `int` untuk bilangan bulat, `String` untuk teks, `double` untuk bilangan berpecahan, `boolean` untuk benar-atau-salah. Tanda minus di depan setiap *field* berarti *private*: hanya pesanan itu sendiri yang boleh melihat atau mengubahnya. Bagian ketiga mendaftar **metode** (*method*) — perilakunya — dengan tanda plus yang berarti *public*: siapa pun boleh meminta pesanan melakukan hal-hal ini. Metode pertama, `Order(number, customer, kilograms, express)`, istimewa: ia adalah **konstruktor** (*constructor*), operasi yang mengisi sebuah formulir kosong baru.

Kotak objek sengaja digambar berbeda. Namanya digarisbawahi dan ditulis sebagai `nama : Kelas`, dan kotak itu menampilkan *nilai*, bukan tipe. Kedua objek memiliki *field* bernama `customer`, tetapi pada yang satu isinya `"Ani"` dan pada yang lain `"Budi"`. Setiap objek memiliki salinan sendiri dari setiap *field*; mengubah status Ani tidak menyentuh status Budi.

> **WAWASAN KUNCI:** Kelas ditulis sekali, oleh pemrogram, sebagai kode sumber. Objek diciptakan ketika program berjalan, sebanyak yang diperlukan — satu per pelanggan, satu per mesin — dan masing-masing membawa keadaannya sendiri. Ketika orang berkata "tulis sebuah kelas", maksudnya "rancang formulir kosongnya"; ketika mereka berkata "buat sebuah objek", maksudnya "isi satu formulir".

1.3.4 Pesan: bagaimana objek-objek bekerja sama

Kios bukan tumpukan benda; kios adalah benda-benda yang *bekerja sama*. Pelanggan menyerahkan cucian kepada kasir. Kasir mengisi nota dan menanyakan harganya kepada nota itu. Nota menghitung harganya dan menjawab. Kasir memberi tahu pelanggan, lalu memasukkan pakaian ke mesin dan menekan tombol mulai. Dalam program berorientasi objek, setiap interaksi ini adalah sebuah **pesan** (*message*): satu objek meminta objek lain melakukan salah satu perilakunya, kadang sambil menyerahkan informasi bersama permintaan itu, dan kadang menerima jawaban kembali. **Gambar 1.3** memberi nomor pada pesan-pesan di suatu pagi yang biasa.



Gambar 1.3 Suatu pagi di meja layanan sebagai percakapan antarobjek: panah utuh adalah pesan, panah putus-putus adalah jawaban, dan angkanya menunjukkan urutan kejadiannya.

Ikutilah nomor-nomornya. Pada pesan 1 pelanggan menyerahkan 3.5 kilogram. Pada pesan 2 kasir membuat pesanan baru — di Java hal itu secara harfiah ditulis `new Order(1, "Ani", 3.5, false)`. Pada pesan 3 kasir menanyakan harga kepada pesanan, dan pada jawaban 4 pesanan menjawab `24500`, yang dihitungnya dengan Persamaan (1.1) dari beratnya sendiri. Kasir meneruskannya kepada pelanggan (jawaban 5), lalu memberi tahu pesanan bahwa pencucian telah dimulai (pesan 6) dan memasukkan empat kilogram ke mesin A (pesan 7).

Dua hal patut diperhatikan. Pertama, kasir *tidak menghitung harga*: ia bertanya kepada pesanan, yang menghitungnya dari informasi yang hanya dimiliki pesanan itu, sehingga jika daftar harga berubah tahun depan, hanya metode harga pada pesanan yang berubah. Kedua, nama-nama pesan itu adalah nama metode di dalam kelas: `price()`, `startWashing()`, `load(4)`. Di Java dan Python pesan ditulis dengan titik: `ani.price()` berarti "kirimkan pesan *price* kepada objek `ani`". Istilah *pesan* berasal dari bahasa berorientasi objek awal, Smalltalk, yang perancangnya membayangkan program sebagai komunitas komputer-komputer kecil yang saling bercakap [2]; pemrogram Java dan Python lebih sering berkata "memanggil metode pada sebuah objek", yang artinya sama.

1.3.5 Abstraksi: menyimpan hanya yang penting

Nota *laundry* sungguhan mungkin bernoda kopi, bertuliskan nomor telepon di baliknya, dan terlipat di tengah. Kelas `Order` kita tidak memiliki semua itu. Ia hanya menyimpan apa yang penting bagi pekerjaan yang harus dilakukan program: menerima pesanan, menghitung harganya, dan melacaknya sampai diambil. Memutuskan apa yang disimpan dan apa yang ditinggalkan disebut **abstraksi** (*abstraction*), dan inilah keterampilan desain pertama dalam pemrograman berorientasi objek. Model yang menyimpan terlalu sedikit tidak dapat menjalankan tugasnya (pesanan tanpa berat tidak dapat dihitung harganya); model yang menyimpan terlalu banyak mahal dibangun dan membingungkan dipakai.

Abstraksi juga berlaku pada perilaku. Kasir tidak perlu tahu *bagaimana* pesanan menghitung harganya — aturan pembulatan, biaya tambahan ekspres, atau tahun depan potongan untuk anggota — cukup bahwa pesanan menjawab pesan `price()` dengan sejumlah rupiah. *Apa*-nya bersifat publik; *bagaimana*-nya urusan pesanan itu sendiri.

1.3.6 Empat gagasan besar, sekilas

Seluruh isi buku ini tumbuh dari beberapa gagasan di subbab ini. Para pemrogram biasanya meringkas pemrograman berorientasi objek sebagai empat gagasan besar. Di sini masing-masing diberikan dalam satu paragraf, dengan kios *laundry* sebagai contoh dan bab tempat gagasan itu diajarkan secara lengkap.

Enkapsulasi (*encapsulation*) berarti membungkus keadaan setiap objek bersama perilaku yang memakainya, dan menyembunyikan keadaan itu di balik perilaku tersebut, sehingga keadaan hanya dapat berubah dengan cara yang masuk akal. `Order` di Subbab 1.8 sudah melakukannya: statusnya hanya dapat bergerak maju, satu langkah demi satu langkah, dan hanya melalui metode-metodenya sendiri. Bab 3 membangun kelas dengan benar, dan Bab 7 membahas kontrol akses secara lengkap.

Abstraksi, yang baru saja Anda kenal, berarti menggambarkan *apa* yang dapat dilakukan sebuah objek tanpa mengikatkan diri pada *bagaimana*-nya. Alatnya yang paling tajam adalah **antarmuka** (*interface*): sebuah janji seperti "apa pun yang dapat dicuci memiliki metode `wash()`", yang dapat ditepati oleh mesin cuci dan oleh layanan cuci kering dengan cara yang sama sekali berbeda. Bab 6 memperkenalkan antarmuka.

Pewarisan (*inheritance*) berarti mendefinisikan kelas baru sebagai kasus khusus dari kelas yang sudah ada: `ExpressOrder` adalah sebuah `Order` dengan tenggat lebih pendek dan harga lebih tinggi, dan ia dapat memakai ulang segala hal yang sudah dikerjakan pesanan biasa. Bab 7 mengajarkannya.

Polimorfisme (*polymorphism*) — dari bahasa Yunani, "banyak bentuk" — berarti pesan yang sama dapat menghasilkan perilaku berbeda yang sesuai, bergantung pada objek yang menerimanya. Tanyakan `price()` kepada pesanan biasa dan pesanan ekspres, dan masing-masing menjawab dengan benar menurut caranya sendiri, sementara kasir mengirim pesan yang sama kepada keduanya dan tidak pernah perlu memeriksa jenis mana yang sedang dipegangnya. Bab 7 mengajarkan polimorfisme bersama pewarisan.

1.4 Dari Prosedur ke Objek

1.4.1 Program buku catatan

Untuk melihat mengapa para pemrogram beralih ke objek, mari kita tulis dahulu program *laundry* dengan cara lama, yang disebut **pemrograman prosedural** (*procedural programming*). Beginilah cara kerja buku catatan Bu Sari: satu halaman berkolom — pelanggan, kilogram, ekspres, status — dan satu baris untuk setiap pesanan. Dalam program, cara paling langsung untuk menyimpan halaman seperti itu adalah sekumpulan **larik paralel** (*parallel arrays*): satu larik untuk setiap kolom, dengan posisi 0 pada setiap larik menggambarkan pesanan pertama, posisi 1 pesanan kedua, dan seterusnya. **Larik** (*array*) adalah deretan kotak bernomor yang sejenis, dan nomor-nomornya, yang disebut **indeks**, dimulai dari 0.

Kode 1.1 adalah program itu dalam Java dan **Kode 1.2** dalam Python. Jangan mencoba memahami setiap lambang dulu — Subbab 1.7 dan 1.8 menjelaskan tata bahasanya baris demi baris. Bacalah untuk melihat *bentuknya*: empat larik paralel, sebuah fungsi terpisah yang menghitung harga, sebuah fungsi terpisah yang mencetak setiap baris, lalu dua baris yang ceroboh.

Kode 1.1 `LaundryLedger.java`: buku catatan *laundry* sebagai empat larik paralel dan dua fungsi yang berdiri sendiri, serta dua baris ceroboh yang merusaknya.

```java
public class LaundryLedger {
    public static void main(String[] args) {
        // Empat larik paralel: posisi i pada setiap larik menggambarkan pesanan i.
        String[] customer = {"Ani", "Budi", "Citra"};
        double[] kilograms = {3.5, 5.0, 2.0};
        boolean[] express = {false, true, false};
        String[] status = {"RECEIVED", "RECEIVED", "RECEIVED"};

        printAll(customer, kilograms, express, status);

        // Dua baris ceroboh, ditulis jauh di bagian lain program yang panjang:
        status[2] = "PICKED UP";   // cucian Citra bahkan belum dicuci
        kilograms[1] = -5.0;       // salah ketik: kelebihan satu tanda minus

        System.out.println("-- after two careless lines --");
        printAll(customer, kilograms, express, status);
    }

    static long price(double kg, boolean isExpress) {
        long base = Math.round(kg * 7000);          // Rp7.000 per kilogram
        return isExpress ? base + base / 2 : base;  // ekspres 50% lebih mahal
    }

    static void printAll(String[] customer, double[] kilograms,
                         boolean[] express, String[] status) {
        for (int i = 0; i < customer.length; i++) {
            System.out.println(customer[i] + ": " + kilograms[i] + " kg, Rp "
                    + price(kilograms[i], express[i]) + ", " + status[i]);
        }
    }
}
```

Kode 1.2 `laundry_ledger.py`: buku catatan yang sama sebagai empat *list* Python paralel.

```python
def price(kg, is_express):
    base = round(kg * 7000)                 # Rp7.000 per kilogram
    return base + base // 2 if is_express else base   # ekspres 50% lebih mahal


def print_all(customer, kilograms, express, status):
    for i in range(len(customer)):
        print(customer[i] + ": " + str(kilograms[i]) + " kg, Rp "
              + str(price(kilograms[i], express[i])) + ", " + status[i])


def main():
    # Empat list paralel: posisi i pada setiap list menggambarkan pesanan i.
    customer = ["Ani", "Budi", "Citra"]
    kilograms = [3.5, 5.0, 2.0]
    express = [False, True, False]
    status = ["RECEIVED", "RECEIVED", "RECEIVED"]

    print_all(customer, kilograms, express, status)

    # Dua baris ceroboh, ditulis jauh di bagian lain program yang panjang:
    status[2] = "PICKED UP"   # cucian Citra bahkan belum dicuci
    kilograms[1] = -5.0       # salah ketik: kelebihan satu tanda minus

    print("-- after two careless lines --")
    print_all(customer, kilograms, express, status)


main()
```

Kedua program mencetak hal yang persis sama:

```
Ani: 3.5 kg, Rp 24500, RECEIVED
Budi: 5.0 kg, Rp 52500, RECEIVED
Citra: 2.0 kg, Rp 14000, RECEIVED
-- after two careless lines --
Ani: 3.5 kg, Rp 24500, RECEIVED
Budi: -5.0 kg, Rp -52500, RECEIVED
Citra: 2.0 kg, Rp 14000, PICKED UP
```

1.4.2 Apa yang salah

Tiga baris keluaran pertama benar, dan Anda dapat memeriksanya dengan Persamaan (1.1) dan Persamaan (1.2): $p(3.5) = 24500$ untuk Ani, $p_{\text{ekspres}}(5) = 52500$ untuk Budi, $p(2) = 14000$ untuk Citra. Masalahnya ada pada tiga baris terakhir. Cucian Budi kini beratnya *minus* lima kilogram dan harganya *minus* Rp52.500 — kios malah berutang kepadanya. Dan pesanan Citra berstatus "PICKED UP", padahal pakaiannya belum pernah dicuci; jika kios memercayai baris itu, kantongnya akan tergeletak di rak selamanya sementara catatan menyatakan ia sudah membawanya pulang.

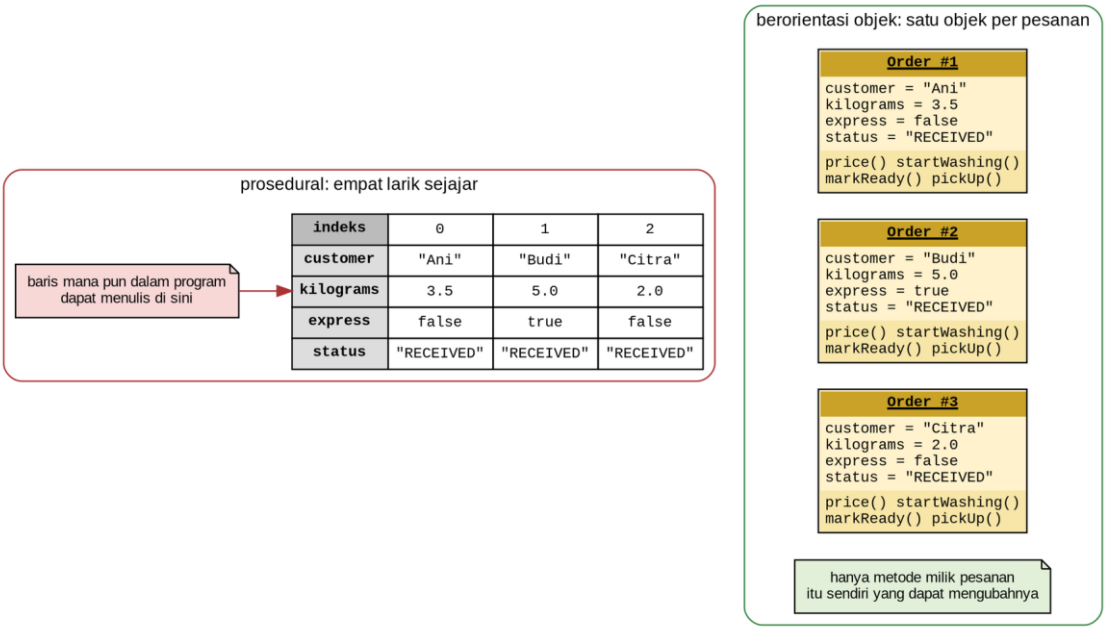
Dalam **Kode 1.1** kedua baris ceroboh itu bersebelahan di bawah sebuah komentar, sehingga mudah ditemukan. Sekarang bayangkan program itu tumbuh menjadi dua puluh ribu baris, ditulis lima orang selama tiga tahun. Baris mana pun dapat menulis ke `status` atau `kilograms`, karena larik-larik itu hanyalah data yang tergeletak terbuka, dan tidak ada satu tempat pun dalam program yang bertanggung jawab menjaga agar sebuah pesanan tetap masuk akal. Aturan "status hanya bergerak maju, satu langkah demi satu langkah" hanya hidup di kepala para pemrogram — dan manusia bisa lupa, keluar dari perusahaan, dan salah ketik pada pukul sebelas malam.

Ada masalah kedua yang lebih senyap: keempat larik harus tetap *seiring*. Tambahkan satu pesanan ke `customer` tetapi lupa menambahkannya ke `status`, maka setiap baris sesudahnya diam-diam bergeser. Dan ketika Bu Sari memutuskan untuk mencatat nomor telepon, larik kelima harus ditambahkan, dan setiap fungsi yang menerima empat larik harus diubah agar menerima lima.

> **PERINGATAN:** "Hati-hati saja" bukanlah strategi. Cara itu berhasil untuk program dua ratus baris yang ditulis satu orang dalam satu malam. Cara itu gagal, dengan pasti, untuk program apa pun yang dipakai bersama, diubah, atau disimpan lebih dari beberapa minggu — dengan kata lain, untuk hampir setiap program yang penting.

1.4.3 Jawaban objek

Jawaban berorientasi objek adalah berhenti menyebarkan sebuah pesanan ke empat larik dan menjadikan setiap pesanan satu **objek** yang menyimpan sendiri pelanggan, berat, penanda ekspres, dan statusnya — dan yang menyimpan, di tempat yang sama, satu-satunya operasi yang diizinkan mengubah semua itu. **Gambar 1.4** menempatkan kedua cara penyusunan itu berdampingan.



Gambar 1.4 Tiga pesanan yang sama disusun dengan dua cara: di kiri, empat larik paralel yang dapat ditulisi baris mana pun dalam program; di kanan, satu objek per pesanan, yang keadaannya hanya dapat diubah oleh metode-metodenya sendiri.

Di kiri, setiap kolom terbuka bagi seluruh program. Di kanan, setiap pesanan adalah kotak tertutup dengan beberapa tombol: `price()`, `startWashing()`, `markReady()`, `pickUp()`. Sebuah baris ceroboh di bagian lain program tidak dapat menulis "PICKED UP" ke pesanan Citra, karena status pesanan itu bersifat *private*; satu-satunya cara mengubahnya adalah menekan tombol, dan tombol `pickUp()` memeriksa status saat ini lebih dahulu dan *menolak* jika pakaiannya belum siap. Menambahkan nomor telepon berarti menambahkan satu *field* pada satu kelas; kode lain tidak perlu berubah kecuali ingin memakai informasi baru itu.

Tidak ada di sini yang memungkinkan program berorientasi objek menghitung sesuatu yang tidak dapat dihitung program prosedural. Orientasi objek bukanlah jenis aritmetika baru. Ia adalah cara *menata* aritmetika yang sama agar program tetap dapat dipahami dan aman ketika tumbuh. Subbab 1.8 membangun persis sisi kanan **Gambar 1.4** dalam Java dan Python, dan Anda akan menyaksikan dengan mata kepala sendiri sebuah pesanan menolak diambil sebelum waktunya.

1.5 Mengapa Java, dan Mengapa Python

Ada ratusan bahasa pemrograman, dan puluhan di antaranya mendukung objek. Buku ini memakai dua di antaranya secara berdampingan, dengan alasan-alasan yang layak dinyatakan terus terang.

1.5.1 Java: sejarah singkat

Java diciptakan di Sun Microsystems oleh sebuah tim yang dipimpin James Gosling. Bahasa ini bermula pada awal 1990-an sebagai bahasa untuk perangkat kecil seperti *set-top box* televisi, tempat sebuah program harus berjalan pada banyak cip berbeda tanpa ditulis ulang untuk masing-masing. Java dirilis untuk umum pada 1995 sebagai bagian inti *platform Java* milik Sun, tepat ketika World Wide Web mulai melesat, dan kemampuannya menjalankan program yang sama di mesin mana pun membuatnya langsung cocok untuk Internet. Sun diakuisisi Oracle pada

2010, dan sejak itu Oracle mengelola Java, bersama komunitas terbuka yang besar yang mengembangkan implementasi rujukannya, **OpenJDK**, secara terbuka [3].

Tata bahasa Java sengaja dipinjam dari C dan C++: kurung kurawal mengapit blok kode, titik koma di akhir pernyataan, `if`, `for`, dan `while` yang ditulis hampir persis seperti di C. Pemrogram C dapat membaca Java sejak hari pertama. Apa yang ditinggalkan Java juga disengaja. Java memiliki lebih sedikit fasilitas *tingkat rendah* — fitur yang bekerja dekat dengan bahasa mesin, seperti aritmetika *pointer* dan pembebasan memori secara manual — karena justru di fasilitas itulah bug paling berbahaya dalam program C dan C++ berasal. Java mengelola memori sendiri: ketika sebuah objek tidak dipakai lagi, bagian JVM yang disebut ***garbage collector*** (pengumpul sampah) mengambil kembali ruangnya secara otomatis.

Java berkembang dengan mantap. Versi baru muncul setiap enam bulan, dan setiap dua tahun salah satunya ditetapkan sebagai rilis ***long-term support*** (LTS, dukungan jangka panjang), yang terus diperbaiki para vendor dengan pembaruan keamanan dan perbaikan bug selama bertahun-tahun. Rilis LTS sejauh ini adalah Java 8, 11, 17, 21, dan 25; Java 25 dirilis pada September 2025 dan merupakan rilis LTS terkini [4]. Rilis enam-bulanan terbaru, Java 27, hadir pada September 2026 [5]; rilis itu bukan LTS, sehingga Java 25 tetap menjadi versi yang dipelajari dan dipasang di lingkungan produksi. Buku ini menyasar **Java 25**, dan setiap program di dalamnya juga dapat dikompilasi dan dijalankan pada Java 17 dan 21, sehingga instalasi yang lebih lama di laboratorium sekolah pun dapat dipakai.

1.5.2 Mengapa Java: portabel, didukung di mana-mana

Empat alasan membuat Java tetap berada di puncak dunia pemrograman selama tiga puluh tahun.

Java portabel. Program Java berjalan di setiap perangkat yang memiliki Java Virtual Machine — desktop, laptop, server, dan, melalui adaptasi bahasa ini oleh Android, ponsel pintar. Semboyannya adalah **WORA: Write Once, Run Anywhere**, dan Subbab 1.2.3 telah menunjukkan bagaimana *bytecode* mewujudkannya.

Java didukung di mana-mana. Java memiliki salah satu komunitas terbesar di antara semua bahasa: pustaka gratis untuk hampir setiap tugas, jawaban untuk hampir setiap pertanyaan, dan kelompok pengguna di sebagian besar kota besar, termasuk di Indonesia.

Java dipakai luas, terutama untuk sistem **klien–server** (*client–server*), tempat peramban atau ponsel (klien) bercakap dengan program di komputer yang jauh (server). Bank, maskapai penerbangan, layanan pemerintah, dan toko daring menjalankan sistem Java yang sangat besar.

Java membuat orientasi objek menjadi eksplisit. Setiap potong kode Java hidup di dalam kelas, setiap variabel menyatakan tipenya, dan `public` serta `private` menyatakan dengan gamblang siapa boleh memakai apa. Hal itu terasa berat dalam program sepuluh baris, tetapi bagi pembelajar ia adalah anugerah: tata bahasanya mengeja gagasan-gagasan Subbab 1.3.

1.5.3 Python: sejarah singkat, dan mengapa Python juga

Python diciptakan oleh Guido van Rossum di Belanda dan pertama kali dirilis pada 1991 [6]. Gagasan penuntunnya adalah bahwa kode jauh lebih sering dibaca daripada ditulis, sehingga kode seharusnya terbaca hampir seperti bahasa Inggris biasa. Python tidak memakai kurung kurawal: **indentasi** sebuah baris — seberapa jauh baris itu menjorok ke kanan — menunjukkan blok tempat baris itu berada. Python tidak meminta Anda menyatakan tipe: sebuah variabel dapat menyimpan nilai jenis apa pun, dan nilai itu mengetahui jenisnya sendiri. Python memeriksa tipe

ketika program berjalan, bukan sebelumnya, yang disebut **pengetikan dinamis** (*dynamic typing*); Java memeriksanya ketika program dikompilasi, yang disebut **pengetikan statis** (*static typing*).

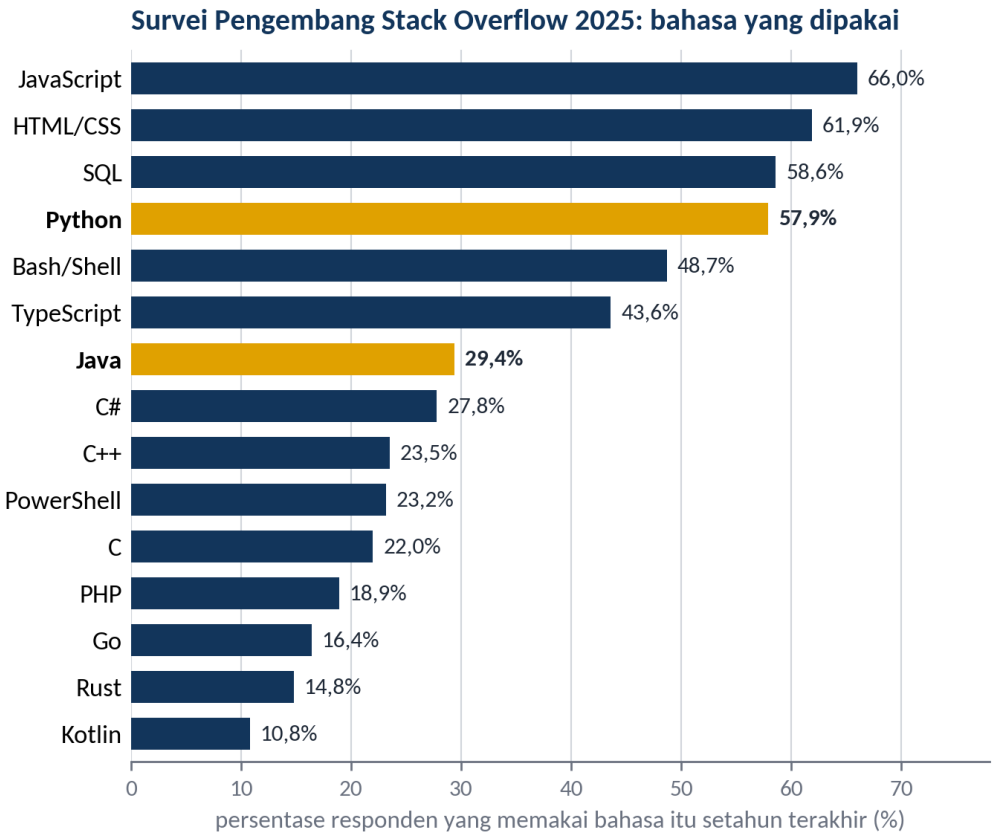
Python sepenuhnya berorientasi objek — di Python *setiap* nilai, bahkan bilangan 7, adalah objek — tetapi ia tidak memaksa Anda menulis kelas sebelum dapat mencetak satu baris. Hal itu membuatnya sangat lugas untuk program kecil dan skrip, dan itulah sebabnya Python menjadi bahasa pilihan untuk analisis data, komputasi ilmiah, otomatisasi, dan, dalam sepuluh tahun terakhir, kecerdasan artifisial dan pembelajaran mesin. Python 3.14, dirilis pada Oktober 2025, adalah versi terkini pada saat buku ini ditulis (September 2026), dengan rilis pemeliharaan ketujuhnya, 3.14.7, terbit pada Agustus 2026 [7], [8], [9]; buku ini menyasarnya, dan setiap program di dalamnya juga berjalan pada Python 3.10 dan yang lebih baru.

Alasan mengajarkan keduanya bersifat praktis. Java memperlihatkan orientasi objek dengan setiap sendinya terlihat; Python memperlihatkan gagasan yang sama tanpa perancahnya. Setelah melihat sebuah gagasan dengan dua cara, Anda memahami gagasannya, bukan sekadar ejaannya dalam satu bahasa. Dan hampir pasti Anda akan memakai keduanya dalam kehidupan kerja Anda.

1.5.4 Apa kata peringkat popularitas

“Populer” adalah kata yang licin, dan empat pengukuran terkenal masing-masing menceritakan bagian yang berbeda dari kisahnya.

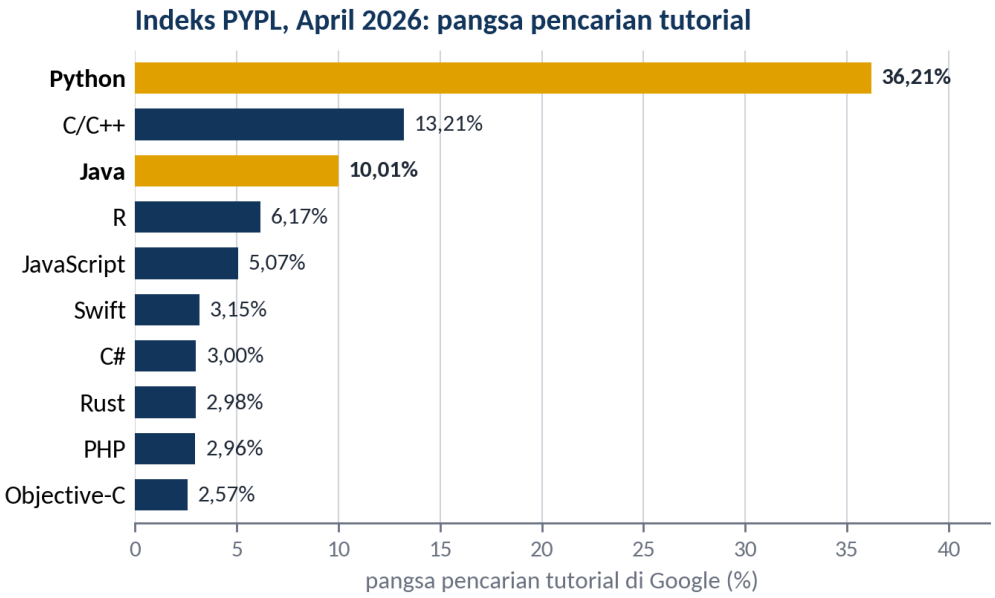
Yang pertama adalah survei. Setiap tahun Stack Overflow, situs tanya-jawab bagi para pemrogram, bertanya kepada para pengembang di seluruh dunia bahasa apa saja yang mereka pakai dalam setahun terakhir. **Gambar 1.5** memperlihatkan lima belas bahasa yang paling banyak dipakai di antara 31.771 orang yang menjawab pertanyaan itu dalam survei 2025 [10]. JavaScript memimpin dengan 66.0%, karena hampir setiap halaman web menjalankannya; bahasa *markup* dan kueri HTML/CSS serta SQL menyusul; Python berada di urutan keempat dengan 57.9%, dan Java di urutan ketujuh dengan 29.4%, di depan C#, C++, dan C.



Sumber: Stack Overflow Developer Survey 2025, seluruh responden (n = 31.771). 15 teratas.

Gambar 1.5 Bahasa yang paling banyak dipakai menurut Survei Pengembang Stack Overflow 2025, seluruh responden: Python di urutan keempat dan Java di urutan ketujuh dari lima belas bahasa yang ditampilkan.

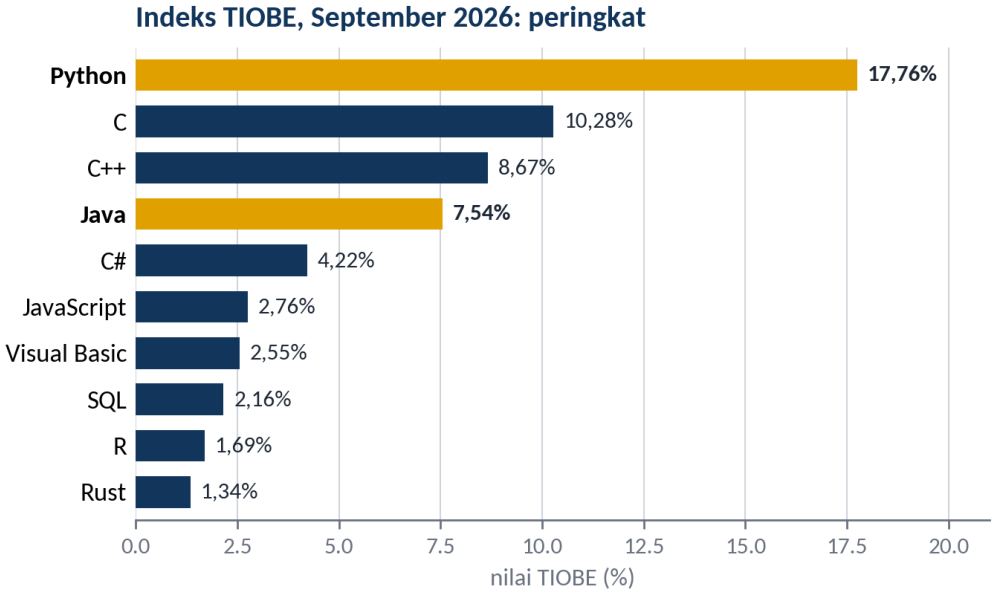
Pengukuran kedua adalah **indeks PYPL** (*PopularitY of Programming Language*). Indeks ini menghitung seberapa sering orang mencari tutorial setiap bahasa di Google, dengan data mentah dari Google Trends, berdasarkan penalaran bahwa makin sering tutorial sebuah bahasa dicari, makin banyak orang yang sedang mempelajarinya [11]. **Gambar 1.6** memperlihatkan peringkat seluruh dunianya untuk April 2026, bulan terakhir yang dapat kami verifikasi [12]: Python jauh di depan dengan 36.21% pencarian tutorial, C/C++ kedua dengan 13.21%, dan Java ketiga dengan 10.01%.



Sumber: PYPL PopularitY of Programming Language index, seluruh dunia, April 2026.

Gambar 1.6 Peringkat PYPL untuk April 2026, berdasarkan seberapa sering tutorial setiap bahasa dicari: Python pertama, Java ketiga.

Yang ketiga adalah **indeks TIOBE**, yang diperbarui setiap bulan. Penilaiannya didasarkan pada jumlah insinyur terampil di seluruh dunia, kursus, dan vendor pihak ketiga untuk setiap bahasa, yang dihimpun melalui mesin pencari dan situs populer seperti Google, Bing, Wikipedia, Amazon, YouTube, dan Baidu [13]. TIOBE sendiri menegaskan bahwa indeksnya bukan tentang bahasa pemrograman *terbaik*, dan bukan pula tentang bahasa yang paling banyak dipakai untuk menulis baris kode. **Gambar 1.7** memperlihatkan peringkatnya untuk September 2026 [14]: Python pertama dengan 17.76%, lalu C dengan 10.28%, C++ dengan 8.67%, dan Java dengan 7.54%.

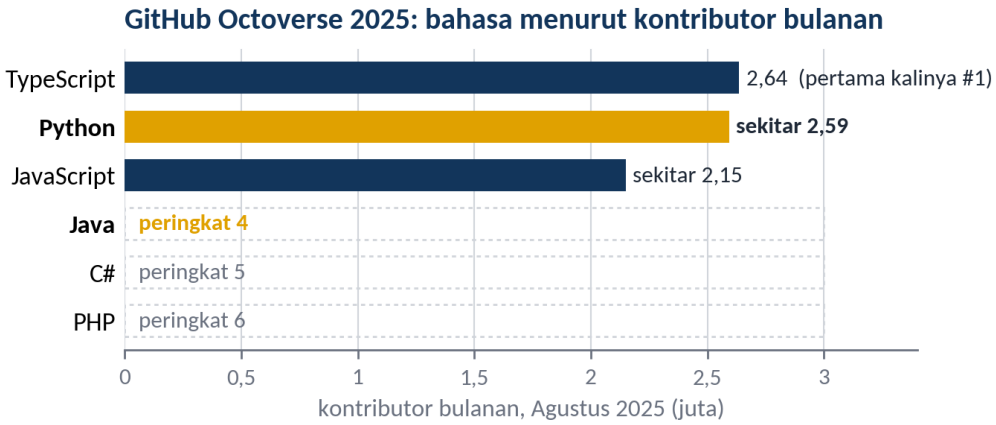


Sumber: TIOBE Index edisi September 2026.

Gambar 1.7 Peringkat TIOBE untuk September 2026, berdasarkan insinyur, kursus, dan vendor per bahasa: Python pertama, Java keempat.

Yang keempat mengukur apa yang benar-benar dibangun orang secara terbuka. **GitHub** adalah salah satu rumah kode sumber terbuka terbesar di dunia, dan setiap tahun laporan *Octoverse*-nya

menghitung, di antara banyak hal lain, berapa pengembang yang berkontribusi setiap bulan dalam setiap bahasa [15]. **Gambar 1.8** memperlihatkan peringkat 2025. Pada Agustus 2025 TypeScript, yaitu JavaScript yang diberi tipe statis, untuk pertama kalinya menjadi bahasa yang paling banyak dipakai di GitHub, dengan 2.636.006 kontributor bulanan, sekitar 42.000 lebih banyak daripada Python; JavaScript ketiga dengan sekitar 2,15 juta, dan Java keempat, disusul C# dan PHP.



Sumber: GitHub Octoverse 2025. Peringkat 4-6 ditampilkan menurut urutan saja.

Gambar 1.8 Peringkat GitHub Octoverse 2025 menurut kontributor bulanan: TypeScript, Python, dan JavaScript memimpin, dan Java di urutan keempat.

Satukan keempat gambar itu dan sebuah pola pun muncul. Pengukuran-pengukuran itu tidak sepakat tentang bahasa mana yang pertama, karena yang diukur berbeda — pemakaian di tempat kerja, pembelajaran, kehadiran di industri, karya sumber terbuka. Python berada di puncak atau dekat puncak pada setiap pengukuran itu, dan Java selalu berada di tujuh besar, bersama bahasa sistem C dan C++ serta bahasa web JavaScript dan TypeScript. Rinciannya bergeser dari bulan ke bulan, sebagaimana peringkat selalu bergeser, tetapi pola itu telah bertahan bertahun-tahun. Pemrogram yang menguasai Java dan Python siap menghadapi sebagian besar dunia perangkat lunak.

1.5.5 Kedua bahasa dibandingkan

Tabel 1.2 menyandingkan kedua bahasa. Jangan menghafalnya sekarang; kembalilah ke tabel ini setiap kali Anda menjumpai salah satu perbedaan ini dalam sebuah program.

Tabel 1.2 Java dan Python berdampingan: perbedaan-perbedaan yang pertama kali dijumpai pemula.

Pertanyaan	Java	Python
Pertama dirilis	1995, Sun Microsystems (James Gosling)	1991, Guido van Rossum
Versi yang dipakai buku ini	Java 25 LTS (juga berjalan pada 17+)	Python 3.14 (juga berjalan pada 3.10+)
Cara berjalan	dikompilasi `javac` menjadi *bytecode*, dijalankan JVM (`java`)	dijalankan interpreter (`python`), yang mengompilasi ke *bytecode* secara internal
Tipe	statis: dinyatakan, diperiksa sebelum berjalan	dinamis: diperiksa ketika berjalan
Blok kode	kurung kurawal `{ }`	indentasi
Akhir pernyataan	titik koma `;`	akhir baris
Tempat kode berada	setiap metode di dalam kelas	kelas opsional; fungsi dan skrip biasa pun boleh
Keadaan *private*	dipaksakan dengan kata kunci `private`	sebuah konvensi: garis bawah di depan, `_status`
Mencetak satu baris	`System.out.println("Hi");`	`print("Hi")`
Pemakaian khas	sistem klien–server dan perusahaan berskala besar, Android	analisis data, AI, skrip, *back end* web

1.5.6 Satu cara berpikir, dua ejaan

Baris terpenting **Tabel 1.2** adalah baris yang tidak ada di sana: *gagasannya* sama. Kelas di Java dan kelas di Python sama-sama formulir kosong; objek dalam kedua bahasa memiliki keadaan, perilaku, dan identitas; pengiriman pesan ditulis dengan titik dalam keduanya. Ketika Anda mempelajari gagasan baru dalam buku ini, pelajarilah dahulu sebagai gagasan, lalu bagaimana Java mengejanya, lalu bagaimana Python mengejanya. Dengan begitu, ketika Anda bertemu bahasa berorientasi objek ketiga — C#, Kotlin, Swift, TypeScript — Anda akan mendapati bahwa sebagian besarnya sudah Anda ketahui.

1.6 Perkakas Kerja

Sebelum memasang apa pun, mari kita pahami apa yang akan kita pasang dan mengapa. Tukang kayu yang tahu fungsi setiap perkakas di kotaknya tidak akan panik ketika salah satunya hilang.

1.6.1 JDK dan JRE

Untuk *menulis* program Java Anda memerlukan **Java Development Kit** (JDK). Nama resmi edisi sehari-harinya adalah *Java SE Development Kit*, dengan SE berarti *Standard Edition*. JDK berisi kompilator `javac`, peluncur `java`, pustaka kelas (ribuan kelas siap pakai untuk teks, berkas, jaringan, jendela, dan banyak lagi), serta sejumlah perkakas lain. **Tabel 1.3** mendaftar perkakas yang dipakai buku ini.

Tabel 1.3 Perkakas Java Development Kit yang dipakai buku ini.

Perkakas	Fungsinya	Tempat buku ini memakainya
`javac`	mengompilasi berkas sumber `.java` menjadi *bytecode* `.class`	sejak Subbab 1.7
`java`	menyalakan JVM dan menjalankan program; juga menjalankan satu berkas sumber secara langsung	sejak Subbab 1.7
`jshell`	jendela interaktif tempat Anda mengetik Java satu baris demi satu baris	Bab 3
`javadoc`	membangun dokumentasi berupa halaman web dari komentar dalam kode sumber	Bab 6
`jar`	mengemas banyak berkas `.class` menjadi satu arsip yang dapat dibagikan	Bab 2

Untuk sekadar *menjalankan* program Java, dahulu cukup sebuah paket yang lebih kecil: **Java Runtime Environment** (JRE), berisi JVM dan pustaka kelas tetapi tanpa kompilator. Selama bertahun-tahun keduanya dipasang terpisah; sejak Java 11 JDK sudah memuat *runtime* lengkapnya sendiri, sehingga untuk buku ini Anda cukup memasang JDK, tidak yang lain.

Selain JDK milik Oracle sendiri, beberapa organisasi membangun JDK gratis yang teruji baik dari kode sumber OpenJDK yang sama. Buku ini menganjurkan **Eclipse Temurin**, dari proyek Adoptium milik Eclipse Foundation [16]: gratis untuk keperluan apa pun, tersedia untuk Windows, macOS, dan Linux, serta diperbarui secara berkala.

1.6.2 Di dalam JDK: folder `bin` dan PATH

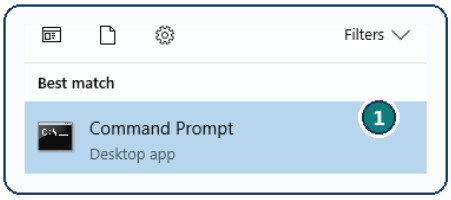
JDK adalah sebuah folder, dan mengenal susunannya akan menghemat berjam-jam kebingungan di kemudian hari. Segala sesuatu yang Anda jalankan ada di subfoldernya, `bin` (singkatan dari *binaries*, yaitu program yang dapat dijalankan). **Gambar 1.9** memperlihatkan isi folder `bin` sebuah JDK di komputer Linux: `java` menjalankan program, `javac` mengompilasinya, `javadoc` membuat dokumentasi, `jar` mengemas arsip, dan `jshell` adalah jendela interaktif pada **Tabel 1.3**; kelimanya disorot dalam daftar itu, sedangkan program lainnya adalah perkakas khusus yang untuk sementara boleh Anda abaikan. JDK pada gambar itu versi 21, tetapi folder `bin` JDK 25 berisi program yang pada dasarnya sama, dan di Windows setiap program itu berakhiran `.exe`: `java.exe`, `javac.exe`, `javadoc.exe`, dan seterusnya. Untuk contoh-contoh berikut, andaikan di sebuah komputer Windows JDK dipasang di folder `D:\Program\Java\JDK`, sehingga program-programnya berada di `D:\Program\Java\JDK\bin`.



Gambar 1.9 Folder `bin` sebuah JDK, ditampilkan di terminal Linux; kelima perkakas pada **Tabel 1.3** disorot, dan di Windows setiap berkasnya berakhiran `.exe`.

Terminal — di Windows *Command Prompt* atau *PowerShell*, di macOS dan Linux *Terminal* — adalah jendela tempat Anda mengetik perintah alih-alih mengeklik. Di Windows Anda dapat

membuka Command Prompt dengan mengeklik tombol Search di bilah tugas dan mengetik `cmd`; Windows menyarankan *Command Prompt* sebagai kecocokan terbaik, seperti pada **Gambar 1.10**, lalu Anda mengekliknya.



Gambar 1.10 Membuka Command Prompt di Windows: setelah mengetik `cmd` di kotak pencarian bilah tugas, pilih kecocokan terbaik (1).

Dari terminal Anda dapat melakukan segalanya: menyunting program Java dengan penyunting teks polos apa pun (Notepad di Windows, `gedit` atau `vim` di Linux), mengompilasinya dengan `javac`, dan menjalankannya dengan `java`. Tetapi ketika Anda mengetik `javac`, bagaimana terminal tahu di mana `javac.exe` berada? Terminal menelusuri daftar folder yang tersimpan dalam sebuah pengaturan bernama **PATH**. Jika folder `bin` milik JDK tidak ada di PATH, terminal menjawab bahwa ia tidak pernah mendengar tentang `javac`. Agar praktis, supaya Anda dapat memanggil `javac.exe` dan `java.exe` dari folder mana pun, tambahkan folder `bin` itu ke PATH. Cara tercepat, yang hanya berlaku selama jendela terminal itu terbuka, adalah perintah pada **Gambar 1.11**.



Gambar 1.11 Menambahkan folder `bin` milik JDK ke bagian depan PATH untuk jendela Command Prompt saat ini dengan perintah `set path=`.

Dua catatan tentang perintah itu. Pertama, akhiran `;%path%` penting: `%path%` mewakili daftar yang lama, sehingga perintah itu menaruh folder `bin` milik JDK di depannya dan mempertahankan semua yang lain. Tanpa akhiran itu, `set path=D:\Program\Java\JDK\bin` akan *mengganti* seluruh PATH jendela itu dengan satu folder saja, dan banyak perintah lain berhenti bekerja di jendela itu. Kedua, perubahan itu hilang ketika jendela ditutup. Penginstal modern, seperti penginstal Temurin di Subbab 1.7, mengatur PATH secara permanen untuk Anda. Tetapi setiap kali terminal berkata tidak mengenali `java` atau `javac`, kini Anda tahu apa yang salah: folder `bin` tidak ada di PATH jendela itu.

Pengaturan yang berkaitan, **JAVA_HOME**, menyebut folder JDK itu sendiri (bukan `bin`-nya); beberapa perkakas, seperti sistem *build*, membacanya untuk menemukan Java. Di macOS dan Linux, PATH dipisahkan dengan titik dua, bukan titik koma.

1.6.3 Interpreter Python dan kawan-kawannya

Python lebih mudah dipasang karena ia memiliki satu program pusat: **interpreter**, bernama `python` (di Windows juga `py`, dan di macOS serta Linux biasanya `python3`). Distribusi resmi dari python.org berisi interpreter, **pustaka standar** (*standard library*) yang besar berisi modul siap pakai, sebuah penyunting kecil bernama **IDLE**, dan **pip**, perkakas yang mengunduh pustaka tambahan dari Python Package Index.

Satu gagasan Python lagi akan memudahkan hidup Anda ketika proyek-proyek Anda tumbuh: **lingkungan virtual** (*virtual environment*). Lingkungan virtual adalah folder pribadi, biasanya bernama `.venv`, yang menyimpan salinan pengaturan interpreter dan kumpulan pustaka tambahannya sendiri untuk satu proyek, sehingga dua proyek yang memerlukan versi pustaka berbeda tidak saling bertengkar. Program-program buku ini hanya memakai pustaka standar dan tidak memerlukannya, tetapi Subbab 1.7.6 menunjukkan cara membuatnya, karena VS Code akan menawarkannya.

1.6.4 Lingkungan pengembangan terpadu

Anda *dapat* menulis program di Notepad dan menjalankannya dari terminal, dan setiap pemrogram sebaiknya melakukannya paling tidak sekali untuk memahami apa yang terjadi di baliknya. Tetapi para profesional bekerja di dalam **lingkungan pengembangan terpadu** (*integrated development environment*, IDE): satu aplikasi yang menggabungkan penyunting yang memahami bahasanya, kompilator atau interpreter, tempat melihat keluaran, dan ***debugger*** (pelacak kesalahan) yang memungkinkan Anda menjalankan program satu baris demi satu baris sambil mengamati datanya berubah. **Tabel 1.4** membandingkan empat IDE yang paling mungkin Anda jumpai.

Tabel 1.4 Empat lingkungan pengembangan untuk Java dan Python.

IDE	Pembuat	Bahasa	Catatan
Eclipse	Eclipse Foundation, www.eclipse.org	terutama Java; banyak bahasa lain melalui *plug-in*	gratis; sebagian besar ditulis dalam Java; ruang kerja dasar ditambah sistem *plug-in* yang dapat diperluas
NetBeans	Apache Software Foundation, netbeans.apache.org	Java; juga PHP, C, C++, HTML5, JavaScript	gratis; dibangun dari modul; berjalan di Windows, macOS, Linux, dan Solaris
IntelliJ IDEA	JetBrains	Java dan Kotlin (PyCharm untuk Python)	edisi Community gratis; perkakas *refactoring* yang kuat
Visual Studio Code	Microsoft	hampir semua, melalui ekstensi	gratis; ringan; sama di Windows, macOS, dan Linux

Eclipse sebagian besar ditulis dalam Java dan terutama dipakai untuk Java [17]. Ia terdiri atas ruang kerja dasar ditambah sistem ***plug-in*** — paket tambahan yang mengajarinya keterampilan baru — yang melaluinya ia mendukung puluhan bahasa lain, dari C, C++, dan PHP (dengan perkakas CDT dan PDT-nya) hingga Python, Rust, dan Scala, bahkan dokumen LaTeX. **NetBeans** [18] membangun aplikasi dari komponen-komponen modular yang disebut **modul**, berjalan di Windows, macOS, Linux, dan Solaris, dan dapat diperluas sendiri oleh pengembang lain.

Buku ini memakai **Visual Studio Code** (VS Code) sebagai meja kerja utamanya: gratis dan sama di ketiga sistem operasi, menangani Java dan Python sama baiknya dalam satu jendela, dan dapat berjalan di laptop sederhana. Bab 2 memasang Eclipse di sampingnya dan menjelajahi *debugger* dengan saksama. Jika laboratorium Anda memakai Eclipse, NetBeans, atau IntelliJ IDEA, setiap program dalam buku ini berjalan di sana tanpa perubahan, karena semuanya berkas `.java` dan `.py` biasa.

1.7 Menyiapkan VS Code untuk Java dan Python di Windows, macOS, dan Linux

Setiap program dalam buku ini dimaksudkan untuk Anda ketik, jalankan, dan ubah sendiri. Sebelum subbab berikutnya meminta Anda melakukannya, mari kita bangun meja kerjanya sekali, dengan cermat, di komputer mana pun yang Anda miliki. Anda memerlukan empat hal:

1. **Java Development Kit**, versi 25 LTS. Kita memakai **Eclipse Temurin 25** [16]. JDK versi 17 atau yang lebih baru dapat menjalankan program-program buku ini, tetapi pasanglah versi 25 bila memungkinkan.
2. **Interpreter Python**, versi 3.14, dari python.org atau manajer paket sistem Anda [8]. Versi apa pun mulai 3.10 dapat dipakai.
3. **Visual Studio Code**, penyuntingnya, dari code.visualstudio.com.
4. **Dua ekstensi VS Code**: **Extension Pack for Java** dan ekstensi **Python**, keduanya diterbitkan Microsoft [19], [20].

Pasanglah perkakas bahasa lebih dahulu, periksa di terminal, lalu pasang penyuntingnya, lalu hubungkan keduanya. Memeriksa di terminal *sebelum* membuka VS Code itu penting: jika ada yang salah dengan JDK atau Python, terminal memberi pesan yang singkat dan jelas, sedangkan penyunting mungkin memberi pesan yang membingungkan. **Tabel 1.5** menghimpun seluruh prosedur untuk ketiga sistem operasi dalam satu halaman, agar Anda dapat kembali kepadanya kelak; sub-subbab sesudahnya menjelaskan setiap langkah.

Tabel 1.5 Meja kerja di tiga sistem operasi, dari komputer kosong hingga program berjalan dalam kedua bahasa. Di Fedora gunakan `sudo dnf install java-25-openjdk-devel python3.14`; di Arch, `sudo pacman -S jdk-openjdk python`.

Langkah	Windows 10/11	MacOS	Linux (Ubuntu/Debian)
1. Pasang JDK	Penginstal `.msi` Temurin 25 dari adoptium.net, atau `winget install EclipseAdoptium.Temurin.25.JDK`	Penginstal `.pkg` Temurin 25 dari adoptium.net, atau `brew install --cask temurin@25`	Tambahkan repositori Adoptium, lalu `sudo apt install temurin-25-jdk`
2. Pasang Python	Penginstal dari python.org (centang **Add python.exe to PATH**), atau `winget install Python.Python.3.14`	Penginstal dari python.org, atau `brew install python@3.14`	`sudo apt install python3.14 python3.14-venv` (dari arsip *deadsnakes* di Ubuntu)
3. Periksa Java	`java -version` dan `javac -version`	sama	sama
4. Periksa Python	`python --version` (atau `py --version`)	`python3 --version`	`python3.14 --version`
5. Pasang VS Code	Penginstal dari code.visualstudio.com; centang **Add to PATH**	Unduh, seret *Visual Studio Code* ke *Applications*	`.deb` dari code.visualstudio.com, atau `sudo snap install code --classic`
6. Tambahkan ekstensi	Tampilan Extensions (`Ctrl+Shift+X`): **Extension Pack for Java** dan **Python**	sama, dengan `Cmd+Shift+X`	sama, dengan `Ctrl+Shift+X`
7. Jalankan berkas Java	`javac Hello.java` lalu `java Hello`	sama	sama
8. Jalankan berkas Python	`python hello.py`	`python3 hello.py`	`python3 hello.py`

Satu catatan tentang nomor versi sebelum Anda mulai. Java 25 dan Python 3.14 menerima *rilis pembaruan* secara berkala yang memperbaiki bug dan masalah keamanan: Java 25.0.1, 25.0.2, dan seterusnya, Python 3.14.1, 3.14.2, dan seterusnya; pada September 2026 yang terbaru adalah Temurin 25.0.4 dan Python 3.14.7 [21], [9]. Selalu ambil pembaruan terbaru dari versi yang disebut buku ini; angka sesudah titik kedua di layar Anda akan lebih besar daripada di contoh-contoh di sini, dan memang begitulah seharusnya.

1.7.1 Windows, langkah demi langkah

Pasang JDK.

1. Buka peramban web dan kunjungi **adoptium.net**. Halaman depannya menawarkan rilis LTS Temurin terbaru untuk sistem Anda; pastikan tertulis **25** dan **Windows x64** (atau **aarch64** pada laptop ARM), lalu unduh penginstal **.msi** untuk **JDK**.
2. Jalankan penginstal itu. Pada halaman bernama *Custom Setup*, pastikan **Add to PATH** dan **Set JAVA_HOME variable** keduanya akan dipasang — klik masing-masing dan pilih *Will be installed on local hard drive*. Kedua pengaturan ini melakukan secara permanen apa yang dilakukan **Gambar 1.11** untuk satu jendela. Terima sisanya dan klik **Install**.
3. Jika Anda lebih suka mengetik daripada mengeklik, Windows 10 dan 11 menyertakan manajer paket bernama **winget**. Buka PowerShell dan ketik `winget install EclipseAdoptium.Temurin.25.JDK`; perintah itu mengunduh dan menjalankan penginstal yang sama.

Pasang Python.

4. Kunjungi **python.org**, buka **Downloads → Windows**, dan unduh Python **3.14**. Halaman unduhan menawarkan *Python install manager*, yang memasang Python beserta perintah `py` yang dapat mengelola beberapa versi berdampingan, serta penginstal klasik. Keduanya baik. Dengan penginstal klasik, **centang kotak "Add python.exe to PATH"** di halaman pertamanya sebelum mengeklik *Install Now* — inilah satu hal yang paling sering terlewat oleh pemula. Dengan *install manager*, ikuti petunjuknya, lalu ketik `py install 3.14` di terminal baru jika Anda diminta memilih versi.
5. Sebagai alternatif, di PowerShell: `winget install Python.Python.3.14`.

Periksa keduanya.

6. Tutup setiap jendela terminal yang terbuka, lalu buka jendela PowerShell atau Command Prompt yang *baru*. Jendela lama masih memakai PATH lama dan tidak akan menemukan perkakas baru. Ketik ketiga perintah ini, tekan Enter setelah masing-masing:

```
java -version
javac -version
python --version
```

Anda akan melihat sesuatu yang mirip dengan berikut. Perintah pertama mencetak tiga baris tentang *runtime*, perintah kedua mencetak versi kompilator, dan perintah ketiga mencetak versi Python.

```
openjdk version "25.0.1" 2025-10-21 LTS
OpenJDK Runtime Environment Temurin-25.0.1+8 (build 25.0.1+8-LTS)
OpenJDK 64-Bit Server VM Temurin-25.0.1+8 (build 25.0.1+8-LTS, mixed mode, sharing)
javac 25.0.1
Python 3.14.0
```

Jika `python --version` malah membuka Microsoft Store, atau tidak mencetak apa pun, berarti Windows menjalankan program pengganti kecil yang disebut *app execution alias* alih-alih Python yang sebenarnya. Cobalah `py --version`; jika berhasil, gunakan `py` di setiap tempat buku ini menulis `python`. Agar `python` sendiri berfungsi, buka **Settings → Apps → Advanced app settings → App execution aliases** dan matikan dua entri bernama *python.exe* dan *python3.exe*, lalu buka terminal baru.

1.7.2 macOS, langkah demi langkah

Pasang JDK.

1. Kunjungi **adoptium.net**, pilih **Temurin 25**, **macOS**, dan prosesor yang tepat: **aarch64** untuk Mac dengan Apple silicon (M1, M2, M3, M4, dan yang lebih baru), **x64** untuk Mac Intel yang lebih lama. Jika ragu, buka menu Apple → **About This Mac**: di sana tertulis *Chip Apple M…* atau *Processor Intel…*. Unduh penginstal **.pkg** dan jalankan. Penginstal itu memasang JDK ke `/Library/Java/JavaVirtualMachines/`, tempat macOS menemukannya secara otomatis; tidak ada PATH yang perlu diatur.
2. Jika Anda memakai manajer paket **Homebrew**, `brew install --cask temurin@25` melakukan hal yang sama.

Pasang Python.

3. macOS mungkin sudah memiliki perintah `python3`, tetapi sering kali itu versi lama milik perkakas pengembang sistem; jangan mengandalkannya. Kunjungi **python.org → Downloads → macOS**, unduh penginstal **Python 3.14** (paket *universal2* yang berfungsi pada kedua jenis Mac), lalu jalankan. Di akhir, penginstal membuka sebuah folder; klik dua kali **Install Certificates.command** di dalamnya, agar Python dapat membuat koneksi web yang aman.

4. Dengan Homebrew: `brew install python@3.14`.

Periksa keduanya. Buka **Terminal** (tekan `Cmd+Space`, ketik *Terminal*, tekan Enter) dan ketik:

```
java -version
javac -version
python3 --version
```

Di macOS, ketik `python3`, bukan `python`: nama `python` tidak ada atau menunjuk ke tempat yang tidak membantu. Di setiap tempat buku ini menulis `python`, pengguna Mac mengetik `python3`.

1.7.3 Linux, langkah demi langkah

Distribusi Linux memasang perangkat lunak dari **repositori** melalui manajer paket. Perintah-perintah di bawah ini untuk Ubuntu dan Debian; **Tabel 1.5** memberikan padanannya untuk Fedora dan Arch.

Pasang JDK. Distribusi Anda mungkin sudah menawarkan paket bernama `openjdk-25-jdk`; cobalah `sudo apt install openjdk-25-jdk` lebih dahulu. Jika tidak ditemukan, tambahkan repositori Adoptium, yang selalu memuat rilis Temurin terkini. Ketik baris-baris ini satu per satu; baris-baris yang panjang adalah satu baris:

```
sudo apt install -y wget apt-transport-https gpg
wget -qO - https://packages.adoptium.net/artifactory/api/gpg/key/public | gpg --dearmor |
sudo tee /etc/apt/trusted.gpg.d/adoptium.gpg > /dev/null
echo "deb https://packages.adoptium.net/artifactory/deb $(awk -F=
'/^VERSION_CODENAME/{print$2}' /etc/os-release) main" | sudo tee
/etc/apt/sources.list.d/adoptium.list
sudo apt update
sudo apt install temurin-25-jdk
```

Baris pertama memasang tiga program pembantu kecil; baris kedua mengunduh kunci yang dipakai Adoptium untuk menandatangani paket-paketnya, agar sistem Anda dapat memastikan keasliannya; baris ketiga memberi tahu manajer paket di mana repositori itu berada; dua baris terakhir menyegarkan daftar paket dan memasang JDK. Ketik kata sandi Anda ketika `sudo` memintanya — tidak ada yang muncul di layar saat Anda mengetik, dan itu normal.

Pasang Python. Ubuntu menyertakan Python 3 sebagai bagian dari sistem, tetapi biasanya bukan versi terbaru, dan Anda tidak boleh menghapus atau mengganti Python milik sistem, karena sistem operasi itu sendiri memakainya. Pasanglah 3.14 *di samping*-nya. Di Ubuntu sumber yang lazim adalah arsip *deadsnakes*:

```
sudo add-apt-repository ppa:deadsnakes/ppa
sudo apt update
sudo apt install python3.14 python3.14-venv
```

Setelah itu `python3.14` menjalankan versi baru dan `python3` tetap menjalankan versi sistem. Jika distribusi Anda sudah memiliki Python 3.10 atau yang lebih baru sebagai `python3` — periksa dengan `python3 --version` — Anda cukup memakainya untuk buku ini.

Periksa keduanya.

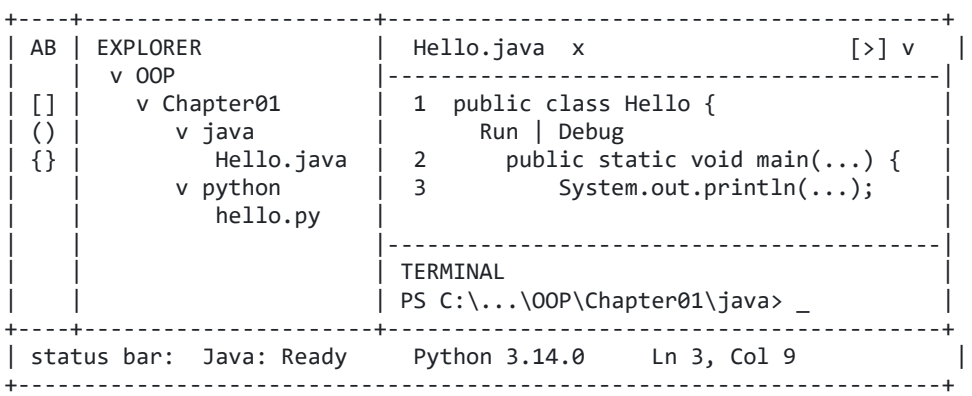
```
java -version
javac -version
python3.14 --version
```

Jika `java -version` melaporkan Java lama yang sudah terpasang sebelumnya, pilih yang baru dengan `sudo update-alternatives --config java`, lalu lakukan lagi untuk `javac`.

1.7.4 VS Code dan kedua ekstensinya

1. **Pasang VS Code.** Unduh dari **code.visualstudio.com**. Di Windows, jalankan penginstalnya dan, pada halaman *Select Additional Tasks*, centang **Add to PATH** dan kedua opsi **Open with Code**; opsi-opsi itu memungkinkan Anda mengetik `code .` di terminal, atau mengeklik kanan sebuah folder di Explorer, untuk membuka folder itu di VS Code. Di macOS, buka arsip yang diunduh dan seret *Visual Studio Code* ke *Applications*; lalu jalankan VS Code, tekan `Cmd+Shift+P` untuk membuka **Command Palette**, ketik *shell command*, dan pilih **Shell Command: Install 'code' command in PATH**. Di Linux, pasang paket `.deb` dengan `sudo apt install ./code_*.deb` dari folder unduhan, atau gunakan `sudo snap install code --classic`.
2. **Pasang ekstensi Java.** Jalankan VS Code dan buka tampilan **Extensions**: klik ikon empat kotak kecil di tepi kiri, atau tekan `Ctrl+Shift+X` (`Cmd+Shift+X` di Mac). Cari **Extension Pack for Java** dan pasang yang diterbitkan Microsoft. Ini adalah paket yang memasang beberapa ekstensi sekaligus, di antaranya *Language Support for Java by Red Hat* (yang memahami kode Java, mewarnainya, melengkapinya, dan menggarisbawahi kesalahan saat Anda mengetik), *Debugger for Java*, *Test Runner for Java* (dipakai di Bab 6), *Maven for Java*, dan *Project Manager for Java* [19].
3. **Pasang ekstensi Python.** Di tampilan yang sama, cari **Python** dan pasang yang diterbitkan Microsoft. Ekstensi ini juga membawa **Pylance**, yang memahami kode Python, dan **Python Debugger** [20].
4. **Beri tahu VS Code Java dan Python mana yang dipakai.** Biasanya ia menemukannya sendiri. Untuk memeriksanya, buka Command Palette (`Ctrl+Shift+P` atau `Cmd+Shift+P`). Untuk Java, jalankan **Java: Configure Java Runtime**: halaman yang terbuka mendaftar JDK yang ditemukan VS Code; pastikan yang dipakai proyek Anda adalah versi 25. Untuk Python, jalankan **Python: Select Interpreter** dan pilih Python 3.14 dari daftarnya. Interpreter yang terpilih ditampilkan di bilah status kanan bawah setiap kali sebuah berkas Python terbuka.
5. **Buat rumah untuk buku ini.** Buat sebuah folder, misalnya `OOP` di dalam folder Documents, dengan satu subfolder per bab dan di dalam masing-masing folder `java` dan `python`: `OOP/Chapter01/java`, `OOP/Chapter01/python`. Persis beginilah folder kode pendamping `Code_new` disusun. Di VS Code pilih **File → Open Folder** dan buka `OOP`. Jika VS Code bertanya apakah Anda memercayai berkas-berkas di folder ini, jawablah ya: itu folder Anda sendiri.

Jendela VS Code memiliki lima area yang akan terus Anda pakai. Berikut sketsanya, agar nama-nama pada langkah-langkah berikut bermakna (VS Code menampilkan labelnya dalam bahasa Inggris secara bawaan):



Activity bar (bilah aktivitas, AB) di paling kiri berpindah-pindah antartampilan: Explorer (berkas-berkas Anda), Search, Source Control, Run and Debug, dan Extensions. **Side bar** (bilah samping) di sebelahnya menampilkan tampilan yang aktif — di sini Explorer dengan pohon foldernya. **Editor** di tengah memuat berkas-berkas yang terbuka, dengan tombol **Run** (segitiga)

di kanan atasnya. **Panel** di bawah editor memuat **Terminal**, terminal sungguhan yang sudah terbuka di folder Anda; bukalah dengan **Terminal → New Terminal**. **Status bar** (bilah status) di bagian bawah memberi tahu Anda, antara lain, apakah dukungan Java sudah selesai dinyalakan ("Java: Ready") dan interpreter Python mana yang terpilih.

1.7.5 Program Java pertama Anda

Di Explorer, klik kanan folder `Chapter01/java`, pilih **New File**, dan beri nama `Hello.java` — dengan H kapital, karena di Java nama berkas harus sama persis dengan nama kelas, termasuk huruf kapitalnya. Ketik **Kode 1.3** dan simpan dengan `Ctrl+S` (`Cmd+S`).

Kode 1.3 `Hello.java`: program Java pertama, yang menyapa pembaca dan melaporkan versi Java yang menjalankannya.

```
public class Hello {
    public static void main(String[] args) {
        System.out.println("Hello from Profitina Laundry!");
        System.out.println("Java is running on version " +
System.getProperty("java.version") + ".");
    }
}
```

Ada tiga cara menjalankannya, dan Anda sebaiknya mencoba ketiganya sekali.

Dengan tombol Run. Di atas baris `public static void main`, VS Code menampilkan dua tautan kecil, **Run | Debug**. Klik **Run**. (Atau klik segitiga di kanan atas editor.) VS Code mengompilasi berkas itu dan menjalankannya, dan keluarannya muncul di panel Terminal.

Dengan dua perintah di terminal, persis seperti yang digambarkan **Gambar 1.1**. Buka terminal (**Terminal → New Terminal**), masuk ke folder dengan `cd Chapter01/java` (di Windows `cd Chapter01\java`), lalu ketik:

```
javac Hello.java
java Hello
```

Perintah pertama mengompilasi `Hello.java` menjadi `Hello.class` dan tidak mencetak apa pun jika semuanya beres — dalam pemrograman, diam biasanya berarti berhasil. Perintah kedua menjalankan kelas itu. Perhatikan bahwa Anda menulis `java Hello`, tanpa `.class`: Anda menyebut kelas yang akan dijalankan, bukan berkasnya.

Dengan satu perintah. Sejak Java 11, peluncur dapat mengompilasi satu berkas sumber di memori dan langsung menjalankannya [22]: `java Hello.java`. Cara ini praktis untuk program kecil; untuk program yang terdiri atas beberapa berkas, seperti program `Order` di Subbab 1.8, gunakan `javac` lebih dahulu.

Cara mana pun yang Anda pilih, program itu mencetak:

```
Hello from Profitina Laundry!
Java is running on version 21.0.10.
```

Baris kedua memberi tahu Java mana yang sebenarnya menjalankan program. Listing-listing dalam buku ini diperiksa dengan Java 21 pada mesin yang menghasilkan buku ini, itulah sebabnya tertulis `21.0.10`; di komputer Anda, jika Anda mengikuti Subbab 1.7.1, 1.7.2, atau 1.7.3, akan tertulis `25.0.` diikuti nomor pembaruannya. Jika yang tampil angka yang lebih lama, VS Code atau terminal memakai JDK lain daripada yang Anda pasang; Subbab 1.7.8 menjelaskan cara memperbaikinya.

Sekarang mari kita baca **Kode 1.3** baris demi baris.

- `public class Hello {` mendeklarasikan kelas bernama `Hello`. Di Java setiap potong kode hidup di dalam kelas, bahkan program yang belum ada hubungannya dengan objek. Kata `public` berarti kode lain mana pun boleh memakai kelas ini. Kurung kurawal buka `{` memulai badan kelas, yang berakhir pada `}` pasangannya di baris terakhir.
- `public static void main(String[] args) {` mendeklarasikan **metode main**, tempat setiap program Java dimulai. Setiap katanya punya tugas. `public`: peluncur Java, yang berada di luar kelas, harus diizinkan memanggilnya. `static`: metode ini milik kelas itu sendiri, sehingga dapat berjalan sebelum ada objek apa pun dari kelas itu. `void`: metode ini tidak mengembalikan jawaban. `main`: nama yang dicari peluncur. `String[] args`: larik *string* berisi kata-kata apa pun yang diketik sesudah nama kelas di baris perintah; kita belum memakainya, tetapi ia harus ada.
- `System.out.println("Hello from Profitina Laundry!");` mencetak satu baris. `System` adalah kelas dari pustaka Java; `out` adalah objek di dalamnya yang mewakili keluaran terminal; `println` ("*print line*", cetak baris) adalah pesan yang dikirim kepada objek itu, memintanya mencetak teks di dalam tanda kutip lalu pindah ke baris baru. Titik koma mengakhiri pernyataan.
- Pernyataan berikutnya menyusun barisnya dengan `+`, yang di antara *string* berarti *menyambungkan*: teks `"Java is running on version "`, lalu nilai yang dikembalikan `System.getProperty("java.version")` — pesan yang menanyakan versi Java kepada sistem yang sedang berjalan — lalu sebuah titik.

> **WAWASAN KUNCI:** Bahkan program pertama pun penuh dengan objek dan pesan: `System.out` adalah objek, dan `println(...)` adalah pesan yang dikirim kepadanya dengan titik. Anda sudah melakukan pemrograman berorientasi objek sejak baris Java pertama Anda.

Java 25 menambahkan cara yang lebih ringan untuk menulis program pertama, yang disebut *compact source file* (berkas sumber ringkas): sebuah berkas boleh berisi hanya `void main() { IO.println("Hello"); }` tanpa kelas yang membungkusnya [23]. Ini kebaruan yang menyenangkan bagi pemula, tetapi tidak ada di Java 17 atau 21, sehingga buku ini selalu menulis bentuk lengkap **Kode 1.3**, yang berfungsi di setiap versi.

1.7.6 Program Python pertama Anda

Di Explorer, klik kanan `Chapter01/python`, pilih **New File**, beri nama `hello.py` (nama berkas Python lazimnya huruf kecil semua), lalu ketik **Kode 1.4**.

Kode 1.4 `hello.py`: program Python pertama, yang melakukan hal yang sama dengan **Kode 1.3**.

```
import platform

print("Hello from Profitina Laundry!")
print("Python is running on version " + platform.python_version() + ".")
```

Jalankan dengan salah satu dari dua cara.

Dengan tombol Run. Klik segitiga di kanan atas editor (keterangannya berbunyi **Run Python File**). VS Code menjalankan berkas itu dengan interpreter yang tampil di bilah status, dan keluarannya muncul di panel Terminal.

Di terminal. Masuk ke folder itu (`cd Chapter01/python`) dan ketik `python hello.py` di Windows, `python3 hello.py` di macOS dan Linux (atau `python3.14 hello.py` jika Anda memasang 3.14 di samping Python sistem yang lebih lama). Tidak ada langkah kompilasi terpisah.

Program itu mencetak:

```
Hello from Profitina Laundry!
Python is running on version 3.11.15.
```

Sekali lagi baris kedua jujur tentang interpreter yang menjalankan program: mesin buku ini memiliki Python 3.11, dan milik Anda akan menampilkan `3.14.` dan nomor pembaruannya.

Baris demi baris:

- `import platform` membuat modul `platform` dari pustaka standar tersedia. **Modul** adalah berkas berisi kode Python siap pakai; `import` membawanya ke dalam program Anda. Java tidak memerlukan baris seperti itu karena `System` selalu tersedia.
- Baris kosong hanya untuk kenyamanan mata pembaca.
- `print("Hello from Profitina Laundry!")` mencetak satu baris. `print` adalah fungsi bawaan; tidak ada kelas yang perlu ditulis, tidak ada `main`, tidak ada titik koma. Seluruh berkas adalah programnya, dijalankan dari atas ke bawah.
- `print("Python is running on version " + platform.python_version() + ".")` menyambungkan tiga *string* dengan `+`, sama seperti Java. `platform.python_version()` menanyakan versi yang sedang berjalan kepada modul itu.

Letakkan kedua listing itu berdampingan dan Anda memperoleh seluruh **Tabel 1.2** dalam bentuk mini: Java membungkus programnya dalam sebuah kelas dan metode `main` serta mengakhiri pernyataan dengan titik koma; Python menjalankan berkas dari atas; keduanya menyambungkan teks dengan `+`; keduanya memakai titik untuk menjangkau isi sesuatu (`System.out`, `platform.python_version`).

Lingkungan virtual, jika VS Code menawarkannya. Ketika Anda membuka folder Python, VS Code mungkin menyarankan pembuatan lingkungan virtual. Untuk buku ini hal itu opsional. Jika Anda menginginkannya, buka Command Palette dan jalankan **Python: Create Environment**, pilih **Venv** dan Python 3.14 Anda; VS Code membuat folder `.venv` dan memilihnya. Padanannya di terminal adalah `python -m venv .venv` (atau `python3 -m venv .venv`).

1.7.7 *Debugging*: mengamati program berpikir

Ketika sebuah program memberi jawaban yang salah, cara tercepat untuk mengetahui sebabnya adalah *mengamatinya* berjalan, satu baris demi satu baris, sambil melihat datanya berubah. Perkakas yang memungkinkan hal itu adalah ***debugger***. Mari kita coba pada program yang memuat perulangan, agar ada sesuatu yang dapat diamati. Setiap mesin Profitina Laundry menampung $c = 8$ kilogram; maka sebuah pesanan seberat w kilogram memerlukan

$$L(w) = \left\lceil \frac{w}{c} \right\rceil \tag{1.3}$$

muatan, dengan $\lceil x \rceil$, "*ceiling* dari x", adalah bilangan bulat terkecil yang tidak kurang dari x: pesanan 9 kilogram tidak muat dalam satu mesin, sehingga memerlukan $\lceil 9/8 \rceil = \lceil 1.125 \rceil = 2$ muatan. Komputer membagi bilangan bulat dengan membuang sisanya, yang merupakan *floor*, bukan *ceiling*; kiat baku untuk memperoleh *ceiling* dari pembagian bilangan bulat adalah

$$\left\lceil \frac{w}{c} \right\rceil = \left\lfloor \frac{w + c - 1}{c} \right\rfloor \quad \text{untuk bilangan bulat } w \geq 0,\ c \geq 1. \tag{1.4}$$

Mengapa Persamaan (1.4) benar? Tulislah $w = qc + r$ dengan sisa $0 \leq r \leq c - 1$. Jika $r = 0$, maka $w/c = q$ tepat, dan $(w + c - 1)/c = q + (c - 1)/c$, yang *floor*-nya q karena $(c - 1)/c < 1$. Jika $r \geq 1$, *ceiling*-nya $q + 1$, dan $(w + c - 1)/c = q + (r + c - 1)/c$, dengan $c \leq r + c - 1 \leq 2c - 2 <$

$2c$, sehingga *floor*-nya $q + 1$. Kedua kasus sepakat. Periksa dengan angka: $w = 9$, $c = 8$ memberi $\lfloor 16/8 \rfloor = 2$; $w = 8$ memberi $\lfloor 15/8 \rfloor = 1$; $w = 3$ memberi $\lfloor 10/8 \rfloor = 1$.

Kode 1.5 dan **Kode 1.6** menerapkan Persamaan (1.4) pada setiap pesanan dari lima pesanan dan menjumlahkan muatannya.

Kode 1.5 `Loads.java`: menghitung muatan mesin untuk pesanan-pesanan hari ini, program kecil dengan perulangan untuk diamati di *debugger*.

```
public class Loads {
    public static void main(String[] args) {
        int capacity = 8;                  // kilogram yang dicuci satu mesin sekaligus
        int[] orders = {3, 9, 5, 12, 7};   // kilogram tiap pesanan hari ini
        int totalLoads = 0;

        for (int i = 0; i < orders.length; i++) {
            // ceiling bilangan bulat: berapa muatan penuh atau sebagian yang diperlukan
pesanan ini
            int loads = (orders[i] + capacity - 1) / capacity;
            totalLoads = totalLoads + loads;
            System.out.println("order " + (i + 1) + ": " + orders[i]
                    + " kg -> " + loads + " load(s)");
        }
        System.out.println("machine loads today: " + totalLoads);
    }
}
```

Kode 1.6 `loads.py`: perhitungan yang sama dalam Python; `//` adalah pembagian bilangan bulat Python.

```
capacity = 8                  # kilogram yang dicuci satu mesin sekaligus
orders = [3, 9, 5, 12, 7]     # kilogram tiap pesanan hari ini
total_loads = 0

for i in range(len(orders)):
    # ceiling bilangan bulat: berapa muatan penuh atau sebagian yang diperlukan pesanan ini
    loads = (orders[i] + capacity - 1) // capacity
    total_loads = total_loads + loads
    print("order " + str(i + 1) + ": " + str(orders[i])
          + " kg -> " + str(loads) + " load(s)")
print("machine loads today: " + str(total_loads))
```

Keduanya mencetak:

```
order 1: 3 kg -> 1 load(s)
order 2: 9 kg -> 2 load(s)
order 3: 5 kg -> 1 load(s)
order 4: 12 kg -> 2 load(s)
order 5: 7 kg -> 1 load(s)
machine loads today: 7
```

Periksa dengan tangan memakai Persamaan (1.3): $\lceil 3/8 \rceil = 1$, $\lceil 9/8 \rceil = 2$, $\lceil 5/8 \rceil = 1$, $\lceil 12/8 \rceil = 2$, $\lceil 7/8 \rceil = 1$, dan $1 + 2 + 1 + 2 + 1 = 7$.

Sekarang *debug* program itu. Langkah-langkahnya sama dalam kedua bahasa:

1. **Pasang *breakpoint*.** Di `Loads.java`, arahkan tetikus ke sebelah kiri nomor baris pada baris `totalLoads = totalLoads + loads;` lalu klik. Sebuah titik merah muncul: ***breakpoint*** (titik henti), tempat *debugger* akan menghentikan sementara program.
2. **Jalankan *debugger*.** Klik **Debug** di atas `main` (Java), atau tekan `F5`. Untuk Python, buka `loads.py`, pasang *breakpoint* pada `total_loads = total_loads + loads`, tekan `F5`, dan pilih

Python Debugger: Current File jika VS Code bertanya. Program mulai dan berhenti di titik merah, dengan baris itu disorot kuning. Baris yang disorot *belum* dijalankan.

3. **Lihat panel Variables** di sebelah kiri. Panel itu mendaftar setiap variabel beserta nilainya saat ini: `capacity = 8`, `orders` (klik panahnya untuk melihat kelima angkanya), `totalLoads = 0`, `i = 0`, `loads = 1`.
4. **Melangkah.** Tekan `F10` (**Step Over**) untuk menjalankan baris yang disorot dan berhenti di baris berikutnya. Amati `totalLoads` berubah dari `0` menjadi `1`. Tekan `F5` (**Continue**) untuk berjalan sampai *breakpoint* tercapai lagi, pada putaran berikutnya: kini `i = 1`, `loads = 2`, `totalLoads = 1`. Lanjutkan lagi, dan lagi, dan amati jumlahnya bertambah: 1,3,4,6,7.
5. **Amati sebuah ekspresi.** Di panel **Watch**, klik `+` dan ketik `orders[i] / capacity`; ekspresi yang sama berlaku dalam kedua bahasa. Di Java, membagi dua nilai `int` menghasilkan bilangan bulat, sehingga untuk `i = 1` panel itu menampilkan `1`; di Python, `/` selalu menghasilkan pecahan dan menampilkan `1.125`. Perbedaan tunggal itulah sebabnya program memakai `(orders[i] + capacity - 1) / capacity` di Java dan `//` di Python.
6. **Berhenti.** Tekan `Shift+F5`, atau klik kotak merah pada bilah alat *debugging* kecil di bagian atas jendela.

Tombol-tombol lain pada bilah alat itu layak diketahui sejak awal: `F11` (**Step Into**) masuk *ke dalam* pemanggilan metode pada baris saat ini, yang akan terus Anda pakai begitu program tersusun dari banyak metode; `Shift+F11` (**Step Out**) berjalan sampai akhir metode saat ini dan kembali ke pemanggilnya. Bab 2 dikhususkan untuk *debugging* dan memakai semuanya.

1.7.8 Ketika ada yang tidak beres

Setiap masalah pemasangan yang dijumpai pemula sudah dijumpai ribuan orang sebelumnya. **Tabel 1.6** mendaftar masalah yang umum, tampilannya, dan cara memperbaikinya.

Tabel 1.6 Masalah umum saat menyiapkan Java, Python, dan VS Code, serta cara memperbaikinya.

Yang Anda lihat	Artinya	Yang harus dilakukan
`'java' is not recognized as an internal or external command` (Windows), atau `java: command not found`	folder `bin` milik JDK tidak ada di PATH terminal ini	buka terminal *baru*; jika masih terjadi, pasang ulang dengan **Add to PATH** dicentang, atau tambahkan folder `bin` secara manual (Subbab 1.6.2)
`java -version` menampilkan 25 tetapi `javac -version` tidak ditemukan	sebuah *runtime* tanpa kompilator berada paling depan di PATH	pasang **JDK**, bukan hanya JRE; letakkan `bin` milik JDK paling depan di PATH
`java -version` menampilkan versi lama seperti 1.8	Java yang lebih lama berada lebih depan di PATH	hapus yang lama, atau pindahkan folder `bin` yang baru ke urutan teratas PATH; di Linux gunakan `update-alternatives`
`error: class Hello is public, should be declared in a file named Hello.java`	nama berkas dan nama kelas berbeda, mungkin hanya pada huruf kapital	ganti nama berkas agar sama persis dengan kelasnya: `Hello.java` untuk `class Hello`
`Error: Could not find or load main class Hello.class`	Anda mengetik `java Hello.class`	ketik `java Hello`, menyebut kelasnya, bukan berkasnya
`python` membuka Microsoft Store, atau tidak mencetak apa pun (Windows)	sebuah *app execution alias* menggantikan Python	gunakan `py`, atau matikan alias itu (Subbab 1.7.1) dan buka terminal baru
`python: command not found` (macOS, Linux)	di sistem ini perintahnya `python3`	ketik `python3` (atau `python3.14`)
`IndentationError: unexpected indent`	sebuah baris Python menjorok padahal tidak seharusnya, atau tab dan spasi tercampur	ratakan blok itu dengan spasi saja; VS Code menyisipkan empat spasi per `Tab` secara bawaan
VS Code menjalankan Python 3.11 (atau versi lain) alih-alih 3.14	interpreter lain yang terpilih	**Python: Select Interpreter** lalu pilih 3.14; periksa bilah status
Kode Java di VS Code tidak berwarna dan tanpa pelengkapan, bilah status menampilkan "Java" yang berputar	*language server* Java masih dinyalakan, atau tidak ditemukan JDK 17+ untuknya	tunggu semenit pada penyalaan pertama; jalankan **Java: Configure Java Runtime** dan pastikan sebuah JDK terdaftar
Jendela program berkedip lalu tertutup	Anda mengeklik dua kali sebuah berkas alih-alih menjalankannya dari VS Code atau terminal	jalankan dengan tombol Run atau dari terminal

Dengan meja kerja yang sudah berdiri, setiap listing di sisa buku ini tinggal satu salinan dan satu kali jalan untuk menjadi milik Anda.

1.8 Kelas Pertama Anda: Sebuah Pesanan *Laundry*

Meja kerja sudah siap dan kedua bahasa sudah menyapa. Sekarang kita membangun sisi kanan **Gambar 1.4**: kelas `Order` yang membungkus keadaan sebuah pesanan bersama perilakunya dan melindungi yang satu dengan yang lain. Kita mendekatinya dalam tiga langkah. Pertama program sederhana dengan variabel, untuk melihat bagaimana nilai diberi nama dan dihitung. Lalu kelasnya. Lalu program kecil yang membuat pesanan dan mengirimi mereka pesan.

1.8.1 Sebuah struk: variabel dan aritmetika

Variabel adalah kotak bernama yang menyimpan sebuah nilai. **Kode 1.7** dan **Kode 1.8** menghitung struk Ani dengan beberapa variabel.

Kode 1.7 `Receipt.java`: struk yang dihitung dengan variabel-variabel dari empat tipe.

```java
public class Receipt {
    public static void main(String[] args) {
        String customer = "Ani";
        double kilograms = 3.5;
        long pricePerKg = 7000;
        long price = Math.round(kilograms * pricePerKg);
        long paid = 50000;
        long change = paid - price;

        System.out.println("PROFITINA LAUNDRY - RECEIPT");
        System.out.println("Customer : " + customer);
        System.out.println("Weight   : " + kilograms + " kg");
        System.out.println("Price    : Rp " + price);
        System.out.println("Paid     : Rp " + paid);
        System.out.println("Change   : Rp " + change);
    }
}
```

Kode 1.8 `receipt.py`: struk yang sama dalam Python, tempat nilai-nilai menentukan tipenya sendiri.

```python
customer = "Ani"
kilograms = 3.5
price_per_kg = 7000
price = round(kilograms * price_per_kg)
paid = 50000
change = paid - price

print("PROFITINA LAUNDRY - RECEIPT")
print("Customer : " + customer)
print("Weight   : " + str(kilograms) + " kg")
print("Price    : Rp " + str(price))
print("Paid     : Rp " + str(paid))
print("Change   : Rp " + str(change))
```

Keduanya mencetak:

```
PROFITINA LAUNDRY - RECEIPT
Customer : Ani
Weight   : 3.5 kg
Price    : Rp 24500
Paid     : Rp 50000
Change   : Rp 25500
```

Di Java, setiap variabel **dideklarasikan** beserta tipenya saat pertama kali muncul. `String customer = "Ani";` membuat kotak bernama `customer` yang hanya boleh menyimpan teks, lalu menaruh `"Ani"` di dalamnya. `double kilograms = 3.5;` membuat kotak untuk bilangan berpecahan. `long pricePerKg = 7000;` membuat kotak untuk bilangan bulat; Java memiliki dua tipe bilangan bulat yang umum, `int`, yang menampung nilai hingga sekitar 2.1 miliar, dan `long`, yang menampung nilai hingga sekitar 9.2×10^{18}, dan kita memakai `long` untuk uang agar jumlah yang besar tidak pernah meluap. `Math.round(kilograms * pricePerKg)` mengalikan lalu membulatkan ke bilangan bulat terdekat, persis seperti dinyatakan Persamaan (1.1); `Math` adalah kelas pustaka yang menyimpan fungsi-fungsi matematika.

Di Python tidak ada kata tipe: `kilograms = 3.5` membuat variabel yang kebetulan menyimpan bilangan berpecahan, dan `price_per_kg = 7000` variabel yang kebetulan menyimpan bilangan bulat. Bilangan bulat Python sama sekali tidak memiliki batas atas. Fungsi `round` membulatkan, dan `str(...)` mengubah bilangan menjadi teks agar dapat disambungkan dengan teks lain memakai `+`; di Java, menyambungkan bilangan dengan *string* mengubahnya secara otomatis, tetapi Python mengharuskan Anda memintanya. Perhatikan juga gaya penamaannya: Java menulis nama bersuku kata banyak dalam **camelCase** (`pricePerKg`), Python dalam **snake_case** (`price_per_kg`). Setiap komunitas mengikuti konvensinya sendiri, dan buku ini mengikuti keduanya.

Kembaliannya $50000 - 24500 = 25500$, dan program pun sepakat. Namun sejauh ini belum ada satu pun objek buatan kita sendiri. Struk itu hanyalah lima variabel lepas. Mari kita bungkus mereka.

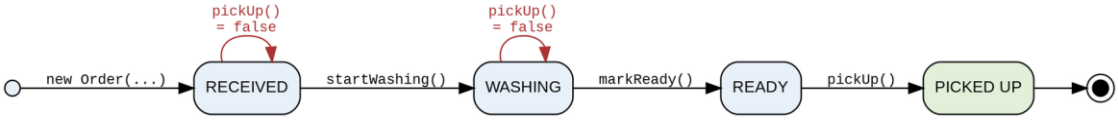
1.8.2 Merancang kelas `Order`

Sebelum menulis kode, kita merancang, memakai pertanyaan-pertanyaan Subbab 1.3.

Apa yang diketahui sebuah pesanan? **Nomornya**, nama **pelanggan**, **kilogram**-nya, apakah ia **ekspres**, dan **status**-nya. Empat yang pertama tetap sejak nota ditulis; status berubah ketika cucian bergerak melewati kios.

Apa yang dapat dilakukan sebuah pesanan? Memberi tahu **harganya**, dihitung dengan Persamaan (1.1) dan Persamaan (1.2) dari berat dan penanda ekspresnya sendiri. Memajukan statusnya: **mulai dicuci**, **tandai siap**, **diambil**. Melaporkan **statusnya**. Dan menggambarkan dirinya dalam satu baris, untuk dicetak.

Apa yang tidak boleh pernah terjadi? Status hanya boleh bergerak maju, satu langkah demi satu langkah, dengan urutan RECEIVED, WASHING, READY, PICKED UP. Khususnya, sebuah pesanan tidak boleh pernah diambil sebelum siap. **Gambar 1.12** menggambar aturan-aturan ini sebagai **diagram keadaan** (*state diagram*): setiap kotak bersudut bulat adalah sebuah keadaan, dan setiap panah adalah pesan yang memindahkan pesanan dari satu keadaan ke keadaan berikutnya.



Gambar 1.12 Riwayat hidup sebuah pesanan: setiap pesan memajukannya satu langkah, dan pesan yang dikirim pada saat yang salah, seperti `pickUp()` ketika cucian masih dicuci, ditolak dan tidak mengubah apa pun.

Lingkaran merah pada **Gambar 1.12** sama pentingnya dengan panah-panah hitam. Lingkaran itu menyatakan bahwa `pickUp()` yang dikirim kepada pesanan berstatus RECEIVED atau WASHING mengembalikan `false` dan membiarkan statusnya persis seperti semula. Hal yang sama berlaku untuk setiap pesan lain yang dikirim pada saat yang salah; gambar itu hanya menggambarkan yang paling menggoda. Pesanan yang *menolak* hal yang tidak masuk akal adalah inti seluruh subbab ini.

1.8.3 Kelas `Order` dalam Java

Kode 1.9 adalah kelasnya. Inilah program terpanjang sejauh ini, jadi kita membacanya perlahan sesudahnya.

Kode 1.9 `Order.java`: pesanan *laundry* yang menyimpan keadaannya sendiri, menghitung harganya sendiri, dan memajukan statusnya hanya bila hal itu masuk akal.

```
public class Order {
    // Keadaan: apa yang diketahui setiap pesanan tentang dirinya.
    private final int number;
    private final String customer;
    private final double kilograms;
    private final boolean express;
    private String status;

    // Konstruktor: berjalan sekali, saat pesanan baru diciptakan dengan "new".
    public Order(int number, String customer, double kilograms, boolean express) {
        this.number = number;
        this.customer = customer;
        this.kilograms = kilograms;
        this.express = express;
        this.status = "RECEIVED";
    }

    // Perilaku: apa yang dapat dilakukan setiap pesanan saat menerima pesan.
    public long price() {
        long base = Math.round(kilograms * 7000);   // Rp7.000 per kilogram
        if (express) {
            return base + base / 2;                 // ekspres 50% lebih mahal
        }
        return base;
    }

    public boolean startWashing() {
        if (!status.equals("RECEIVED")) {
            return false;                           // tolak: saatnya tidak tepat
        }
        status = "WASHING";
        return true;
    }

    public boolean markReady() {
        if (!status.equals("WASHING")) {
            return false;
        }
        status = "READY";
        return true;
    }
```

Kode 1.9 (Lanjutan) *Order.java: pesanan laundry yang menyimpan keadaannya sendiri, menghitung harganya sendiri, dan memajukan statusnya hanya bila hal itu masuk akal.*

```
    public boolean pickUp() {
        if (!status.equals("READY")) {
            return false;                          // cucian basah tidak boleh diambil
        }
        status = "PICKED UP";
        return true;
    }

    public String getStatus() {
        return status;
    }

    @Override
    public String toString() {
        String kind = express ? " express" : "";
        return "Order #" + number + " for " + customer + ": " + kilograms + " kg"
                + kind + ", Rp " + price() + ", " + status;
    }
}
```

Berkas ini tidak memiliki metode `main`, sehingga tidak dapat dijalankan sendirian: ia formulir kosong, bukan program. Bagian-bagiannya, dari atas ke bawah:

Field. Kelima baris yang diawali `private` mendeklarasikan keadaan yang akan dibawa setiap objek `Order`, sesuai dengan kotak kelas pada **Gambar 1.2**. `private` berarti tidak ada kode di luar kelas `Order` yang boleh membaca atau menulis *field* ini secara langsung: baris seperti `ani.status = "PICKED UP";` yang ditulis di kelas lain ditolak kompilator dengan pesan *status has private access in Order*. Satu kata itu mengubah aturan "hanya pesanan yang mengubah statusnya" dari sebuah harapan menjadi sebuah jaminan. Kata `final` pada empat *field* pertama berarti bahwa begitu *field* itu diberi nilai, di konstruktor, nilainya tidak pernah dapat berubah. `status` tidak `final`, karena ia harus berubah.

Konstruktor. `public Order(int number, String customer, double kilograms, boolean express)` tampak seperti metode tetapi tidak memiliki tipe kembalian dan bernama sama dengan kelasnya. Konstruktor berjalan tepat sekali untuk setiap objek baru, pada saat objek itu diciptakan, dan tugasnya mengisi formulir kosong. Keempat nilai di dalam tanda kurung adalah **parameter**-nya: informasi yang diberikan pemanggil. Di dalamnya, `this.number = number;` menyalin parameter `number` ke *field* `number` milik objek yang sedang diciptakan. Kata `this` berarti *objek ini sendiri*, dan diperlukan di sini karena parameter dan *field* bernama sama; `this.number` adalah *field*-nya, `number` saja adalah parameternya. Baris terakhir mengatur status menjadi `"RECEIVED"`, karena setiap pesanan bermula di sana — pemanggil tidak dapat memilih untuk menciptakan pesanan yang sudah diambil.

Metode `price`. `public long price()` tidak menerima parameter, karena semua yang diperlukannya sudah ada di dalam objek, dan mengembalikan sebuah `long`. Metode ini menghitung `base` dengan Persamaan (1.1). Pernyataan `if (express)` menambahkan separuh `base` bila penanda ekspres bernilai benar; `base / 2` membagi dua bilangan bulat, yang di Java membuang sisanya — persis *floor* pada Persamaan (1.2). Kata kunci `return` mengakhiri metode dan menyerahkan nilainya kembali kepada siapa pun yang mengirim pesan.

Ketiga metode status. `startWashing()`, `markReady()`, dan `pickUp()` semuanya mengikuti pola yang sama: *periksa, lalu bertindak*. `pickUp()`, misalnya, lebih dahulu bertanya `!status.equals("READY")` — "apakah statusnya *bukan* READY?" (`!` berarti *tidak*). Jika ya, pesanan menolak: `return false;` langsung mengakhiri metode, dan status tidak tersentuh. Jika

tidak, metode itu mengatur status menjadi `"PICKED UP"` dan mengembalikan `true`, agar pemanggil tahu bahwa permintaannya berhasil. Perhatikan bahwa *string* dibandingkan dengan `equals`, bukan dengan `==`; Subbab 1.11 menjelaskan mengapa hal itu penting di Java.

`getStatus()` menyerahkan status untuk dibaca. Pihak luar boleh *melihat* status melalui metode ini, tetapi tidak ada metode yang mengizinkan mereka *mengaturnya* menjadi nilai sembarang. Metode seperti ini, yang hanya melaporkan sebuah *field*, disebut ***getter***.

`toString()` mengembalikan deskripsi satu baris tentang pesanan. Metode ini istimewa di Java: setiap kali sebuah objek dicetak dengan `System.out.println` atau disambungkan ke *string* dengan `+`, Java secara otomatis memanggil metode `toString()`-nya untuk mengetahui apa yang ditampilkan. Baris `@Override` di atasnya memberi tahu kompilator bahwa kita dengan sengaja mengganti `toString()` yang sudah dimiliki setiap objek Java — bawaan yang membosankan, yang mencetak sesuatu seperti `Order@1b6d3586`. Di dalamnya, `express ? " express" : ""` adalah **operator kondisional**: ia memberi `" express"` jika `express` bernilai benar dan *string* kosong jika tidak.

1.8.4 Kelas `Order` dalam Python

Kode 1.10 adalah kelas yang sama dalam Python.

Kode 1.10 `order.py`: kelas `Order` yang sama dalam Python.

```
class Order:
    # Konstruktor: berjalan sekali, saat pesanan baru diciptakan dengan Order(...).
    def __init__(self, number, customer, kilograms, express):
        # Keadaan: apa yang diketahui setiap pesanan tentang dirinya.
        self._number = number
        self._customer = customer
        self._kilograms = kilograms
        self._express = express
        self._status = "RECEIVED"

    # Perilaku: apa yang dapat dilakukan setiap pesanan saat menerima pesan.
    def price(self):
        base = round(self._kilograms * 7000)   # Rp7.000 per kilogram
        if self._express:
            return base + base // 2            # ekspres 50% lebih mahal
        return base

    def start_washing(self):
        if self._status != "RECEIVED":
            return False                       # tolak: saatnya tidak tepat
        self._status = "WASHING"
        return True

    def mark_ready(self):
        if self._status != "WASHING":
            return False
        self._status = "READY"
        return True

    def pick_up(self):
        if self._status != "READY":
            return False                       # cucian basah tidak boleh diambil
        self._status = "PICKED UP"
        return True

    def get_status(self):
        return self._status

    def __str__(self):
        kind = " express" if self._express else ""
        return ("Order #" + str(self._number) + " for " + self._customer + ": "
                + str(self._kilograms) + " kg" + kind + ", Rp "
                + str(self.price()) + ", " + self._status)
```

Gagasannya identik; ejaannya berbeda di lima tempat.

1. **Baris kelas** adalah `class Order:`, dengan titik dua, dan segala sesuatu yang menjorok di bawahnya milik kelas itu. Tidak ada kurung kurawal di mana pun.
2. **Konstruktor** adalah metode dengan nama ganjil `__init__` (dua garis bawah di depan dan di belakang; diucapkan "*dunder init*"). Python memanggilnya secara otomatis ketika sebuah objek diciptakan. Parameter pertamanya, `self`, adalah objek yang sedang diciptakan — gagasan yang sama dengan `this` di Java, tetapi di Python Anda harus menuliskannya secara eksplisit sebagai parameter pertama setiap metode.
3. ***Field* tidak dideklarasikan lebih dahulu.** *Field* muncul ketika konstruktor pertama kali memberinya nilai: `self._status = "RECEIVED"` menciptakan *field* `_status` pada objek baru itu.
4. **Privasi adalah konvensi.** Python tidak memiliki kata kunci `private`. Garis bawah di depan, seperti pada `_status`, adalah tanda yang disepakati komunitas yang berarti "ini milik objek; jangan disentuh dari luar". Python tidak akan menghentikan pemrogram yang bertekad menulis `ani._status = "PICKED UP"`, tetapi setiap pemrogram Python yang membaca baris itu tahu bahwa ia melanggar aturan, dan perkakas seperti Pylance di VS Code menandainya. Bab 7 menunjukkan pilihan-pilihan Python yang lebih kuat.

5. **Pencetakan memakai `__str__`.** Metode yang dipanggil Python ketika sebuah objek dicetak adalah `__str__`, padanan `toString()` di Java. Dan ekspresi kondisionalnya ditulis dengan kata-kata: `" express" if self._express else ""`.

Dua perbedaan kecil lain layak diperhatikan. Pembagian bilangan bulat Python adalah `//`, sehingga `base // 2` langsung merupakan *floor* pada Persamaan (1.2), dan metode-metodenya dinamai dengan snake_case: `start_washing`, `mark_ready`, `pick_up`, `get_status`. Selain itu Python membandingkan *string* langsung dengan `!=` dan `==`, yang di Python membandingkan *isi string* — aman di sini, tidak seperti `==` di Java.

1.8.5 Menghidupkan pesanan-pesanan

Sebuah kelas tidak melakukan apa pun sampai seseorang menciptakan objek darinya dan mengirimi objek-objek itu pesan. **Kode 1.11** dan **Kode 1.12** adalah program yang melakukannya.

Kode 1.11 `OrderDemo.java`: menciptakan dua pesanan dan mengirimi mereka pesan, termasuk pengambilan yang terlalu dini dan ditolak.

```java
public class OrderDemo {
    public static void main(String[] args) {
        Order ani = new Order(1, "Ani", 3.5, false);
        Order budi = new Order(2, "Budi", 5.0, true);
        System.out.println(ani);
        System.out.println(budi);

        System.out.println("Ani tries to pick up at once: " + ani.pickUp());
        ani.startWashing();
        ani.markReady();
        System.out.println("After washing and drying:     " + ani.getStatus());
        System.out.println("Ani picks up now: " + ani.pickUp());

        System.out.println(ani);
        System.out.println(budi);
    }
}
```

Kode 1.12 `order_demo.py`: program yang sama dalam Python, yang mengimpor kelasnya dari `order.py`.

```python
from order import Order

ani = Order(1, "Ani", 3.5, False)
budi = Order(2, "Budi", 5.0, True)
print(ani)
print(budi)

print("Ani tries to pick up at once: " + str(ani.pick_up()))
ani.start_washing()
ani.mark_ready()
print("After washing and drying:     " + ani.get_status())
print("Ani picks up now: " + str(ani.pick_up()))

print(ani)
print(budi)
```

Untuk menjalankan versi Java dari terminal, kedua berkas harus dikompilasi, karena `OrderDemo` memakai `Order`:

```
javac Order.java OrderDemo.java
java OrderDemo
```

(`javac *.java` mengompilasi setiap berkas Java di folder itu sekaligus. Di VS Code, cukup klik **Run** di atas `main` dalam `OrderDemo.java`; VS Code mengompilasi semua yang diperlukan

program.) Untuk Python, `order.py` dan `order_demo.py` harus berada di folder yang sama; lalu `python order_demo.py` menjalankannya (`python3` di macOS dan Linux). Versi Java mencetak:

```
Order #1 for Ani: 3.5 kg, Rp 24500, RECEIVED
Order #2 for Budi: 5.0 kg express, Rp 52500, RECEIVED
Ani tries to pick up at once: false
After washing and drying:     READY
Ani picks up now: true
Order #1 for Ani: 3.5 kg, Rp 24500, PICKED UP
Order #2 for Budi: 5.0 kg express, Rp 52500, RECEIVED
```

dan versi Python mencetak:

```
Order #1 for Ani: 3.5 kg, Rp 24500, RECEIVED
Order #2 for Budi: 5.0 kg express, Rp 52500, RECEIVED
Ani tries to pick up at once: False
After washing and drying:     READY
Ani picks up now: True
Order #1 for Ani: 3.5 kg, Rp 24500, PICKED UP
Order #2 for Budi: 5.0 kg express, Rp 52500, RECEIVED
```

Kedua keluaran itu sama pada setiap karakter kecuali dua: Java mencetak nilai benar-atau-salah sebagai `true` dan `false`, sedangkan Python sebagai `True` dan `False`, karena begitulah masing-masing bahasa mengejanya.

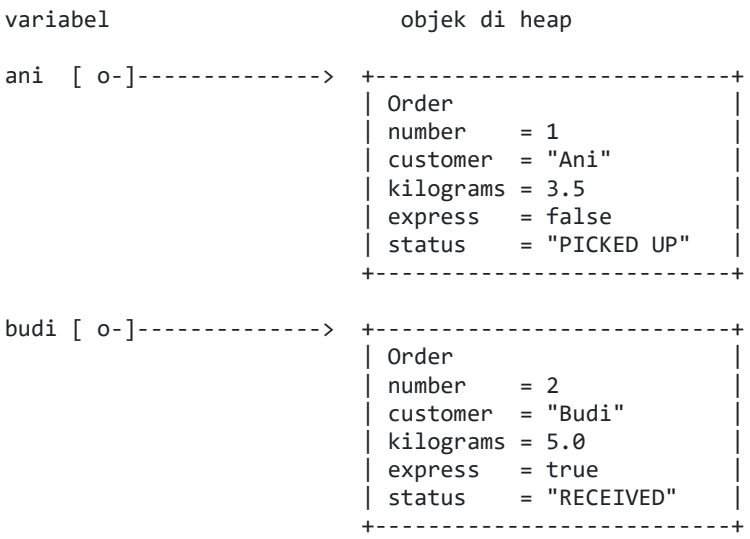
Telusurilah versi Java baris demi baris, dengan **Gambar 1.12** di samping Anda.

- `Order ani = new Order(1, "Ani", 3.5, false);` melakukan tiga hal. `new Order(...)` menciptakan objek baru dan menjalankan konstruktor dengan keempat nilai itu, yang mengatur *field-field*-nya dan statusnya menjadi RECEIVED. `Order ani` mendeklarasikan variabel yang dapat merujuk ke sebuah `Order`. Dan `=` membuat `ani` merujuk ke objek baru itu. `ani = Order(1, "Ani", 3.5, False)` di Python melakukan hal yang sama tanpa kata `new` dan tanpa tipe. Pesanan Budi diciptakan dengan cara yang sama, dengan ekspres bernilai benar.
- `System.out.println(ani);` mencetak objek itu; Java memanggil `ani.toString()` di balik layar, yang menyusun baris keluaran pertama. Harganya, 24500, dihitung oleh objek itu sendiri dengan Persamaan (1.1); 52500 milik Budi dengan Persamaan (1.2).
- `ani.pickUp()` mengirim pesan *pickUp* kepada pesanan Ani ketika statusnya RECEIVED. Metode itu memeriksa, mendapati statusnya bukan READY, dan mengembalikan `false` — lingkaran merah pada **Gambar 1.12**. Baris keluaran menyatakannya, dan statusnya tetap RECEIVED.
- `ani.startWashing();` dan `ani.markReady();` memajukan pesanan melalui dua panah hitam. Kita mengabaikan `true` yang dikembalikan masing-masing.
- `ani.getStatus()` kini melaporkan `READY`, dan `ani.pickUp()` berhasil dan mengembalikan `true`.
- Dua baris terakhir mencetak kedua pesanan lagi. Pesanan Ani PICKED UP; pesanan Budi masih RECEIVED, karena setiap pesan dikirim kepada `ani` dan pesanan Budi memiliki keadaannya sendiri yang terpisah. Itulah **identitas** yang sedang bekerja: dua objek dari kelas yang sama, masing-masing mengurus urusannya sendiri.

> **CONTOH:** Cobalah melanggar aturannya. Tambahkan baris `ani.status = "PICKED UP";` pada `OrderDemo.java` sebelum `println` yang pertama, lalu kompilasi. Kompilator menolak dengan `error: status has private access in Order`, dan tidak ada program yang dihasilkan sama sekali. Kesalahan yang dibuat **Kode 1.1** secara diam-diam kini mustahil dibuat di Java tanpa mengubah kelas `Order` itu sendiri. Hapus lagi baris itu sebelum melanjutkan.

1.8.6 Apa yang disimpan variabel bertipe kelas

Satu gambar lagi melengkapi model mental Anda. Ketika Anda menulis `Order ani = new Order(...)`, variabel `ani` tidak berisi pesanan itu sendiri; ia berisi sebuah **referensi** (*reference*) — pada dasarnya, alamat tempat objek itu tinggal di memori komputer. Objek itu tinggal di wilayah memori yang disebut ***heap***. Anda dapat menggambarnya seperti ini:



Perbedaan itu penting begitu Anda menulis `Order x = ani;`. Baris itu tidak menyalin pesanannya; ia menyalin *referensinya*, sehingga `x` dan `ani` kini menunjuk objek yang sama, dan `x.pickUp()` mengubah pesanan yang sama yang dilihat `ani`. Python berperilaku sama: setiap variabel Python menyimpan referensi ke sebuah objek. Pemrogram C akan mengenali referensi sebagai saudara *pointer* yang lebih aman — Anda dapat mengikutinya, tetapi tidak dapat melakukan aritmetika terhadapnya atau mengarahkannya ke memori sembarang. Bab 3 kembali membahas referensi, dan apa yang terjadi pada sebuah objek ketika tidak ada lagi variabel yang merujuknya.

1.8.7 Apa yang kita peroleh dari kelas

Bandingkan **Kode 1.9** dengan **Kode 1.1**. Buku catatan itu memiliki informasi yang sama dan rumus harga yang sama, tetapi aturan-aturannya tidak bertempat di mana pun. Kelas menempatkan satu-satunya aturannya di tepat satu tempat, ketiga metode status, dan setiap program yang memakai pesanan memperoleh aturan itu dengan cuma-cuma dan tidak dapat mengakalinya. Jika Bu Sari menambahkan satu langkah, misalnya IRONING (disetrika) di antara WASHING dan READY, perubahan dilakukan di satu kelas dan setiap program mengikutinya.

Kelas itu belum selesai. Ia masih menerima pesanan seberat minus lima kilogram, karena konstruktornya tidak memeriksa parameternya; statusnya berupa *string*, sehingga salah ketik seperti `"REDAY"` di dalam kelas tidak akan ketahuan; dan harga-harganya tertanam di dalam kode. Bab 3 mengajarkan perkakas untuk memeriksanya, Bab 5 mengajarkan eksepsi untuk menolak masukan yang buruk dengan benar, dan bab-bab selanjutnya mengganti *string* status dengan tipe yang tidak mungkin salah eja. Kelas yang baik tumbuh selangkah demi selangkah dengan cermat.

1.9 Peta Perjalanan Buku Ini

Buku ini disusun mengikuti irama satu semester perkuliahan: enam belas bab, satu untuk setiap minggu. Di akhir buku Anda akan mampu **menjelaskan** konsep-konsep pemrograman berorientasi objek dan fitur-fitur bahasa yang mendukungnya; **menganalisis** sebuah masalah

dan **memodelkan** penyelesaiannya dengan objek, seperti yang mulai dilakukan Subbab 1.3 untuk kios *laundry*; dan **mengimplementasikan** penyelesaian itu dalam dua bahasa berorientasi objek — merancang, membangun, dan menguji program yang bermutu baik, baik sendiri maupun dalam tim. **Tabel 1.7** membentangkan rutenya.

Tabel 1.7 Keenam belas bab buku ini dan apa yang ditambahkan masing-masing.

Bab	Judul	Yang ditambahkannya
1	Objek, Program, dan Meja Kerja Pertama Anda	apa itu program dan objek; Java dan Python; VS Code
2	Lingkungan Pengembangan: Menyunting, Menjalankan, dan Men-debug	Eclipse dan VS Code; proyek dan paket; *debugger*; pengemasan
3	Kelas, Objek, String, Tipe, Percabangan, dan Perulangan	tata bahasa kedua bahasa; menciptakan objek; referensi
4	Evaluasi dan Pembahasan I	soal-soal yang diselesaikan tentang Bab 1–3
5	Larik, List, Masukan, Eksepsi, dan Berkas	kumpulan objek; membaca masukan; menangani kesalahan; berkas dan *stream*
6	Dokumentasi, Pengujian, dan Antarmuka	Javadoc dan *docstring*; JUnit dan `unittest`; antarmuka; *overloading*
7	Pewarisan, Polimorfisme, dan Kontrol Akses	memperluas kelas; *overriding*; kelas abstrak; *casting*; tingkat akses
8	Evaluasi dan Pembahasan II	soal-soal yang diselesaikan tentang Bab 5–7
9	Koleksi dan Generik	*list*, *set*, dan *map*; tipe generik
10	Antarmuka Grafis dan Penanganan Peristiwa	Swing dan Tkinter; menggambar; peristiwa; *inner class* dan *closure*
11	Pola Model–View–Controller	aplikasi lengkap yang dibangun dari tiga bagian yang bekerja sama
12	Evaluasi dan Pembahasan III	soal-soal yang diselesaikan tentang Bab 9–11
13	Iterator, Imutabilitas, dan Koleksi Lanjutan	menelusuri koleksi; objek yang tidak pernah berubah
14	Thread, Race, Deadlock, dan Livelock	mengerjakan beberapa hal sekaligus, dengan aman
15	Socket, Jaringan, dan Proyek Tim	program yang bercakap melalui jaringan; membangun dalam tim
16	Evaluasi dan Pembahasan IV	soal-soal yang diselesaikan tentang Bab 13–15 dan seluruh buku

Setiap bab keempat — Bab 4, 8, 12, dan 16 — mengevaluasi dan membahas tiga bab sebelumnya: tanpa teori baru, tetapi dengan soal-soal yang diselesaikan secara lengkap dalam kedua bahasa, dengan penalaran yang ditunjukkan langkah demi langkah, serta soal-soal lanjutan untuk Anda coba.

Kios **Profitina Laundry** hadir di sepanjang buku, dan setiap bab menambahkan sesuatu kepadanya: pelanggan dan daftar harga di Bab 3, berkas pesanan harian di Bab 5, pengujian yang membuktikan kebenaran harga di Bab 6, pesanan ekspres dan cuci kering sebagai subkelas di Bab 7, jendela untuk meja layanan di Bab 10, beberapa meja layanan yang dilayani sekaligus di Bab 14, dan kios yang bercakap dengan kurirnya melalui jaringan di Bab 15. Di akhir buku, nota tiga baris pada **Pseudokode 1.1** akan telah menjadi sebuah aplikasi yang kecil tetapi lengkap.

Dua buku menjadi pendamping yang baik bagi buku ini. Untuk Java, buku teks Deitel dan Deitel yang sudah lama beredar [24] membahas bahasa itu secara sangat mendalam, dengan jauh lebih banyak latihan daripada yang dapat dimuat satu buku. Untuk Python, *Learning Python* karya Lutz [25] menjelaskan cara kerja bagian dalam bahasa itu secara tuntas. Dan untuk segala hal lainnya, dokumentasi resmi kedua bahasa, Java Language Specification [3] dan Python Language

Reference [8], tersedia gratis di web dan lebih mudah dibaca daripada kesan yang ditimbulkan judulnya.

1.10 Profitina dalam Praktik

Profitina adalah **platform visibilitas AI (GEO)**, yang beroperasi di `app.profitina.com`. GEO — *Generative Engine Optimization* — adalah disiplin agar *dapat dikutip oleh mesin penjawab*, sebagaimana SEO adalah disiplin agar berperingkat tinggi dalam daftar tautan. Produk ini memindai situs web sebuah merek, mengukur seberapa siap kontennya dikutip oleh mesin penjawab AI dari sinyal-sinyal nyata — apakah konten utamanya dapat diekstrak dengan bersih, apakah ia memuat data terstruktur, apakah situsnya sehat secara teknis, hingga `robots.txt` yang tersusun baik — menjelaskan setiap sinyal yang diukurnya, lalu membantu memperbaiki apa yang ditemukannya.

Platform ini ditulis dalam **Python**, sebagai aplikasi web FastAPI, dan gagasan-gagasan bab ini tampak dalam bentuknya sejak berkas pertama.

Sinyal adalah objek. Sebuah pemindaian tidak menghitung satu angka raksasa dalam satu fungsi raksasa. Setiap hal yang diukur pemindai adalah *sinyal* tersendiri, sebuah objek dari kelasnya sendiri tepat dalam pengertian Subbab 1.3: ia menyimpan keadaannya sendiri (apa yang ditemukannya, skor yang diberikannya) dan perilakunya sendiri (mengukur sebuah halaman, memberi skor pada temuan, menjelaskan skor itu dengan kata-kata sederhana). Pemindai berperilaku seperti kasir pada **Gambar 1.3**: ia tidak tahu bagaimana cara kerja bagian dalam sinyal mana pun; ia mengirim pesan yang sama kepada masing-masing — *ukurlah halaman ini*, *berapa skormu?*, *jelaskan dirimu* — dan mengumpulkan jawabannya. Menambahkan sinyal berarti menulis satu kelas baru; pemindainya tidak berubah. Itulah janji abstraksi dari Subbab 1.3.5, dan Bab 6 membuatnya tepat dengan sebuah **antarmuka** yang diimplementasikan setiap kelas sinyal.

Model AI berada di balik sebuah objek. *Content Engine* (mesin konten) Profitina menyusun draf teks yang ramah GEO dengan **Qwen2.5 yang berjalan secara lokal** di server Profitina sendiri melalui Ollama; tidak ada data pengguna yang dikirim ke layanan AI dari luar. Bagian lain aplikasi tidak pernah bercakap langsung dengan model itu. Mereka bercakap dengan sebuah objek yang membungkusnya, dan objek itu memiliki aturannya sendiri, sama seperti `Order.pickUp()`: ketika tidak ada model yang tersedia, ia *mengatakannya* alih-alih diam-diam mengembalikan templat, dan sebuah draf hanya disajikan jika skornya lebih baik daripada templat bawaan. Karena aturan-aturan itu tinggal di satu kelas, tidak ada bagian kode lain yang dapat mengakalinya secara tidak sengaja.

Hukum adalah bagian dari desain. Platform ini menangani data pribadi, sehingga tunduk pada Undang-Undang Pelindungan Data Pribadi Indonesia, UU PDP 27/2022 [26]: persetujuan harus eksplisit, data harus dapat diekspor, dan penghapusan akun harus benar-benar menghapus. Enkapsulasi adalah cara sebuah program menepati janji-janji seperti ini: jika hanya satu kelas yang boleh menyentuh data pribadi pengguna, ada tepat satu tempat untuk memeriksa persetujuan, satu tempat yang mengekspor, satu tempat yang menghapus. Dan satu prinsip produk ini berlaku bagi setiap kelas yang akan Anda tulis dalam buku ini: **aplikasi memberi saran; pengguna yang memutuskan.** Skor visibilitas adalah hasil pengukuran, tidak pernah jaminan.

1.10.1 Studi kasus terapan: sehari di Profitina Laundry

Studi kasus praktik dalam buku ini selalu **Profitina Laundry**, sebuah usaha kecil untuk keperluan belajar, tidak pernah produk yang sebenarnya. Berikut satu hari penuh di meja layanannya, dibangun dari kelas `Order` pada Subbab 1.8 tanpa mengubah satu baris pun. **Kode 1.13** dan **Kode 1.14** menciptakan empat pesanan, memajukan mereka sepanjang hari, dan menjumlahkan uang dari pesanan yang sudah diambil.

Kode 1.13 `ShopDay.java`: sehari di Profitina Laundry, memakai kelas `Order` tanpa perubahan; kompilasi bersama `Order.java`.

```
public class ShopDay {
    public static void main(String[] args) {
        Order[] orders = {
            new Order(101, "Ani", 3.5, false),
            new Order(102, "Budi", 5.0, true),
            new Order(103, "Citra", 2.0, false),
            new Order(104, "Dewi", 7.5, true)
        };

        // Giliran pagi: setiap pesanan masuk ke mesin.
        for (Order o : orders) {
            o.startWashing();
        }
        // Menjelang siang, tiga pesanan pertama keluar kering dan terlipat.
        orders[0].markReady();
        orders[1].markReady();
        orders[2].markReady();
        // Sore hari, Ani dan Citra mengambil cucian mereka.
        orders[0].pickUp();
        orders[2].pickUp();
        // Dewi datang terlalu cepat dan ditolak dengan sopan.
        boolean dewiOk = orders[3].pickUp();

        long cash = 0;
        int waiting = 0;
        for (Order o : orders) {
            System.out.println(o);
            if (o.getStatus().equals("PICKED UP")) {
                cash = cash + o.price();
            } else {
                waiting = waiting + 1;
            }
        }
        System.out.println("Dewi's early pick-up accepted? " + dewiOk);
        System.out.println("Cash collected today: Rp " + cash);
        System.out.println("Orders still in the shop: " + waiting);
    }
}
```

Kode 1.14 `shop_day.py`: hari yang sama dalam Python; program ini mengimpor `Order` dari `order.py` di folder yang sama.

```python
from order import Order

orders = [
    Order(101, "Ani", 3.5, False),
    Order(102, "Budi", 5.0, True),
    Order(103, "Citra", 2.0, False),
    Order(104, "Dewi", 7.5, True),
]

# Giliran pagi: setiap pesanan masuk ke mesin.
for o in orders:
    o.start_washing()
# Menjelang siang, tiga pesanan pertama keluar kering dan terlipat.
orders[0].mark_ready()
orders[1].mark_ready()
orders[2].mark_ready()
# Sore hari, Ani dan Citra mengambil cucian mereka.
orders[0].pick_up()
orders[2].pick_up()
# Dewi datang terlalu cepat dan ditolak dengan sopan.
dewi_ok = orders[3].pick_up()

cash = 0
waiting = 0
for o in orders:
    print(o)
    if o.get_status() == "PICKED UP":
        cash = cash + o.price()
    else:
        waiting = waiting + 1
print("Dewi's early pick-up accepted? " + str(dewi_ok))
print("Cash collected today: Rp " + str(cash))
print("Orders still in the shop: " + str(waiting))
```

Kompilasi dan jalankan versi Java dengan `javac Order.java ShopDay.java` dan `java ShopDay`, versi Python dengan `python shop_day.py`. Versi Java mencetak:

```
Order #101 for Ani: 3.5 kg, Rp 24500, PICKED UP
Order #102 for Budi: 5.0 kg express, Rp 52500, READY
Order #103 for Citra: 2.0 kg, Rp 14000, PICKED UP
Order #104 for Dewi: 7.5 kg express, Rp 78750, WASHING
Dewi's early pick-up accepted? false
Cash collected today: Rp 38500
Orders still in the shop: 2
```

dan versi Python mencetak baris-baris yang sama, kecuali baris kelimanya berbunyi `Dewi's early pick-up accepted? False`.

Dua unsur tata bahasa baru muncul. `Order[] orders = { ... };` di Java, dan `orders = [ ... ]` di Python, membuat kumpulan empat pesanan; Bab 5 membahas larik dan *list* secara lengkap. Dan `for (Order o : orders)` di Java, `for o in orders:` di Python, mengunjungi setiap pesanan secara bergiliran, menyebutnya `o` — perulangan yang terbaca hampir seperti kalimat biasa.

Periksa angka-angkanya. Pesanan ekspres Dewi seberat 7.5 kilogram berharga $p(7.5) = 52500$ ditambah $\lfloor 52500/2 \rfloor = 26250$, yaitu 78750, menurut Persamaan (1.2). Uang kasnya adalah 24500 milik Ani ditambah 14000 milik Citra, yaitu 38500; pesanan Budi sudah siap tetapi belum diambil, dan pesanan Dewi masih dicuci, sehingga dua pesanan tetap berada di kios. Dan perhatikan baris yang paling penting: Dewi datang terlalu cepat, dan pesanannya *menolak*. `ShopDay` tidak memerlukan satu baris pun untuk menegakkan aturan itu. Kelasnyalah yang melakukannya.

1.11 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya

Menjalankan versi lama program Java. Anda menyunting `Hello.java`, menjalankan `java Hello`, dan melihat keluaran lama. Anda lupa mengompilasi ulang; `java` menjalankan `Hello.class` yang lama. Obatnya: setelah setiap penyuntingan, jalankan `javac` lebih dahulu — atau gunakan tombol Run di VS Code, yang mengompilasi untuk Anda, atau `java Hello.java` untuk satu berkas.

Nama berkas dan nama kelas yang berbeda. Berkas `hello.java` yang berisi `public class Hello` ditolak: *class Hello is public, should be declared in a file named Hello.java*. Aturan itu mencakup huruf kapital, bahkan di Windows, yang sistem berkasnya sendiri tidak peduli. Obatnya: namai berkas persis menurut kelas publiknya.

Lupa bahwa Java peka huruf besar-kecil. `system.out.println` dengan s kecil ditolak dengan pesan yang membingungkan *package system does not exist*, karena Java menganggap `system` sebagai nama paket. `System`, `String`, dan `Math` berhuruf kapital; `main`, `println`, dan `equals` tidak. Python sama pekanya: `Print("hi")` gagal dengan *NameError: name 'Print' is not defined*. Obatnya: salin nama-nama persis seperti yang ditulis buku ini, dan biarkan pelengkapan otomatis VS Code menyarankannya.

Titik koma yang hilang di Java, indentasi yang nyasar di Python. Setiap bahasa memiliki satu tanda baca yang sering terlewat oleh pemula. Java memerlukan `;` di akhir setiap pernyataan, dan pesannya, *';' expected*, menunjuk tepat sesudah tempat tanda itu hilang. Python memerlukan indentasi yang konsisten; baris yang menjorok tanpa alasan menghasilkan *IndentationError: unexpected indent*. Obatnya: bacalah baris yang disebut pesan kesalahan, dan baris *sebelum*-nya.

Membandingkan *string* dengan `==` di Java. Di Java, `==` di antara dua objek menanyakan apakah keduanya *objek yang sama* (referensi yang sama, dalam gambar di Subbab 1.8.6), bukan apakah keduanya berisi karakter yang sama. Dua *string* dengan teks yang sama dapat berupa objek yang berbeda, sehingga `status == "READY"` bisa bernilai salah meskipun statusnya berbunyi READY. Obatnya: selalu bandingkan *string* di Java dengan `equals`, seperti dilakukan **Kode 1.9**. Di Python, `==` membandingkan isinya dan sudah benar.

Membuat *field* publik "agar hemat waktu". Hemat semenit hari ini dan rugi sehari bulan depan, ketika sebuah baris di tempat yang jauh menulis hal yang tidak masuk akal ke dalam *field* itu, persis seperti pada **Kode 1.1**. Obatnya: jadikan setiap *field* `private` (di Python, beri garis bawah di depan), dan tambahkan satu metode untuk setiap hal yang secara sah perlu dilakukan pihak luar.

Mencampuradukkan kelas dengan objek. "Saya mengubah status `Order`" adalah kalimat yang tidak bermakna: kelas tidak memiliki status; setiap objek memiliki statusnya sendiri. Obatnya: ketika berbicara tentang keadaan, sebutkan objeknya — "pesanan Ani", `ani` — bukan kelasnya.

Bergulat dengan instalasi alih-alih membaca pesannya. Hampir setiap kegagalan penyiapan — perkakas yang "tidak dikenali", versi yang salah, Python yang salah di VS Code, terminal yang sudah terbuka sebelum instalasi — tercantum di **Tabel 1.6**. Obatnya: bacalah baris pertama pesannya, temukan di tabel itu, dan terapkan perbaikannya sebelum mencoba hal lain.

Memformat bilangan dengan pengaturan regional komputer. `String.format("%.1f", 3.5)` di Java mencetak `3.5` di komputer yang diatur ke bahasa Inggris tetapi `3,5` di komputer yang diatur ke bahasa Indonesia, karena ia mengikuti konvensi regional untuk koma desimal. Itulah sebabnya listing-listing bab ini menyambungkan bilangan dengan `+`, yang selalu mencetak `3.5`. Obatnya: ketika keluaran harus sama di mana pun — di berkas, di pengujian, di buku ini —

hindari pemformatan regional atau sebutkan konvensinya secara eksplisit; Bab 5 menunjukkan caranya.

Memercayai program karena program itu berjalan. Program yang mencetak *sesuatu* tidak otomatis benar. Buku catatan pada **Kode 1.1** berjalan mulus dan mencetak harga negatif. Obatnya: periksa setiap keluaran dengan tangan terhadap sebuah rumus, seperti dilakukan bab ini dengan Persamaan (1.1) dan Persamaan (1.2); Bab 6 mengubah pemeriksaan semacam itu menjadi pengujian otomatis.

1.12 Ringkasan Bab

Program adalah algoritma — daftar berhingga langkah-langkah yang tepat dari suatu masukan menuju suatu keluaran — yang ditulis dalam bahasa pemrograman. Java **dikompilasi** oleh `javac` menjadi ***bytecode***, yang dijalankan **Java Virtual Machine** di platform mana pun; Python dijalankan oleh **interpreter** yang mengompilasi ke *bytecode* secara internal setiap kali. Di Java Anda mengompilasi lalu menjalankan; di Python Anda cukup menjalankan.

Pemrograman berorientasi objek menata program sebagai masyarakat **objek**, masing-masing dengan **keadaan** (apa yang diketahuinya), **perilaku** (apa yang dapat dilakukannya), dan **identitas** (ia adalah dirinya dan bukan yang lain). **Kelas** adalah formulir kosong; setiap objek adalah ***instance*** dengan nilai-nilainya sendiri. Objek-objek bekerja sama dengan mengirim **pesan**, yang ditulis dengan titik: `ani.price()`. **Abstraksi** menyimpan hanya yang penting dan menyembunyikan *bagaimana* di balik *apa*. Enkapsulasi, abstraksi, pewarisan, dan polimorfisme diajarkan secara lengkap di Bab 3, 6, dan 7.

Larik paralel dan fungsi-fungsi lepas tidak menugasi siapa pun untuk menjaga data tetap masuk akal; baris mana pun dapat merusaknya. Sebuah objek membungkus datanya bersama satu-satunya metode yang boleh mengubahnya, dan metode-metode itu dapat **menolak** permintaan yang tidak masuk akal.

Java (1995, Sun Microsystems, kini Oracle) portabel, didukung luas, dan eksplisit tentang tipe dan akses; Python (1991) ringkas, bertipe dinamis, dan mendominasi pekerjaan data dan AI; keduanya berada di tiga besar setiap pengukuran popularitas utama. Buku ini memakai **Java 25 LTS** (Eclipse Temurin) dan **Python 3.14**, dan setiap program juga berjalan pada Java 17+ dan Python 3.10+.

Meja kerjanya adalah **VS Code** dengan **Extension Pack for Java** dan ekstensi **Python**. Periksa perkakasnya dengan `java -version`, `javac -version`, dan `python --version` (`python3` di macOS dan Linux); jalankan dengan tombol Run atau dari terminal; lakukan *debugging* dengan *breakpoint*, `F5`, `F10`, `F11`, dan panel Variables.

Kelas pertama, `Order`, memiliki lima *field* **private**, sebuah **konstruktor**, metode `price()` yang mengimplementasikan Persamaan (1.1) dan Persamaan (1.2), dan tiga metode status yang memajukan pesanan menyusuri **Gambar 1.12** dan menolak memajukannya pada saat yang salah. Variabel bertipe kelas menyimpan **referensi** ke sebuah objek di *heap*, bukan objek itu sendiri.

1.13 Soal Latihan

1.1 Pilihlah tempat yang Anda kenal baik selain kios *laundry* — kantin sekolah, warung, terminal bus, perpustakaan. Sebutkan empat jenis objek di dalamnya, dan untuk masing-masing berikan paling sedikit tiga butir keadaan dan tiga perilaku, dalam bentuk **Tabel 1.1**.

1.2 Pilahlah yang berikut ini menjadi kelas dan objek, dan untuk setiap objek sebutkan kelas yang menjadi asalnya sebagai *instance*: `WashingMachine`; mesin di dekat pintu, bernama Mesin A; `Customer`; Budi; `Order`; pesanan nomor 104; Mesin B.

1.3 Dengan Persamaan (1.1) dan Persamaan (1.2), hitunglah dengan tangan harga (a) 2.3 kilogram layanan biasa, (b) 7.5 kilogram ekspres, (c) 0.9 kilogram ekspres. Lalu periksa (b) terhadap keluaran **Kode 1.13**.

1.4 Dengan Persamaan (1.3) dan Persamaan (1.4), hitunglah jumlah muatan untuk pesanan seberat 16, 17, dan 1 kilogram, pertama dengan mesin berkapasitas $c = 8$, lalu dengan $c = 7$. Tunjukkan pembagian bilangan bulat untuk masing-masing.

1.5 Ubahlah **Kode 1.3** dan **Kode 1.4** sehingga masing-masing mencetak baris ketiga, `Open every day from 07.00 to 20.00`. Jalankan keduanya. Baris apa yang Anda tambahkan dalam setiap bahasa, dan di mana tepatnya baris itu harus diletakkan?

1.6 Perkirakan apa yang terjadi dalam setiap kasus, lalu cobalah: (a) **Kode 1.3** disimpan sebagai `hello.java` dengan h kecil; (b) titik koma di akhir baris `println` pertama pada **Kode 1.3** dihapus; (c) `System` ditulis `system`; (d) baris kedua sebuah berkas Python menjorok empat spasi sedangkan baris pertama tidak.

1.7 Andaikan dua baris `ani.startWashing();` dan `ani.markReady();` pada **Kode 1.11** ditukar, sehingga `ani.markReady()` dipanggil *sebelum* `ani.startWashing()`. Apa yang dikembalikan masing-masing pemanggilan itu, dan apa yang dilaporkan `ani.getStatus()` sesudahnya? Gunakan **Gambar 1.12** untuk membenarkan jawaban Anda.

1.8 Pada **Kode 1.13**, tambahkan baris `orders[1].pickUp();` tepat sesudah pengambilan oleh Citra. Berapa jumlah uang kas yang baru dan berapa pesanan yang masih berada di kios? Periksa dengan menjalankannya.

1.9 Tulislah kelas `WashingMachine`, dalam Java dan Python, mengikuti uraian Subbab 1.3.1: sebuah mesin memiliki nama, kapasitas dalam kilogram bulat, kilogram yang sedang dimuat, dan apakah ia sedang berputar. Kelas itu menawarkan `load(kg)`, yang menolak (mengembalikan *false*) jika mesin sedang berputar, jika `kg` tidak positif, atau jika muatannya akan melebihi kapasitas; `start()`, yang menolak jika mesin sudah berputar atau kosong; `finish()`, yang menghentikan mesin, mengosongkannya, dan mengembalikan kilogram yang dicuci (atau 0 jika mesin tidak sedang berputar); dan deskripsi satu baris. Beri kelas itu sebuah `main` yang menguji setiap penolakan.

1.10 *Field* `status` pada `Order` bersifat `private`, dan *field* `number`, `customer`, `kilograms`, dan `express` juga `final`. Jelaskan masing-masing dalam satu atau dua kalimat: (a) apa yang bisa salah jika `status` bersifat publik; (b) mengapa `status` tidak dapat dibuat `final`; (c) apa yang dilindungi `final` pada `kilograms`.

1.11 Bu Sari ingin setiap pesanan mencatat nomor telepon pelanggan. Daftarlah setiap perubahan yang diperlukan pada **Kode 1.1** yang prosedural, dan setiap perubahan yang diperlukan pada **Kode 1.9** dan **Kode 1.11** yang berorientasi objek. Versi mana yang memerlukan perubahan di lebih banyak tempat, dan mengapa?

1.12 (a) Di Python, berapakah nilai `9 / 8`, dan berapakah nilai `9 // 8`? Apa yang akan dicetak **Kode 1.6** untuk pesanan kedua jika barisnya memakai `/` alih-alih `//` dan hasilnya kemudian diubah menjadi bilangan bulat dengan `int(...)`? (b) Tunjukkan bahwa Persamaan (1.4) berlaku untuk $c = 1$ dan setiap bilangan bulat $w \geq 0$.

1.14 Penyelesaian Soal Latihan

Penyelesaian 1.1. Untuk kantin sekolah, salah satu jawaban yang baik adalah sebagai berikut. *Lapak makanan* mengetahui namanya, menunya beserta harga, dan apakah ia sedang buka; ia dapat menerima pesanan, memberi tahu harga, serta membuka dan menutup lapak. *Siswa* mengetahui nama, kelas, dan uang di sakunya; ia dapat memesan, membayar, dan makan. *Nampan* mengetahui hidangan yang dibawanya dan apakah ia sudah dikembalikan; ia dapat diisi, dibawa, dan dikembalikan. *Kotak kasir* mengetahui uang kas hari ini dan daftar pembayaran; ia dapat menerima pembayaran, memberi kembalian, dan melaporkan total hari itu. Hal terpenting dalam jawaban apa pun adalah pemilahannya: keadaan adalah informasi, dalam kata benda; perilaku adalah tindakan, dalam kata kerja; dan setiap perilaku memakai atau mengubah keadaan objek itu sendiri.

Penyelesaian 1.2. Kelas-kelasnya adalah `WashingMachine`, `Customer`, dan `Order` — masing-masing adalah suatu jenis benda, sebuah formulir kosong. Objek-objeknya adalah Mesin A dan Mesin B, keduanya *instance* dari `WashingMachine`; Budi, *instance* dari `Customer`; dan pesanan nomor 104, *instance* dari `Order`. Patokan yang berguna: nama kelas menjawab "jenis apa?", sebuah objek menjawab "yang mana?".

Penyelesaian 1.3. (a) $p(2.3) = \text{round}(7000 \times 2.3) = \text{round}(16100) = 16100$, jadi Rp16.100. (Di dalam komputer, 7000×2.3 menghasilkan 16099.999999999998 karena 2.3 tidak dapat disimpan secara tepat dalam biner; pembulatan pada Persamaan (1.1) itulah yang membuat hasilnya tepat 16100.) (b) $p(7.5) = 52500$ dan $p_{\text{ekspres}}(7.5) = 52500 + \lfloor 52500/2 \rfloor = 52500 + 26250 = 78750$, jadi Rp78.750 — persis yang dicetak **Kode 1.13** untuk Dewi. (c) $p(0.9) = 6300$ dan $p_{\text{ekspres}}(0.9) = 6300 + 3150 = 9450$, jadi Rp9.450.

Penyelesaian 1.4. Dengan $c = 8$, Persamaan (1.4) menghitung $\lfloor (w + 7)/8 \rfloor$. Untuk $w = 16$: $\lfloor 23/8 \rfloor = 2$ (karena $23 = 2 \times 8 + 7$). Untuk $w = 17$: $\lfloor 24/8 \rfloor = 3$. Untuk $w = 1$: $\lfloor 8/8 \rfloor = 1$. Hasil ini sesuai dengan Persamaan (1.3): $\lceil 2 \rceil = 2$, $\lceil 2.125 \rceil = 3$, $\lceil 0.125 \rceil = 1$. Dengan $c = 7$ rumusnya $\lfloor (w + 6)/7 \rfloor$. Untuk $w = 16$: $\lfloor 22/7 \rfloor = 3$ (karena $22 = 3 \times 7 + 1$), dan memang $\lceil 16/7 \rceil = \lceil 2.29 \rceil = 3$. Untuk $w = 17$: $\lfloor 23/7 \rfloor = 3$, dan $\lceil 2.43 \rceil = 3$. Untuk $w = 1$: $\lfloor 7/7 \rfloor = 1$.

Penyelesaian 1.5. Di Java, tambahkan `System.out.println("Open every day from 07.00 to 20.00");` sebagai baris baru *di dalam* `main`, sesudah `println` kedua dan sebelum kurung kurawal penutup `main`; beri indentasi seperti baris-baris tetangganya. Di luar `main` baris itu tidak akan dapat dikompilasi, karena pernyataan Java harus berada di dalam sebuah metode. Di Python, tambahkan `print("Open every day from 07.00 to 20.00")` sebagai baris terakhir yang baru, *tanpa* indentasi. Kedua program lalu mencetak dua baris sebelumnya diikuti `Open every day from 07.00 to 20.00`.

Penyelesaian 1.6. (a) Kompilator menolak, dan di mesin buku ini mencetak persis:

```
hello.java:1: error: class Hello is public, should be declared in a file named Hello.java
public class Hello {
       ^
1 error
```

(b) Kompilator melaporkan titik koma yang hilang dan menunjuk tepat sesudah tempat seharusnya titik koma itu berada:

```
Hello.java:3: error: ';' expected
       System.out.println("Hello from Profitina Laundry!")
                                                          ^
1 error
```

(c) Kompilator menganggap `system` sebagai nama paket dan mencetak `Hello.java:3: error: package system does not exist`, dengan tanda sisipan di bawah titik sesudah `system`. (d) Python menolak sebelum menjalankan satu baris pun. Baris terakhir pesannya adalah `IndentationError: unexpected indent`, dan baris di atasnya menampilkan baris yang bermasalah; baris pertama pesan itu menyebut nama berkas dan berkata `line 2`. Dalam setiap kasus pelajarannya sama: bacalah nama berkas dan nomor baris dalam pesan, lihatlah baris itu dan baris sebelumnya, lalu perbaiki kesalahan pertama lebih dahulu.

Penyelesaian 1.7. `ani.markReady()` akan mengembalikan `false`: `markReady()` hanya berhasil dari WASHING, dan pesanan itu masih RECEIVED, sehingga metode itu menolak dan tidak mengubah apa pun. Kemudian `ani.startWashing()` akan mengembalikan `true` dan memindahkan pesanan ke WASHING. `ani.getStatus()` akan melaporkan `WASHING`, bukan READY. Pada **Gambar 1.12**, satu-satunya panah yang keluar dari RECEIVED adalah `startWashing()`; pesan lain apa pun yang dikirim di sana ditolak. (Akibatnya `ani.pickUp()` berikutnya pada **Kode 1.11** juga akan mengembalikan `false`.)

Penyelesaian 1.8. Pesanan Budi berstatus READY pada saat itu, sehingga `orders[1].pickUp()` berhasil. Uang kas menjadi $24500 + 52500 + 14000 = 91000$, dan hanya pesanan Dewi yang tersisa, sehingga program mencetak `Cash collected today: Rp 91000` dan `Orders still in the shop: 1`, dan baris Budi berakhir dengan `PICKED UP`. Menjalankan program yang telah diubah itu mengonfirmasi kedua angka tersebut.

Penyelesaian 1.9. **Kode 1.15** dan **Kode 1.16** adalah salah satu penyelesaiannya. Seperti `Order`, setiap metode memeriksa lebih dahulu dan hanya bertindak jika permintaannya masuk akal.

Kode 1.15 `WashingMachine.java`: mesin cuci yang menolak muatan berlebih, pengisian ketika sedang berputar, dan penyalaan ketika kosong.

```
public class WashingMachine {
    private final String name;
    private final int capacity;     // kilogram
    private int loadKg;             // kilogram yang ada di dalam saat ini
    private boolean running;

    public WashingMachine(String name, int capacity) {
        this.name = name;
        this.capacity = capacity;
        this.loadKg = 0;
        this.running = false;
    }

    public boolean load(int kg) {
        if (running || kg <= 0 || loadKg + kg > capacity) {
            return false;           // tolak: sedang berputar, tidak masuk akal, atau
terlalu berat
        }
        loadKg = loadKg + kg;
        return true;
    }

    public boolean start() {
        if (running || loadKg == 0) {
            return false;           // tidak ada yang dicuci, atau sudah mencuci
        }
        running = true;
        return true;
    }

    public int finish() {
        if (!running) {
            return 0;
        }
        int washed = loadKg;
        running = false;
        loadKg = 0;
        return washed;              // kilogram yang keluar dalam keadaan bersih
    }

    @Override
    public String toString() {
        return name + " [" + loadKg + "/" + capacity + " kg, "
                + (running ? "running" : "idle") + "]";
    }
```

Kode 1.15 (Lanjutan) *WashingMachine.java: mesin cuci yang menolak muatan berlebih, pengisian ketika sedang berputar, dan penyalaan ketika kosong.*

```
    public static void main(String[] args) {
        WashingMachine m = new WashingMachine("Machine A", 8);
        System.out.println(m);
        System.out.println("load 5 kg: " + m.load(5));
        System.out.println("load 4 kg: " + m.load(4));
        System.out.println("load 3 kg: " + m.load(3));
        System.out.println("start:     " + m.start());
        System.out.println(m);
        System.out.println("load 1 kg: " + m.load(1));
        System.out.println("finish:    " + m.finish() + " kg washed");
        System.out.println(m);
    }
}
```

Kode 1.16 `washing_machine.py`: kelas yang sama dalam Python; blok di bawah `if __name__ == "__main__":` hanya berjalan ketika berkas itu dijalankan langsung, bukan ketika diimpor.

```
class WashingMachine:
    def __init__(self, name, capacity):
        self._name = name
        self._capacity = capacity     # kilogram
        self._load_kg = 0             # kilogram yang ada di dalam saat ini
        self._running = False

    def load(self, kg):
        if self._running or kg <= 0 or self._load_kg + kg > self._capacity:
            return False              # tolak: sedang berputar, tidak masuk akal, atau
terlalu berat
        self._load_kg = self._load_kg + kg
        return True

    def start(self):
        if self._running or self._load_kg == 0:
            return False              # tidak ada yang dicuci, atau sudah mencuci
        self._running = True
        return True

    def finish(self):
        if not self._running:
            return 0
        washed = self._load_kg
        self._running = False
        self._load_kg = 0
        return washed                 # kilogram yang keluar dalam keadaan bersih

    def __str__(self):
        return (self._name + " [" + str(self._load_kg) + "/" + str(self._capacity)
                + " kg, " + ("running" if self._running else "idle") + "]")


if __name__ == "__main__":
    m = WashingMachine("Machine A", 8)
    print(m)
    print("load 5 kg: " + str(m.load(5)))
    print("load 4 kg: " + str(m.load(4)))
    print("load 3 kg: " + str(m.load(3)))
    print("start:     " + str(m.start()))
    print(m)
    print("load 1 kg: " + str(m.load(1)))
    print("finish:    " + str(m.finish()) + " kg washed")
    print(m)
```

Versi Java mencetak:

```
Machine A [0/8 kg, idle]
load 5 kg: true
load 4 kg: false
load 3 kg: true
start:     true
Machine A [8/8 kg, running]
load 1 kg: false
finish:    8 kg washed
Machine A [0/8 kg, idle]
```

dan versi Python mencetak hal yang sama dengan `True` dan `False` berhuruf kapital. Telusurilah penolakan-penolakannya: 5 kilogram muat ke dalam mesin 8 kilogram yang kosong; 4 kilogram lagi akan menjadi $9 > 8$ dan ditolak; 3 kilogram lagi menjadi tepat 8 dan diterima; sesudah `start()`, memuat bahkan 1 kilogram pun ditolak karena mesin sedang berputar; `finish()` mengembalikan 8 kilogram yang dicuci dan meninggalkan mesin dalam keadaan kosong dan diam. Jawaban yang lengkap juga menguji `start()` pada mesin kosong dan `load(0)`; keduanya mengembalikan *false*.

Penyelesaian 1.10. (a) Jika `status` bersifat publik, baris mana pun dari program mana pun dapat menulis teks apa pun ke dalamnya — `"PICKED UP"` untuk cucian yang masih di mesin, atau salah eja seperti `"REDAY"` — dan aturan hanya-maju pada **Gambar 1.12** akan sama tidak terlindunginya seperti pada **Kode 1.1**. (b) `final` berarti "diberi nilai sekali dan tidak pernah lagi", padahal seluruh tujuan `status` adalah berubah seiring cucian bergerak melewati kios, sehingga ia tidak dapat dibuat `final`. (c) `final` pada `kilograms` menjamin bahwa bahkan metode-metode kelas itu *sendiri* tidak dapat mengubah berat setelah konstruktor mengaturnya, sehingga harga yang dihitung di meja layanan tidak pernah dapat berubah diam-diam di kemudian hari.

Penyelesaian 1.11. Pada versi prosedural: tambahkan larik kelima `phone` di `main`, dengan satu entri per pesanan pada posisi yang sama dengan larik-larik lain; tambahkan parameter `phone` pada `printAll` dan teruskan larik baru itu pada *kedua* pemanggilannya; dan ubah baris pencetakannya. Setiap fungsi di masa depan yang menangani pesanan juga harus menerima larik kelima itu. Pada versi berorientasi objek: tambahkan satu *field* `private final String phone;` dan satu parameter konstruktor pada `Order`, beri nilai di konstruktor, dan tambahkan ke `toString()` jika perlu dicetak; di `OrderDemo`, tambahkan nomor telepon pada kedua pemanggilan `new Order(...)`. Metode seperti `price()` dan `pickUp()` sama sekali tidak berubah. Versi prosedural memerlukan perubahan di mana pun pesanan *diteruskan*; versi objek memerlukan perubahan hanya di tempat pesanan *didefinisikan* dan *diciptakan*, karena sebuah pesanan berpindah sebagai satu objek dan membawa *field* barunya bersamanya.

Penyelesaian 1.12. (a) `9 / 8` bernilai `1.125`: di Python, `/` selalu menghasilkan bilangan berpecahan. `9 // 8` bernilai `1`: pembagian bilangan bulat, yang membuang sisanya. `int(9 / 8)` juga bernilai `1`, karena `int` memotong pecahannya. Jadi versi **Kode 1.6** yang memakai `int(orders[i] / capacity)` akan mencetak `order 2: 9 kg -> 1 load(s)` — salah, karena 9 kilogram tidak muat dalam satu mesin 8 kilogram — dan total hari itu akan terlalu kecil. Ekspresi yang benar adalah `(orders[i] + capacity - 1) // capacity`, yang menghasilkan 2. (b) Untuk $c = 1$ ruas kanan Persamaan (1.4) adalah $\lfloor (w + 1 - 1)/1 \rfloor = \lfloor w \rfloor = w$, karena w bilangan bulat, dan ruas kirinya adalah $\lceil w/1 \rceil = \lceil w \rceil = w$ dengan alasan yang sama. Kedua ruas sama untuk setiap bilangan bulat $w \geq 0$. ■

Daftar Pustaka

[1] G. Booch, R. A. Maksimchuk, M. W. Engle, B. J. Young, J. Conallen, and K. A. Houston, *Object-Oriented Analysis and Design with Applications*, 3rd ed. Upper Saddle River, NJ, USA: Addison-Wesley, 2007.

[2] A. C. Kay, "The early history of Smalltalk," *ACM SIGPLAN Notices*, vol. 28, no. 3, pp. 69–95, Mar. 1993.

[3] J. Gosling, B. Joy, G. Steele, G. Bracha, A. Buckley, D. Smith, and G. Bierman, *The Java Language Specification, Java SE 25 Edition*. Redwood City, CA, USA: Oracle America, 2025. [Online]. Available: https://docs.oracle.com/javase/specs/

[4] Oracle, "Oracle Java SE support roadmap." Accessed: Sep. 23, 2026. [Online]. Available: https://www.oracle.com/java/technologies/java-se-support-roadmap.html

[5] Oracle, "The arrival of Java 27!," *Inside.java*, Sep. 15, 2026. [Online]. Available: https://inside.java/2026/09/15/jdk-27-available/

[6] G. van Rossum, "A brief timeline of Python," *The History of Python*, Jan. 2009. [Online]. Available: https://python-history.blogspot.com/2009/01/brief-timeline-of-python.html

[7] H. van Kemenade, "PEP 745 – Python 3.14 release schedule," Python Enhancement Proposals, 2024. [Online]. Available: https://peps.python.org/pep-0745/

[8] Python Software Foundation, *The Python Language Reference*, release 3.14, 2025. [Online]. Available: https://docs.python.org/3.14/reference/

[9] Python Software Foundation, "Python 3.14.7," Aug. 5, 2026. [Online]. Available: https://www.python.org/downloads/latest/python3.14/

[10] Stack Overflow, "2025 Developer Survey: Technology — most popular technologies, programming, scripting, and markup languages." Accessed: Sep. 24, 2026. [Online]. Available: https://survey.stackoverflow.co/2025/technology

[11] P. Carbonnelle, "PYPL PopularitY of Programming Language index." Accessed: Sep. 24, 2026. [Online]. Available: https://pypl.github.io/PYPL.html

[12] StatisticsTimes, "Top computer languages 2026" (PYPL index, worldwide, April 2026). Accessed: Sep. 24, 2026. [Online]. Available: https://statisticstimes.com/tech/top-computer-languages.php

[13] TIOBE Software, "TIOBE index for September 2026." Accessed: Sep. 24, 2026. [Online]. Available: https://www.tiobe.com/tiobe-index/

[14] TechRepublic, "TIOBE Index for September 2026: top 10 most popular programming languages," Sep. 2026. Accessed: Sep. 24, 2026. [Online]. Available: https://www.techrepublic.com/article/news-tiobe-index-language-rankings/

[15] GitHub, "Octoverse: A new developer joins GitHub every second as AI leads TypeScript to #1," *The GitHub Blog*, Oct. 2025. Accessed: Sep. 24, 2026. [Online]. Available: https://github.blog/news-insights/octoverse/octoverse-a-new-developer-joins-github-every-second-as-ai-leads-typescript-to-1/

[16] Eclipse Adoptium, "Eclipse Temurin." Accessed: Sep. 23, 2026. [Online]. Available: https://adoptium.net/

[17] Eclipse Foundation, "Eclipse IDE." Accessed: Sep. 23, 2026. [Online]. Available: https://www.eclipse.org/

[18] Apache Software Foundation, "Apache NetBeans." Accessed: Sep. 23, 2026. [Online]. Available: https://netbeans.apache.org/

[19] Microsoft, "Getting started with Java in VS Code." Accessed: Sep. 23, 2026. [Online]. Available: https://code.visualstudio.com/docs/java/java-tutorial

[20] Microsoft, "Getting started with Python in VS Code." Accessed: Sep. 23, 2026. [Online]. Available: https://code.visualstudio.com/docs/python/python-tutorial

[21] Eclipse Adoptium, "Eclipse Temurin 8u502, 11.0.32, 17.0.20, 21.0.12, 25.0.4 and 26.0.2 available," Aug. 2026. [Online]. Available: https://adoptium.net/news/2026/08/eclipse-temurin-8u502-11032-17020-21012-2504-2602-available

[22] J. Gibbons, "JEP 330: Launch single-file source-code programs," OpenJDK, 2018. [Online]. Available: https://openjdk.org/jeps/330

[23] R. Pressler, J. Gibbons, and G. Bierman, "JEP 512: Compact source files and instance main methods," OpenJDK, 2025. [Online]. Available: https://openjdk.org/jeps/512

[24] P. Deitel and H. Deitel, *Java How to Program, Early Objects*, 10th ed. Boston, MA, USA: Pearson, 2015.

[25] M. Lutz, *Learning Python*, 5th ed. Sebastopol, CA, USA: O'Reilly Media, 2013.

[26] Republic of Indonesia, *Law Number 27 of 2022 on Personal Data Protection* (UU PDP), Oct. 2022.

Bab 2

Lingkungan Pengembangan: Menyunting, Menjalankan, dan Men-debug

Pada akhir bab ini Anda akan sanggup menjelaskan, dengan kata-kata sederhana, apa itu *integrated development environment* (lingkungan pengembangan terpadu) dan apa yang dikerjakannya untuk Anda; menyatakan dengan tepat apa yang terjadi sejak Anda menekan *Run* sampai program Anda mencetak baris pertamanya, baik di Java (kompilator, *bytecode*, Java Virtual Machine) maupun di Python (interpreter, *bytecode*, mesin virtual Python); menyiapkan dan menguasai dua meja kerja profesional, Eclipse dan Visual Studio Code, berdampingan; menata kode ke dalam proyek, paket, dan modul serta menghubungkannya dengan *import*; memakai *content assist*, *quick fix*, dan *refactoring* agar alat yang mengerjakan bagian-bagian yang membosankan; dan, yang terpenting, *men-debug*: menghentikan program yang sedang berjalan pada baris pilihan Anda, melangkahinya satu pernyataan demi satu pernyataan, mengamati variabelnya berubah, membaca tumpukan pemanggilannya, memasang *breakpoint* yang hanya aktif pada kondisi yang Anda tulis sendiri, serta membaca *stack trace* dari baris teratas sampai terbawah. Semua ini akan Anda kerjakan pada bug sungguhan dalam program sungguhan, yaitu kalkulator harga Profitina Laundry, dalam Java dan Python, dan Anda akan melihat bug itu ditemukan langkah demi langkah, bukan ditebak.

2.1 Di Mana Kita Berhenti

Bab 1 telah menaruh sebuah meja kerja di hadapan Anda. Anda memasang Java Development Kit (Eclipse Temurin, Java 25) dan Python 3.14, menyiapkan Visual Studio Code dengan Extension Pack for Java dan ekstensi Python, menjalankan program pertama dalam kedua bahasa, dan pada Subbab 1.7.7 untuk pertama kalinya mengamati sebuah gelung berjalan di dalam *debugger*. Anda juga menulis kelas pertama, `Order` untuk pesanan cucian, dan berkenalan dengan gagasan inti buku ini: program adalah sebuah komunitas *objek*, yang masing-masing menyimpan datanya sendiri dan menawarkan perilakunya sendiri.

Bab itu sengaja bergerak cepat. Bab ini memperlambat langkah dan menengok meja kerja itu sendiri, karena Anda akan menghabiskan ratusan jam di sana. Tukang kayu yang tidak tahu fungsi masing-masing pahatnya tetap bisa membuat meja, tetapi lambat dan penuh serpihan.

Tiga pertanyaan menata bab ini.

1. **Apa yang sebenarnya terjadi ketika saya menjalankan program?** Subbab 2.3 menjawabnya dengan gagasan *kompilasi*, *interpretasi*, dan *mesin virtual*, sehingga setiap pesan galat mendapat tempat dalam cerita.
2. **Bagaimana saya mengendalikan alat-alat saya dengan baik?** Subbab 2.4 sampai 2.9 menelusuri Eclipse dan VS Code secara berdampingan. Gagasannya sama di keduanya; hanya letak tombolnya yang berpindah.

3. **Ketika program salah, bagaimana saya mengetahui sebabnya?** Subbab 2.10 sampai 2.13 mengajarkan *debugging* dari nol dan memakainya pada tagihan cucian yang terlalu besar, pendapatan harian yang terlalu kecil, dan program yang mogok.

> **WAWASAN KUNCI:** Anda tidak akan mengingat setiap jalur menu dalam bab ini, dan memang tidak perlu. Yang perlu Anda simpan adalah *kosakatanya* (*workspace*, proyek, paket, *perspective*, *breakpoint*, *step over*, *step into*, tumpukan pemanggilan) dan *kebiasaan* meraih *debugger* alih-alih menebak. Menu dapat dicari; kebiasaan tidak.

2.2 Apa Itu IDE, dan Mengapa Anda Memerlukannya

2.2.1 Meja kerja, bukan kotak ajaib

Anda dapat menulis program Java atau Python hanya dengan penyunting teks biasa seperti Notepad, lalu mengompilasi dan menjalankannya dari terminal. Banyak pemrogram berpengalaman masih melakukannya untuk skrip kecil, dan di Subbab 2.3 kita sengaja melakukannya sendiri agar Anda melihat apa yang dikerjakan alat-alat itu di balik layar.

Namun, untuk program yang lebih dari beberapa puluh baris, cara ini melelahkan. Anda harus mengingat setiap nama metode di setiap berkas, mengetik perintah panjang untuk mengompilasi dan menjalankan, menghitung sampai baris 47 ketika kompilator mengeluhkan baris 47, dan menaburkan pernyataan cetak di mana-mana setiap kali program mencetak angka yang salah.

Integrated development environment, atau **IDE**, menaruh semua kegiatan itu dalam satu jendela dan menghubungkannya satu sama lain. Kata kuncinya adalah *terpadu*. Penyunting mengenal kompilator, sehingga kesalahan Anda digarisbawahi selagi Anda mengetik. Kompilator mengenal penyunting, sehingga mengeklik pesan galat membawa Anda ke barisnya. *Debugger* mengenal keduanya, sehingga ketika program berhenti di sebuah *breakpoint*, penyunting menyorot baris yang tepat dan menampilkan nilai variabel di sampingnya.

> **DEFINISI.** *Integrated development environment* (IDE, lingkungan pengembangan terpadu) adalah program yang menggabungkan, dalam satu jendela, penyunting kode sumber, sarana untuk membangun (mengompilasi) dan menjalankan program, *debugger*, serta alat untuk menjelajahi dan menata ulang kode, yang semuanya berbagi pemahaman yang sama tentang proyek.

IDE bukan sihir. Semua yang dikerjakannya dapat Anda kerjakan sendiri dengan alat-alat terpisah, dan ketika IDE mengejutkan Anda, Anda sebaiknya dapat bertanya, “Alat mana di baliknya yang melakukan ini, dan apa yang harus saya ketik untuk melakukannya sendiri?” Subbab 2.3 menjawabnya untuk urusan menjalankan program, dan Subbab 2.11 untuk *debugging*.

2.2.2 Lima tugas sebuah IDE

Ada gunanya memberi nama pada tugas-tugas sebuah IDE, karena setiap menu di Eclipse dan setiap panel di VS Code termasuk salah satunya. **Tabel 2.1** mendaftarnya.

Tabel 2.1 Lima tugas setiap IDE, dan subbab yang membahas masing-masing.

Tugas	Artinya	Tempat Anda menjumpainya dalam bab ini
Menyunting	Mengetik dan mengubah kode sumber, dengan warna, indentasi, pelengkapan, dan tanda galat seketika	Subbab 2.5, 2.6, dan 2.9
Membangun	Mengubah kode sumber menjadi sesuatu yang dapat dijalankan mesin (mengompilasi, mengemas ke dalam JAR)	Subbab 2.3 dan 2.6
Menjalankan	Memulai program dengan pengaturan yang benar dan menampilkan keluarannya	Subbab 2.6 dan 2.7
Men-debug	Menjeda program, melangkahinya, memeriksa dan mengubah keadaannya	Subbab 2.10 sampai 2.13
Menjelajah dan menata ulang	Menemukan tempat sesuatu didefinisikan, mengganti namanya di mana-mana, merapikan *import*	Subbab 2.8 dan 2.9

2.2.3 Eclipse dan VS Code: dua meja kerja, satu perangkat gagasan

Buku ini memakai dua alat, dan alasannya perlu disampaikan.

Eclipse adalah IDE gratis bersumber terbuka yang dikelola Eclipse Foundation [1]. Eclipse berjalan di Windows, macOS, dan Linux, dan sudah lebih dari dua puluh tahun dipakai untuk mengembangkan segala jenis aplikasi Java, termasuk (selama bertahun-tahun) aplikasi Android. Eclipse adalah IDE *lengkap* dengan pengetahuan bawaan yang mendalam tentang Java [2].

Visual Studio Code (VS Code) adalah *penyunting* gratis bersumber terbuka dari Microsoft yang menjadi IDE untuk suatu bahasa begitu ekstensi bahasa itu dipasang [3], [4]. Dengan Extension Pack for Java, VS Code memahami proyek Java; dengan ekstensi Python, ia memahami Python. VS Code lebih ringan daripada Eclipse, memperlakukan Java dan Python dengan cara yang sama, dan sudah dipasang di Bab 1.

Karena buku ini mengajarkan setiap gagasan dalam Java *dan* Python, sebagian besar waktu Anda akan memakai VS Code untuk kedua bahasa. Walaupun begitu, Eclipse tetap kita pelajari dengan cermat: kosakatanya (*workspace*, *perspective*, *view*, Package Explorer) adalah kosakata klasik perkakas Java yang dipinjam IDE lain seperti IntelliJ IDEA, dan Eclipse memperlihatkan beberapa gagasan yang disembunyikan VS Code, misalnya perbedaan antara *view* dan *editor*.

Tabel 2.2 adalah kamus yang akan terus Anda buka kembali. Jangan mencoba menghafalnya sekarang; setiap barisnya dijelaskan pada subbab-subbab berikutnya.

Tabel 2.2 Gagasan yang sama dengan dua nama: kamus antara Eclipse dan VS Code.

Gagasan	Eclipse	VS Code
Tempat pengaturan dan proyek Anda	*Workspace* (folder yang dipilih saat Eclipse dimulai)	Folder yang Anda buka (*File > Open Folder*), bisa juga berkas *multi-root* `.code-workspace`
Satuan kode yang dibangun bersama	Proyek Java (dengan berkas `.project`)	Folder yang dibuka; proyek Java dikenali otomatis dan tampil di *view Java Projects*
Menjelajahi kode	*View* Package Explorer atau Project Explorer	*View* Explorer dan *view* Java Projects
Susunan panel untuk suatu tugas	*Perspective* (Java, Debug, ...)	Satu tata letak; *view* Run and Debug muncul ketika men-debug
Java atau Python mana yang menjalankan program	Installed JREs (*Window > Preferences > Java > Installed JREs*)	*Runtime* Java diatur lewat *Java: Configure Java Runtime*; interpreter Python dipilih lewat *Python: Select Interpreter*
Cara menjalankan program yang disimpan	*Launch configuration* (*Run > Run Configurations...*)	`launch.json` di folder `.vscode`
Daftar galat	*View* Problems	Panel Problems
Pelengkapan saat mengetik	Content Assist (Ctrl+Space)	IntelliSense (Ctrl+Space)
Memperbaiki galat yang ditandai	Quick Fix (Ctrl+1)	Quick Fix (Ctrl+.)

2.3 Bagaimana Program Berjalan: Kompilasi, Interpretasi, dan Mesin Virtual

Subbab 1.2.3 telah membuat sketsa tentang bagaimana kode sumber menjadi program yang berjalan. Sebelum membuka IDE mana pun, kita melihatnya lebih dekat, sebab tanpa gambaran ini IDE hanyalah kotak bertombol hijau, sedangkan dengan gambaran ini setiap pesan yang ditampilkan IDE mendapat tempat dalam cerita.

2.3.1 Prosesor hanya menguasai satu bahasa

Prosesor di dalam komputer Anda, yaitu *CPU*, hanya memahami satu jenis instruksi: **kode mesin**, deretan bilangan yang masing-masing berarti sesuatu yang sangat kecil, misalnya "jumlahkan dua register ini" atau "lompat ke alamat ini". Keluarga prosesor yang berbeda berbicara dengan kode mesin yang berbeda: *x86-64* untuk Intel dan AMD, *ARM64* untuk Mac terbaru dan sebagian besar ponsel, dan program untuk yang satu tidak berjalan di yang lain.

Tidak ada lagi yang menulis program dalam kode mesin. Kita menulis dalam *bahasa tingkat tinggi* seperti Java dan Python, yang terbaca hampir seperti bahasa Inggris dan matematika:

```
total = total + line
```

Sesuatu harus menerjemahkan baris itu menjadi kode mesin sebelum CPU dapat berbuat apa pun dengannya. Ada dua cara klasik untuk menerjemahkan, dan Java maupun Python memakai campuran keduanya.

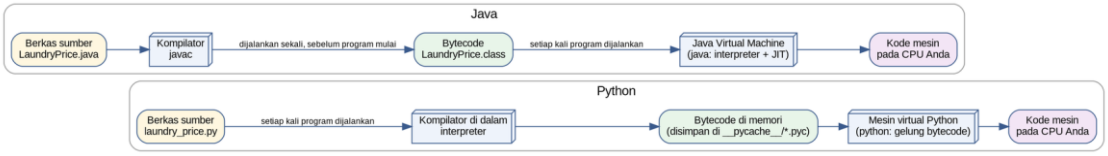
2.3.2 Kompilator dan interpreter, serta kapan masing-masing mengeluh

Subbab 1.2.3 mengibaratkan **kompilator** (*compiler*) sebagai penerjemah yang menerjemahkan seluruh buku sekali lalu mencetaknya, dan **interpreter** sebagai penerjemah yang berdiri di samping Anda dan menerjemahkan secara lisan sambil jalan. Perumpamaan itu membawa akibat yang terasa setiap hari: penerjemah buku menemukan kalimat yang tidak masuk akal *sebelum* ada

yang membaca bukunya, sedangkan penerjemah lisan baru menemukannya ketika pembaca sampai di kalimat itu, mungkin di tengah jalan.

> **DEFINISI.** *Kompilator* menerjemahkan seluruh program ke bentuk lain sebelum program dijalankan. *Interpreter* melaksanakan instruksi program satu per satu selagi program berjalan. *Galat waktu kompilasi* (*compile-time error*) ditemukan kompilator sebelum program dimulai; *galat waktu jalan* (*run-time error*) terjadi selagi program berjalan.

Bahasa nyata mencampur kedua gagasan itu. Java dan Python sama-sama mula-mula *mengompilasi* kode sumber Anda ke bahasa antara yang disebut **bytecode**, lalu sebuah **mesin virtual** *menginterpretasi* bytecode tersebut. Perbedaannya terletak pada *kapan* setiap langkah terjadi dan siapa yang mengerjakannya, dan **Gambar 2.1** memperlihatkannya sekilas.



Gambar 2.1 Dari kode sumber ke prosesor. Java mengompilasi sekali dengan javac dan JVM menjalankan bytecode-nya setiap kali; Python mengompilasi di dalam interpreter setiap kali program dimulai, lalu mesin virtualnya menjalankan bytecode itu.

2.3.3 Java: javac, bytecode, dan Java Virtual Machine

Di Java, penerjemahan terjadi dalam dua langkah terpisah yang dikerjakan dua program terpisah, keduanya bagian dari Java Development Kit (JDK).

Langkah 1: kompilasi dengan `javac`. Kompilator Java, `javac`, membaca berkas `.java` Anda, memeriksa bahwa berkas itu mematuhi setiap aturan bahasa Java (setiap pernyataan diakhiri titik koma, setiap variabel memiliki tipe yang dideklarasikan, setiap metode yang Anda panggil memang ada), lalu menulis berkas `.class`. Berkas `.class` tidak berisi kode mesin untuk CPU Anda. Isinya adalah **Java bytecode**, himpunan instruksi sebuah komputer khayalan bernama **Java Virtual Machine** (JVM), yang didefinisikan secara persis dalam spesifikasi JVM [5].

Langkah 2: jalankan dengan `java`. Peluncur `java` memulai sebuah JVM, yang memuat berkas `.class` dan mengeksekusi bytecode-nya. JVM sendiri adalah program sungguhan yang ditulis untuk CPU Anda yang sungguhan. Mula-mula JVM *menginterpretasi* bytecode, satu instruksi demi satu instruksi. Sambil itu ia menghitung seberapa sering setiap metode dijalankan; metode yang sering dijalankan (metode "panas") diserahkan kepada **just-in-time compiler** (JIT) yang menerjemahkannya menjadi kode mesin sungguhan untuk CPU Anda, sehingga sejak saat itu metode tersebut berjalan dengan kecepatan penuh.

Mengapa dua langkah? Karena berkas `.class` yang sama berjalan di setiap mesin yang memiliki JVM, Windows ataupun Linux, Intel ataupun ARM: "tulis sekali, jalankan di mana saja" (*write once, run anywhere*).

Mari kita amati kedua langkah itu pada program menarik yang paling kecil. **Kode 2.1** mendefinisikan metode yang menjumlahkan dua bilangan dan metode `main` yang memakainya.

Kode 2.1 `Add.java`: metode dua baris, cukup kecil untuk dibaca bytecode-nya.

```java
public class Add {

    static int add(int a, int b) {
        return a + b;
    }

    public static void main(String[] args) {
        System.out.println("2 + 3 = " + add(2, 3));
    }
}
```

Untuk mengompilasi dan menjalankannya di terminal (di Windows gunakan *Command Prompt* atau *PowerShell*; di macOS dan Linux gunakan *Terminal*), masuklah ke folder tempat berkas itu berada dan ketik:

```
javac Add.java
java Add
```

Perintah pertama menghasilkan `Add.class` dan tidak mencetak apa pun jika semuanya beres. Perintah kedua mencetak:

```
2 + 3 = 5
```

Sekarang bagian yang biasanya tersembunyi. JDK menyertakan pembongkar (*disassembler*) bernama `javap` yang mencetak bytecode di dalam berkas `.class` dalam bentuk yang dapat dibaca [6]. Mengetik `javap -c Add` menampilkan:

```
Compiled from "Add.java"
public class Add {
  public Add();
    Code:
       0: aload_0
       1: invokespecial #1                  // Method java/lang/Object."<init>":()V
       4: return

  static int add(int, int);
    Code:
       0: iload_0
       1: iload_1
       2: iadd
       3: ireturn

  public static void main(java.lang.String[]);
    Code:
       0: getstatic     #7                  // Field
java/lang/System.out:Ljava/io/PrintStream;
       3: iconst_2
       4: iconst_3
       5: invokestatic  #13                 // Method add:(II)I
       8: invokedynamic #19,  0             // InvokeDynamic
#0:makeConcatWithConstants:(I)Ljava/lang/String;
      13: invokevirtual #23                 // Method
java/io/PrintStream.println:(Ljava/lang/String;)V
      16: return
}
```

Bacalah metode `add` lebih dahulu. Keempat instruksi bytecode-nya berkata: `iload_0`, dorong parameter `int` pertama (`a`) ke sebuah tumpukan kerja kecil; `iload_1`, dorong yang kedua (`b`); `iadd`, ambil keduanya, jumlahkan, dorong hasilnya; `ireturn`, kembalikan `int` di puncak tumpukan. Itulah persis `return a + b;`, dieja untuk mesin yang bekerja dengan tumpukan nilai. Di `main`, `iconst_2` dan `iconst_3` mendorong konstanta 2 dan 3, `invokestatic` memanggil `add`, dan instruksi sisanya membangun string `"2 + 3 = 5"` lalu memanggil `println`. Anda juga melihat

metode `Add()` yang tidak pernah Anda tulis: itulah *konstruktor bawaan* (*default constructor*) yang ditambahkan kompilator ke setiap kelas yang tidak memilikinya, dan Bab 3 menjelaskannya.

Anda jarang akan membaca bytecode, tetapi satu fakta penting: **JVM tidak menjalankan berkas `.java` Anda, melainkan berkas `.class` yang dibuat `javac` darinya.** Menyunting kode sumber lalu menjalankan lagi tanpa mengompilasi ulang berarti menjalankan program yang *lama*; IDE menyembunyikan hal ini dengan mengompilasi ulang setiap kali Anda menyimpan, terminal tidak.

> **PERINGATAN:** Sejak Java 11 Anda juga boleh mengetik `java Add.java` (dengan ekstensi `.java`). *Mode berkas sumber* (*source-file mode*) ini mengompilasi berkas di memori lalu langsung menjalankannya, praktis untuk program satu berkas seperti sebagian besar kode dalam buku ini. Mode ini tidak meninggalkan berkas `.class`, dan hanya berlaku jika seluruh program berada dalam satu berkas itu.

2.3.4 Python: interpreter, kompilator tersembunyinya, dan `__pycache__`

Di Python, kedua langkah terjadi di dalam satu program, yaitu interpreter Python, yang Anda mulai dengan mengetik `python` (di Windows sering `py`) atau `python3` (di macOS dan Linux). Ketika Anda mengetik

```
python3 add_bytecode.py
```

interpreter membaca seluruh berkas, *mengompilasinya* menjadi bytecode Python di memori, lalu mesin virtualnya, yaitu sebuah gelung di dalam interpreter yang mengambil dan mengeksekusi instruksi bytecode satu demi satu, menjalankan bytecode itu [7]. Tidak ada perintah kompilator tersendiri, dan (biasanya) tidak ada berkas bytecode yang tertinggal di cakram.

Python memungkinkan Anda melihat bytecode-nya dari dalam program, dengan modul standar `dis` (singkatan *disassembler*). **Kode 2.2** adalah kembaran **Kode 2.1**: program ini menjalankan fungsi kecil `add` lalu membongkarnya.

Kode 2.2 `add_bytecode.py`: menjalankan sebuah fungsi, lalu mencetak bytecode yang dibuat Python untuknya.

```
"""add_bytecode.py: menjalankan fungsi kecil, lalu menampilkan bytecode buatan Python
untuknya."""
import dis


def add(a, b):
    return a + b


print("2 + 3 =", add(2, 3))
dis.dis(add)
```

Pada Python 3.11 keluarannya:

```
2 + 3 = 5
  5           0 RESUME                   0

  6           2 LOAD_FAST                0 (a)
              4 LOAD_FAST                1 (b)
              6 BINARY_OP                0 (+)
             10 RETURN_VALUE
```

Angka 5 dan 6 adalah nomor baris kode sumber. `RESUME` menandai awal fungsi, kedua instruksi `LOAD_FAST` mendorong `a` dan `b`, `BINARY_OP 0 (+)` menjumlahkannya, dan `RETURN_VALUE`

mengembalikan hasilnya. Kedua mesin virtual itu sangat mirip: keduanya *mesin tumpukan* (*stack machine*) yang memuat, menjumlahkan, dan mengembalikan. Bedanya, JVM mempunyai satu instruksi untuk setiap tipe (`iadd` untuk `int`, `dadd` untuk `double`), sedangkan satu-satunya `BINARY_OP +` milik Python memutuskan pada waktu jalan apa arti "tambah", karena variabel Python tidak memiliki tipe yang dideklarasikan.

> **PERINGATAN:** Bytecode yang persis berbeda antarversi Python; Python 3.12, 3.13, dan 3.14 mencetak nama instruksi dan *offset* yang sedikit berbeda dari keluaran 3.11 di atas, dan sebagian menambahkan bentuk berwarna atau adaptif. Gagasannya (muat, muat, jumlahkan, kembalikan) sama pada semuanya. Bytecode adalah rincian internal interpreter, bukan bagian dari bahasa.

Di mana bytecode disimpan? Untuk berkas yang Anda jalankan langsung, hanya di memori. Namun ketika program Anda *mengimpor* modul lain milik Anda (Subbab 2.8), Python menyimpan bytecode modul itu di folder bernama `__pycache__` di samping kode sumbernya, agar saat program dimulai berikutnya langkah kompilasi dapat dilewati bila kode sumbernya tidak berubah. Anda dapat melihatnya sendiri dengan berkas **Kode 2.4** (Subbab 2.6): mengimpornya dengan perintah satu baris lalu menampilkan isi foldernya memperlihatkan

```
python3 -c "import my_first_module"
```

```
.
./__pycache__
./__pycache__/my_first_module.cpython-311.pyc
./my_first_module.py
```

Nama `cpython-311` mencatat interpreter (CPython, versi 3.11) yang membuatnya. Anda tidak perlu membuka atau menyalin folder `__pycache__`; folder itu aman dihapus.

2.3.5 Kapan kesalahan tertangkap: waktu kompilasi dan waktu jalan

Sekarang kita dapat menjelaskan sesuatu yang membingungkan hampir setiap pemula yang mempelajari kedua bahasa: **Java menolak memulai program yang akan dengan senang hati dimulai oleh Python, lalu mogok.**

Ambil program pertama dari Subbab 2.6 dan hapus satu titik koma, sehingga baris 6 berbunyi `System.out.println("Hello Eclipse!")`. Mengompilasinya menghasilkan:

```
src/myfirstproject/MyFirstClass.java:6: error: ';' expected
        System.out.println("Hello Eclipse!")
                                           ^
1 error
```

Bacalah pesan itu dari kiri ke kanan: *berkas*nya (`src/myfirstproject/MyFirstClass.java`), *nomor baris*nya (6), *jenis* pesannya (`error`), dan *penjelasan*nya (`';' expected`, titik koma diharapkan). Tanda sisipan `^` pada baris ketiga menunjuk kolom tepat tempat kompilator mengharapkan titik koma. Tidak ada berkas `.class` yang ditulis, sehingga tidak ada yang dapat dijalankan: galat itu tertangkap pada *waktu kompilasi*.

Python juga mengompilasi sebelum menjalankan, sehingga galat *sintaksis*, yaitu baris yang tidak mengikuti tata bahasa bahasa pemrograman itu, juga tertangkap sebelum apa pun berjalan. Hapus titik dua setelah `def main()` di **Kode 2.4** lalu jalankan:

```
  File "/home/student/hello_py/my_first_module.py", line 4
    def main()
              ^
SyntaxError: expected ':'
```

Sekali lagi Anda mendapat berkasnya, barisnya (4), tanda sisipan di bawah tempat titik dua seharusnya berada, dan jenis galatnya (`SyntaxError: expected ':'`). Tidak ada yang tercetak, karena program tidak pernah dimulai.

Perbedaannya muncul pada kesalahan yang benar secara tata bahasa tetapi *tidak bermakna*, misalnya nama yang tidak ada. Kompilator Java memeriksa setiap nama dan tipe sebelum program berjalan; kompilator Python hanya memeriksa tata bahasa dan mencari nama ketika setiap baris dijalankan. Subbab 2.8.4 memperlihatkan reaksi keduanya terhadap `import` yang terlupa: `cannot find symbol` pada waktu kompilasi di Java, `NameError` pada waktu jalan di Python. Nama yang salah eja di cabang Python yang jarang dijalankan dapat bersembunyi berbulan-bulan, salah satu alasan mengapa pengujian (Bab 6) bahkan lebih penting di Python.

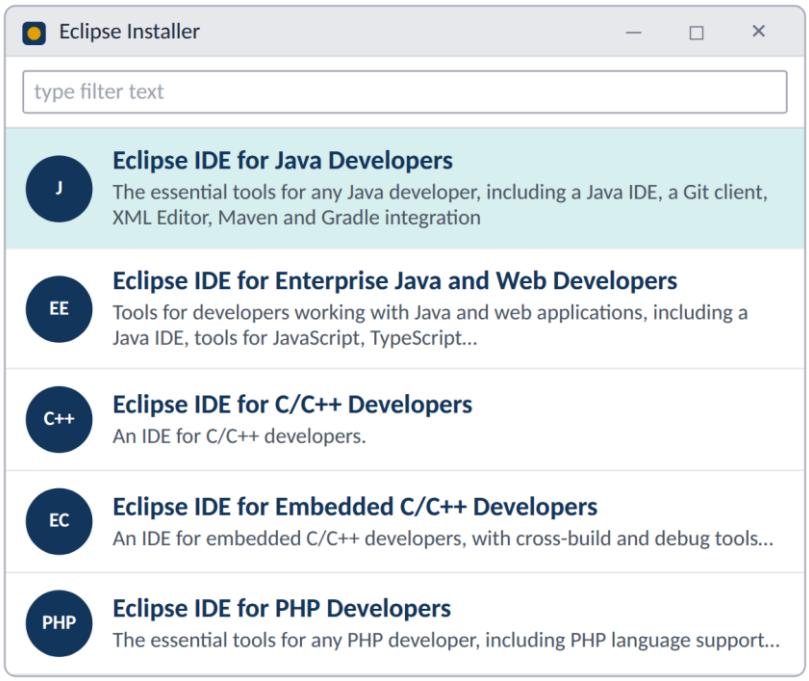
> **WAWASAN KUNCI:** *Java memeriksa nama dan tipe sebelum program berjalan; Python memeriksanya selagi program berjalan.* Keketatan Java menuntut beberapa baris deklarasi lebih banyak dan memberi pesan galat lebih awal. Keluwesan Python menghemat ketikan dan memindahkan sebagian galat ke waktu jalan. Tidak ada yang lebih baik dalam segala keadaan; mengetahui Anda sedang berada di pihak mana memberi tahu Anda di mana mencari kesalahan.

2.4 Memasang dan Memulai Eclipse

2.4.1 Mengunduh dan memasang

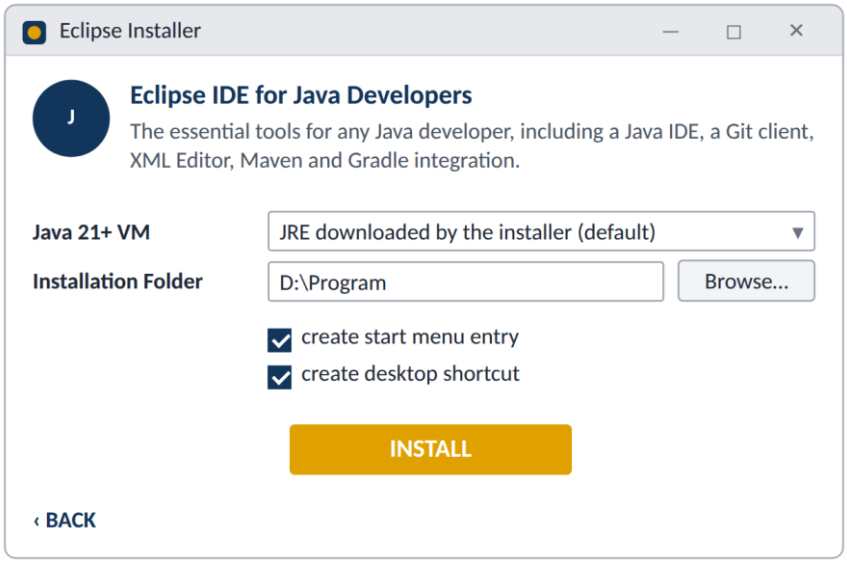
Eclipse diunduh dari halaman unduhan Eclipse Foundation, `https://www.eclipse.org/downloads/` [1]. Halaman itu menawarkan **Eclipse Installer**, sebuah program kecil yang mengunduh dan menyiapkan *paket* Eclipse tertentu yang Anda pilih. Di Windows pemasangnya berupa berkas `.exe`; di macOS ia berupa aplikasi terkemas yang dipasang dengan cara Mac yang biasa; di Linux ia berupa arsip terkompresi yang Anda ekstrak lebih dahulu, lalu Anda jalankan dari folder hasil ekstraksi itu.

Ketika pemasang dimulai, ia menampilkan daftar paket Eclipse, masing-masing menghimpun *plug-in* untuk satu jenis pekerjaan. Pilihlah **Eclipse IDE for Java Developers**, yang berada paling atas pada **Gambar 2.2**. Paket ini berisi penyunting dan kompilator Java, klien Git, serta dukungan untuk alat bangun (*build tool*) Maven dan Gradle; itulah semua yang dibutuhkan buku ini.



Gambar 2.2 Daftar paket pada Eclipse Installer. Eclipse IDE for Java Developers adalah paket yang dipilih.

Layar berikutnya, **Gambar 2.3**, mengajukan dua pertanyaan. *Java 21+ VM* adalah *runtime* Java yang akan menjalankan Eclipse itu sendiri; pemasang menawarkan untuk mengunduhnya, dan pilihan bawaannya sudah memadai. *Installation Folder* adalah folder induk tempat folder `eclipse` akan dibuat; folder mana pun yang dapat Anda tulisi boleh dipakai, misalnya `D:\Program` di Windows atau folder rumah (*home*) Anda di macOS dan Linux. Biarkan kedua kotak centang yang membuat entri di menu Start dan pintasan di desktop tetap tercentang, klik **INSTALL**, setujui lisensinya, lalu tunggu sampai tombol itu berubah menjadi **LAUNCH**.



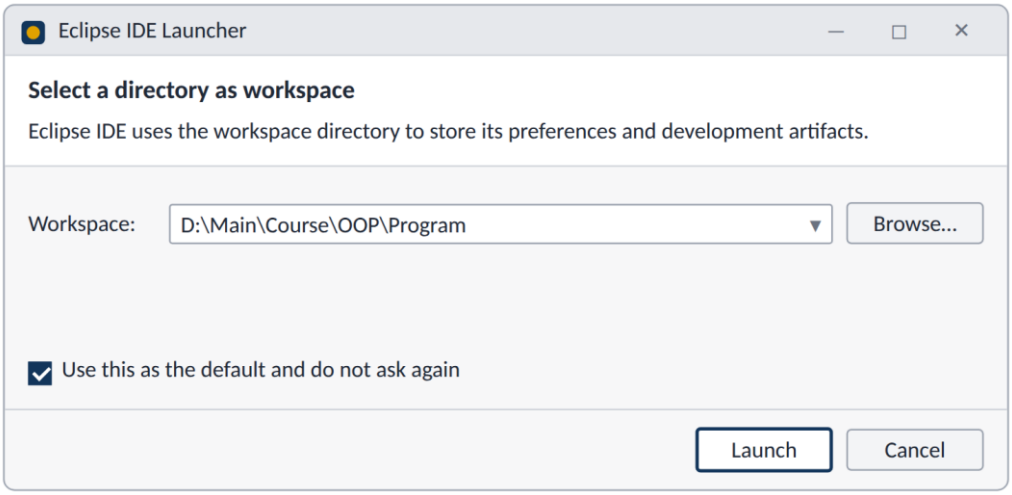
Gambar 2.3 Memilih runtime Java untuk Eclipse dan folder pemasangan, sesaat sebelum menekan INSTALL.

> **PERINGATAN:** Java yang *menjalankan Eclipse* dan Java yang *menjadi sasaran kompilasi program Anda* adalah dua pengaturan yang terpisah. Eclipse boleh saja berjalan di atas *runtime* yang diunduhnya sendiri, sementara proyek-proyek Anda memakai JDK 25 yang Anda pasang di Bab 1. Pengaturan yang kedua ada di *Window > Preferences > Java > Installed JREs*;

bila JDK yang Anda pasang tidak muncul dalam daftar itu, klik *Add...* dan arahkan ke folder JDK tersebut.

2.4.2 *Workspace*

Hal pertama yang ditanyakan Eclipse ketika dimulai adalah di mana *workspace* (ruang kerja) Anda harus berada (**Gambar 2.4**).

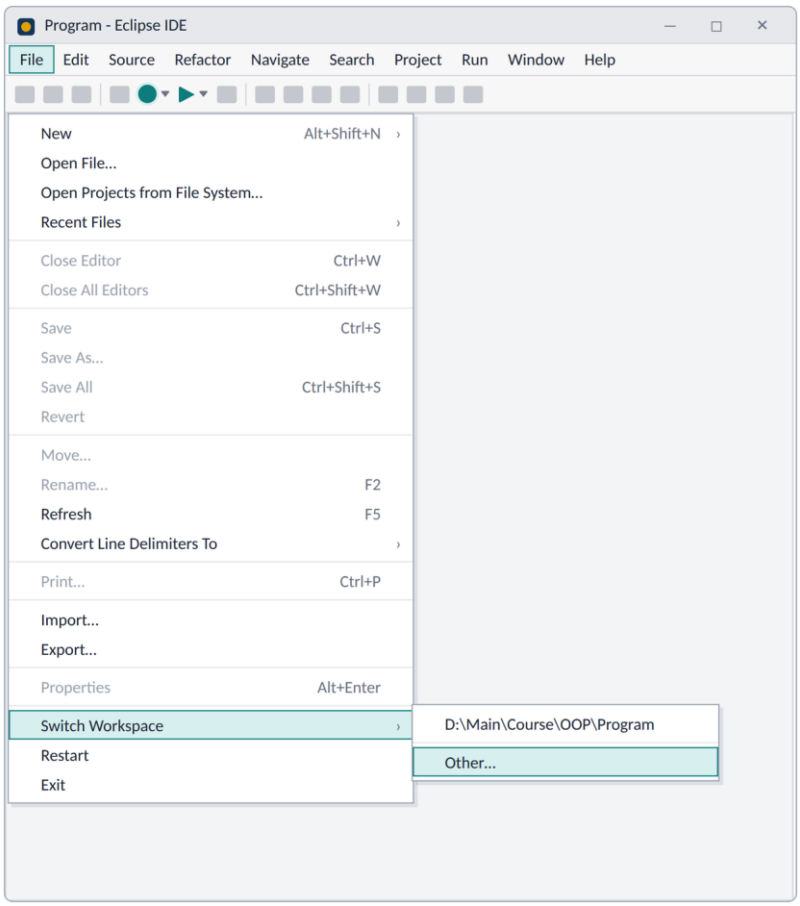


Gambar 2.4 Eclipse IDE Launcher meminta sebuah folder workspace sebelum meja kerja terbuka.

DEFINISI. *Workspace* Eclipse adalah sebuah folder di disk tempat Eclipse menyimpan *metadata*-nya, yaitu preferensi yang Anda atur, keadaan setiap *plug-in*, log, dan daftar proyek Anda, serta tempat Eclipse membuat proyek baru secara bawaan.

Metadata itu tersimpan dalam subfolder tersembunyi, `.metadata`. Proyek-proyek Anda boleh berada di dalam folder *workspace* atau di tempat lain mana pun di disk; proyek yang disimpan dalam repositori Git-nya sendiri, misalnya, biasanya berada di luarnya.

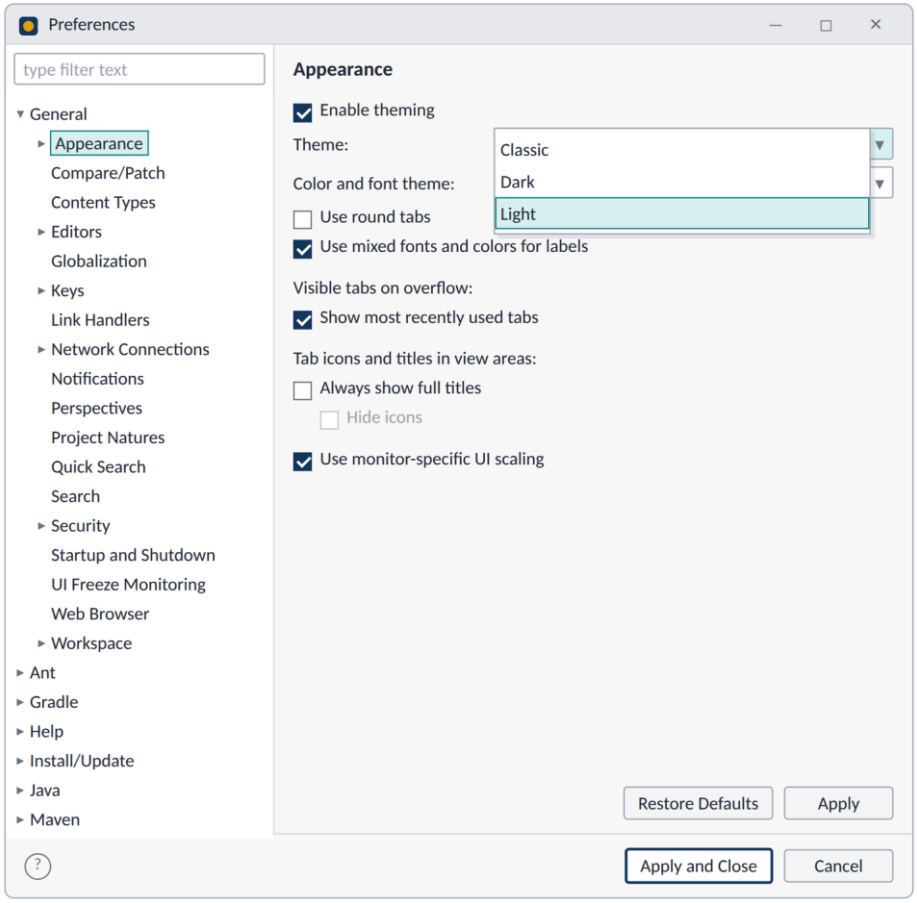
Pilihlah folder kosong, misalnya `D:\Main\Course\OOP\Program` di Windows atau `~/oop/workspace` di macOS dan Linux, centang *Use this as the default and do not ask again* bila Anda mau, lalu klik **Launch**. Menu *File > Switch Workspace > Other...* (**Gambar 2.5**) memindahkan Anda ke *workspace* lain di kemudian hari; satu *workspace* untuk setiap mata ajar menjaga pengaturan dan percobaan tetap terpisah.



Gambar 2.5 File > Switch Workspace > Other… membuka kembali peluncur agar Anda dapat memilih workspace lain.

2.4.3 Tampilan

Secara bawaan Eclipse memakai tema warna terang. Bila Anda lebih suka tampilan gelap atau klasik, buka *Window* > *Preferences* > *General* > *Appearance* (**Gambar 2.6**), pilih sebuah tema, klik *Apply and Close*, lalu mulai ulang Eclipse, karena sebagian gaya bawaan sistem operasi hanya diterapkan saat program dimulai. Di macOS butir *Preferences* terletak di bawah menu *Eclipse*, bukan *Window*; hanya di tempat inilah jalur menu dalam bab ini berbeda antarsistem.



Gambar 2.6 Halaman preferensi Appearance, tempat tema warna dipilih.

2.5 Meja Kerja Eclipse: *View*, Editor, dan *Perspective*

Ketika *workspace* terbuka, Anda melihat **meja kerja** (*workbench*), jendela utama Eclipse. Meja kerja itu tersusun dari dua jenis bagian, *view* dan editor, yang ditata oleh gagasan ketiga, *perspective*. Tiga kata ini menjelaskan hampir semua yang Anda lihat.

2.5.1 *View* dan editor

> **DEFINISI.** *View* adalah panel yang menampilkan informasi terstruktur dan biasanya memungkinkan Anda bertindak langsung terhadapnya, misalnya pohon proyek Anda atau daftar galat. *Editor* adalah panel yang menampilkan isi satu berkas untuk Anda ubah; perubahan di dalam editor tidak ditulis ke disk sampai Anda menyimpannya.

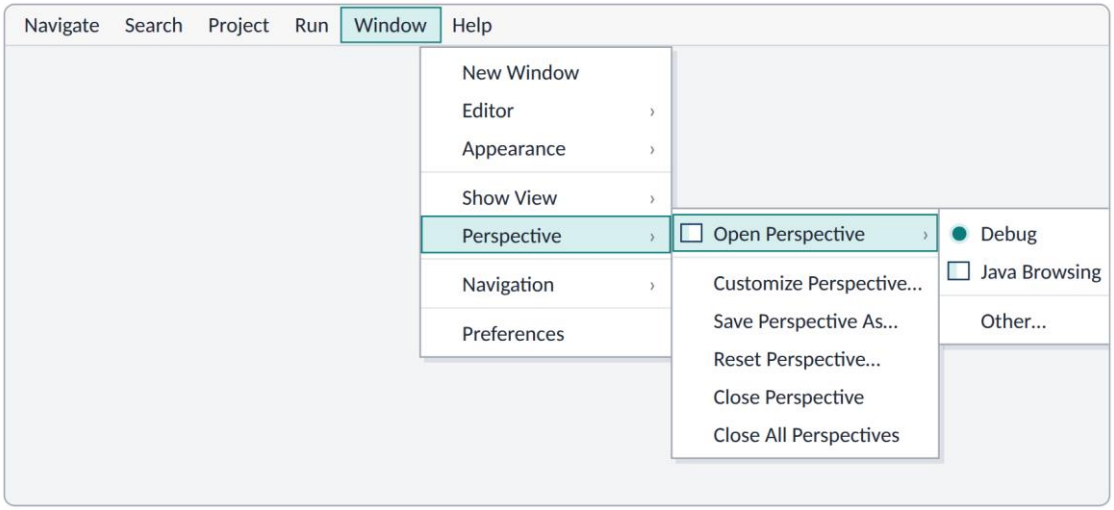
Perbedaan ini penting dalam praktik. Bila Anda mengganti nama sebuah berkas di *view* Package Explorer, berkas di disk langsung berganti nama; tidak ada yang perlu disimpan. Bila Anda mengetik di sebuah *editor*, berkas di disk tidak berubah sampai Anda menekan Ctrl+S (Cmd+S di macOS). Editor yang memuat perubahan belum tersimpan disebut editor *kotor* (*dirty*), dan Eclipse menandainya dengan tanda bintang di depan nama berkas pada tabnya, misalnya `*MyFirstClass.java`. Biasakan melirik tab untuk mencari tanda bintang: menjalankan program ketika editornya masih kotor berarti menjalankan versi yang terakhir *disimpan*, dan keluhan "padahal baris itu sudah saya ubah!" adalah salah satu keluhan pemula yang paling sering terdengar.

Beberapa editor dapat terbuka sekaligus, bertumpuk sebagai tab di bagian tengah jendela; seret sebuah tab untuk melihat dua berkas berdampingan.

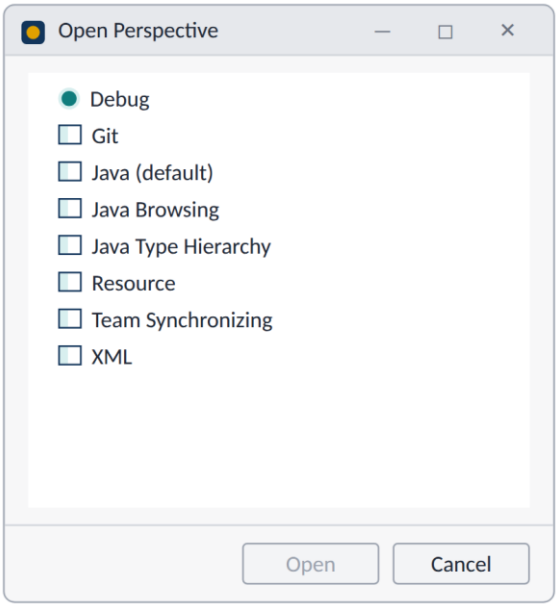
2.5.2 *Perspective*

> **DEFINISI.** *Perspective* adalah susunan *view* dan editor yang diberi nama dan cocok untuk satu jenis tugas, misalnya menulis kode Java (*perspective* Java) atau men-debug (*perspective* Debug).

Hampir sepanjang waktu Anda akan memakai dua *perspective*: **Java** dan **Debug**. Eclipse berpindah sendiri ke *perspective* Debug ketika program berhenti pada sebuah *breakpoint* (Subbab 2.10.3), dan Anda dapat berpindah secara manual melalui *Window > Perspective > Open Perspective*, seperti pada **Gambar 2.7**. Butir *Other...* membuka daftar lengkap *perspective* yang tersedia pada instalasi Anda (**Gambar 2.8**); *perspective* mana yang muncul bergantung pada *plug-in* yang terpasang.



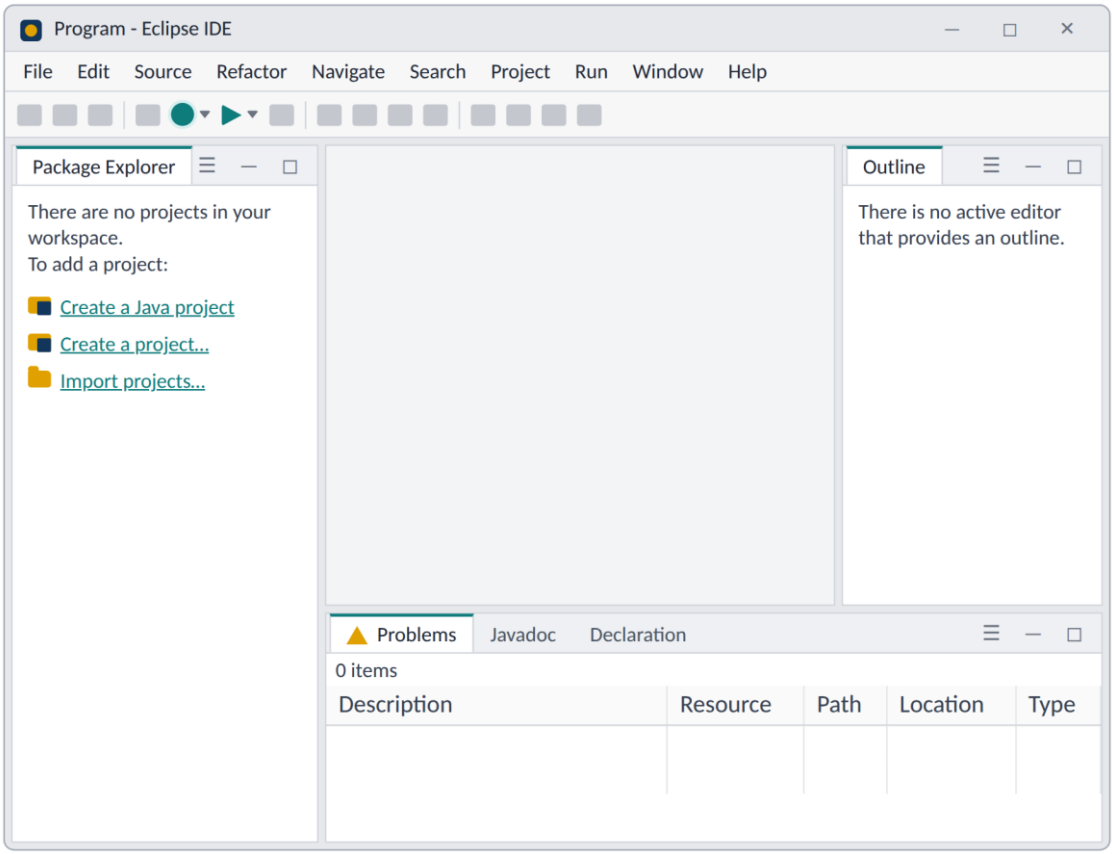
Gambar 2.7 Window > Perspective > Open Perspective, dengan perspective Debug dan Java Browsing ditawarkan langsung dan sisanya di bawah Other....



Gambar 2.8 Dialog Open Perspective mendaftar setiap perspective pada instalasi; Java ditandai sebagai bawaan.

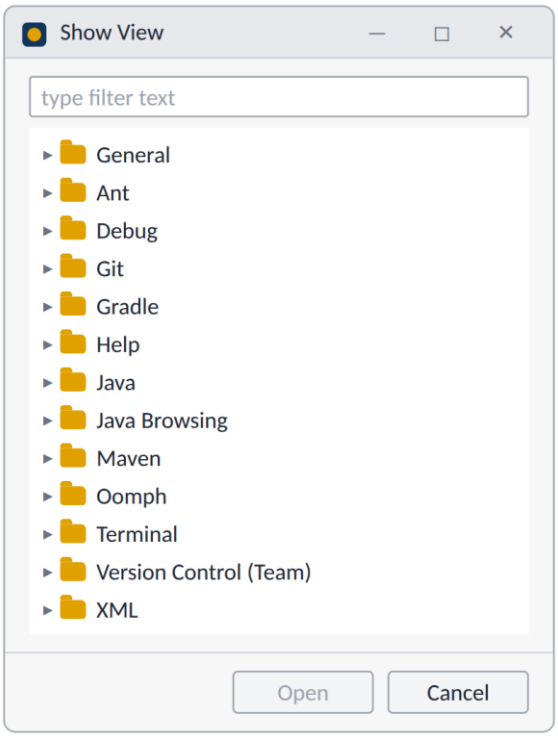
Editor dipakai bersama oleh semua *perspective*: bila `Tester.java` sedang terbuka di *perspective* Java lalu Anda berpindah ke *perspective* Debug, editor yang sama tetap ada di sana. Hanya *view* di sekelilingnya yang berubah.

Gambar 2.9 memperlihatkan *perspective* Java pada *workspace* yang masih baru. Di kiri terdapat **Package Explorer**, pohon proyek-proyek Anda. Area kosong yang lebar di tengah adalah area editor. Di kanan terdapat *view* **Outline**, yang akan menampilkan struktur (*field* dan *method*) dari berkas apa pun yang sedang terbuka. Di bagian bawah berjajar *view* **Problems**, **Javadoc**, dan **Declaration**. Karena *workspace* masih kosong, Package Explorer menawarkan tautan untuk membuat proyek Java atau mengimpornya.



Gambar 2.9 Perspective Java pada workspace kosong: Package Explorer di kiri, area editor di tengah, Outline di kanan, serta Problems, Javadoc, dan Declaration di bawah.

Bila Anda tidak sengaja menutup sebuah *view* atau jendela mulai membingungkan, *Window > Perspective > Reset Perspective...* mengembalikan susunan aslinya, dan *Window > Show View > Other...* (**Gambar 2.10**) menemukan *view* apa pun berdasarkan namanya. *Save Perspective As...* dan *Customize Perspective...* menyimpan susunan buatan Anda sendiri serta menyembunyikan atau menampilkan tombol bilah alat.



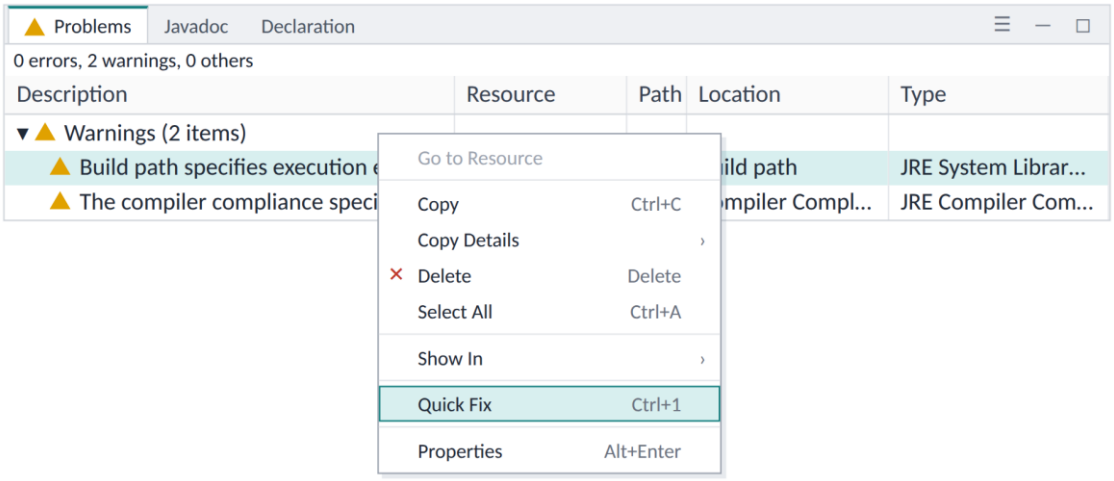
Gambar 2.10 Dialog Show View, dikelompokkan per kategori; ketik sebagian nama view di kotak saring untuk menemukannya.

2.5.3 *View* pada *perspective* Java

Package Explorer. Menjelajahi proyek-proyek Anda, membuka berkas dengan klik ganda, dan mengubah strukturnya (mengganti nama, menghapus, membuat paket dan kelas, menyeret berkas); klik kanan memperlihatkan apa saja yang mungkin dilakukan. *Project Explorer* menampilkan proyek yang sama sebagai folder biasa.

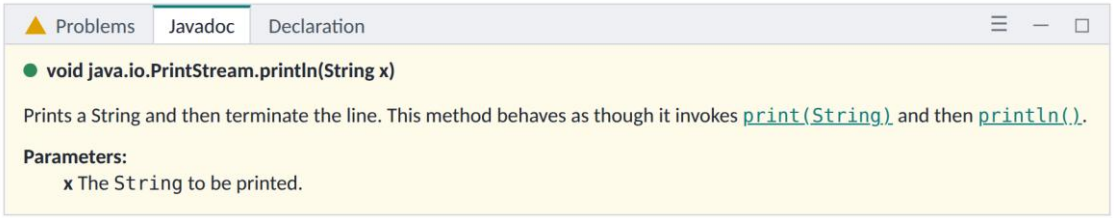
Outline. Menampilkan kelas, *field*, dan *method* dari berkas di editor yang aktif; klik sebuah *method* untuk melompat ke sana.

Problems. *View* Problems mendaftar setiap galat (*error*) dan peringatan (*warning*) di proyek-proyek Anda, masing-masing dengan uraian, berkas, folder, dan barisnya. Klik ganda pada sebuah masalah membuka berkasnya tepat di baris itu. Menu tarik-turunnya memungkinkan Anda menyaring apa yang ditampilkan, misalnya hanya masalah dari proyek yang dipilih, dan klik kanan pada sebuah masalah menawarkan **Quick Fix** (Ctrl+1), perbaikan otomatis yang sama dengan yang kita jumpai di editor pada Subbab 2.9. **Gambar 2.11** memperlihatkan Quick Fix yang ditawarkan untuk sebuah peringatan tentang versi Java proyek.



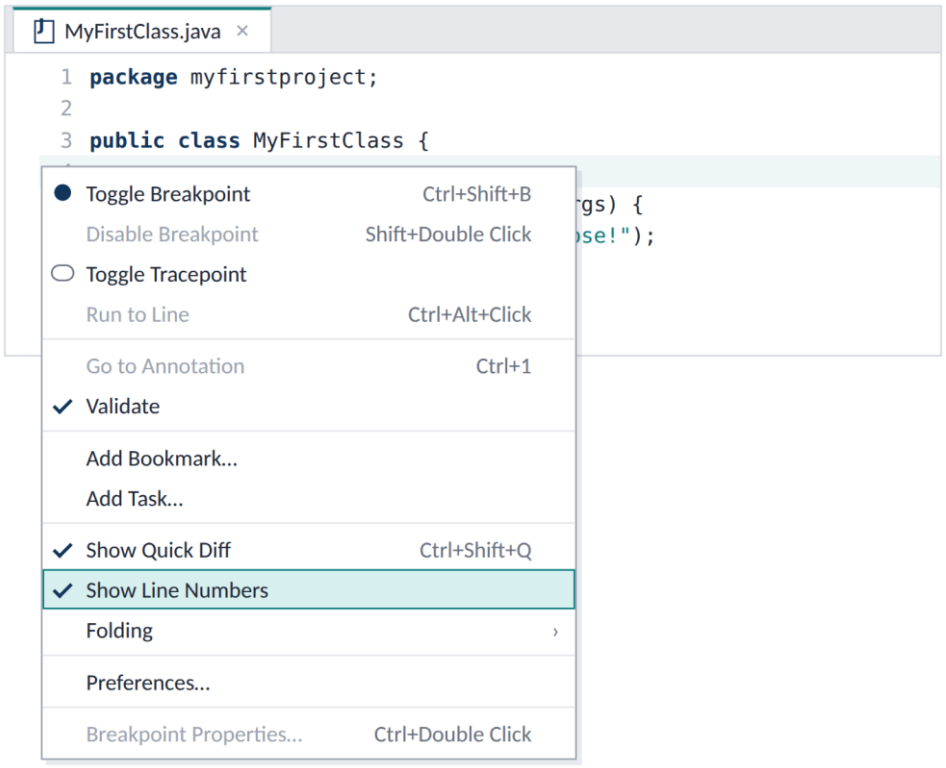
Gambar 2.11 View Problems dengan dua peringatan; menu konteksnya menawarkan Quick Fix.

Javadoc. *View* Javadoc menampilkan dokumentasi elemen Java apa pun yang sedang ditunjuk kursor. Dengan kursor pada `println`, misalnya, ia menampilkan uraian resmi `PrintStream.println(String)` (**Gambar 2.12**). Bab 6 mengajari Anda menulis dokumentasi semacam itu untuk kelas-kelas Anda sendiri.



Gambar 2.12 View Javadoc menampilkan dokumentasi println(String) ketika kursor berada pada pemanggilannya.

Editor Java. Setiap berkas Java terbuka dalam editor Java-nya sendiri. Klik kanan pada jalur abu-abu yang sempit di sebelah kiri teks, yang disebut *ruler* atau *gutter*, menawarkan pengaturan untuk editor itu (**Gambar 2.13**). Nyalakan **Show Line Numbers** sekarang juga: setiap pesan galat dan setiap laporan *debugger* dalam bab ini merujuk ke nomor baris, dan menghitung baris dengan mata hanya membuang waktu hidup Anda. Menu yang sama memuat **Toggle Breakpoint**, yang kita pakai mulai Subbab 2.10.



Gambar 2.13 Klik kanan pada ruler kiri editor: Show Line Numbers tercentang, dan Toggle Breakpoint berada paling atas.

2.5.4 Proyek dan *nature*-nya

Proyek Eclipse adalah folder yang menghimpun berkas sumber, berkas konfigurasi, dan berkas hasil kompilasi dari satu pekerjaan menjadi satu kesatuan yang dapat dibangun dan dipakai ulang. Setiap proyek memiliki berkas kecil bernama `.project` di foldernya, dan berkas itu mencatat *nature* proyek: label-label yang menyatakan jenis proyek tersebut. Proyek dengan *Java nature* adalah proyek Java, sehingga Eclipse mengompilasi berkas-berkas `.java`-nya dan menawarkan perkakas Java untuknya. Sebuah proyek dapat memiliki beberapa *nature* sekaligus, misalnya Java dan Maven, untuk menggambarkan beberapa aspek teknis dari kode yang sama. Anda jarang perlu menyunting `.project` secara manual, tetapi mengetahui keberadaannya menjelaskan mengapa menyalin sebuah folder ke dalam *workspace* tidak sama dengan mengimpor proyek (Subbab 2.8.6).

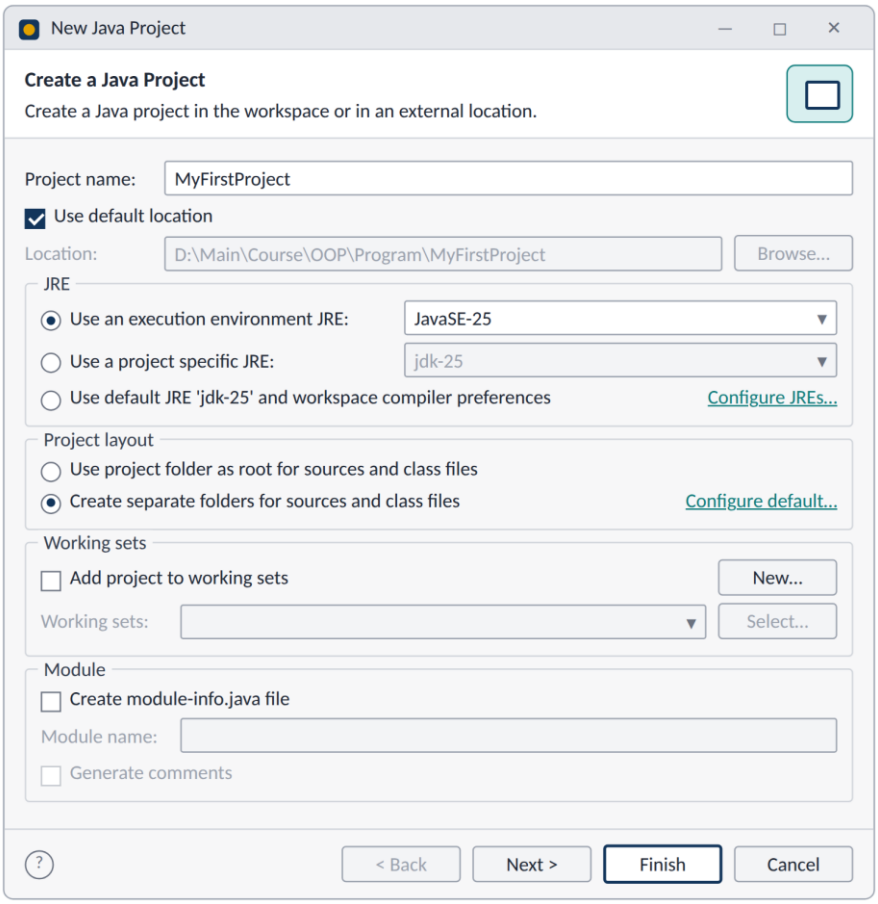
2.6 Proyek, Paket, dan Kelas Eclipse Pertama Anda

Sekarang kita membangun, menjalankan, dan mengekspor program Java sekecil mungkin di Eclipse, beserta kembarannya dalam Python di VS Code.

2.6.1 Membuat proyek

Pilih *File* > *New* > *Java Project*. Pada dialog di **Gambar 2.14**, ketik `MyFirstProject` sebagai nama proyek. Biarkan *Use default location* tercentang, agar proyek ditempatkan di dalam *workspace*. Pada bagian *JRE*, pilih versi Java yang Anda inginkan (Java 25 bila Anda memasangnya di Bab 1). Pada bagian *Project layout*, pertahankan *Create separate folders for sources and class files*: pilihan ini menaruh berkas-berkas `.java` Anda di folder bernama `src` dan berkas-berkas `.class` hasil kompilasi di folder terpisah bernama `bin`, sehingga keduanya tidak pernah bercampur. Bila ada

bagian *Module* yang menawarkan pembuatan berkas `module-info.java`, biarkan kotak itu tidak tercentang; modul Java adalah fitur lanjutan yang tidak dibutuhkan buku ini. Tekan **Finish**.



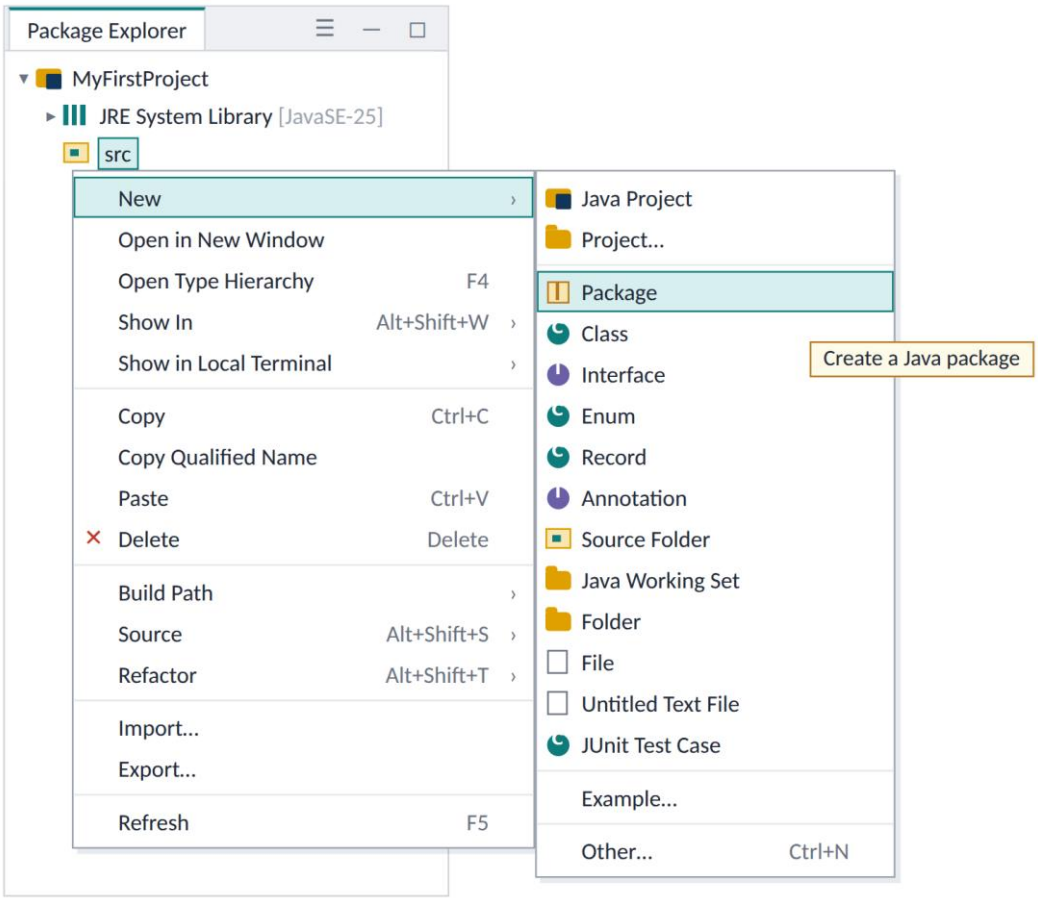
Gambar 2.14 Wizard New Java Project, dengan folder terpisah untuk berkas sumber dan berkas kelas.

Proyek baru itu muncul di Package Explorer sebagai sebuah folder. Bukalah: Anda akan melihat JRE System Library (kelas-kelas Java standar yang dapat dipakai kode Anda) dan folder `src` yang masih kosong.

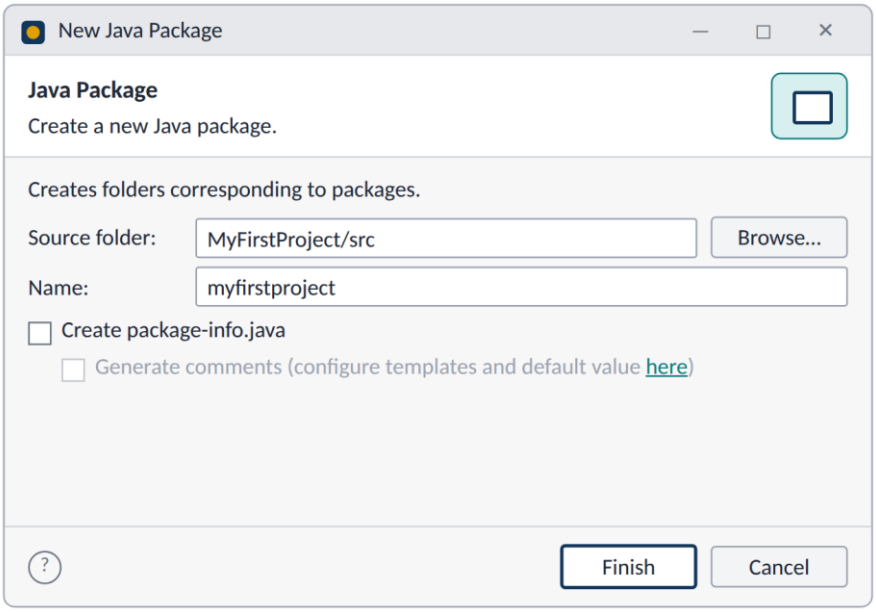
2.6.2 Membuat paket

Paket (*package*) adalah sekelompok kelas yang saling berkaitan dan diberi nama; Subbab 2.8 menjelaskan paket secara tuntas. Menurut konvensi, nama paket ditulis seluruhnya dengan huruf kecil, dan paket tingkat teratas sebuah proyek sering diberi nama yang sama dengan proyeknya, sehingga mudah menemukan proyek asal suatu potongan kode. Karena itu kita memakai nama paket `myfirstproject`.

Klik kanan folder `src` dan pilih *New* > *Package* (**Gambar 2.15**). Ketik namanya dan tekan **Finish** (**Gambar 2.16**).



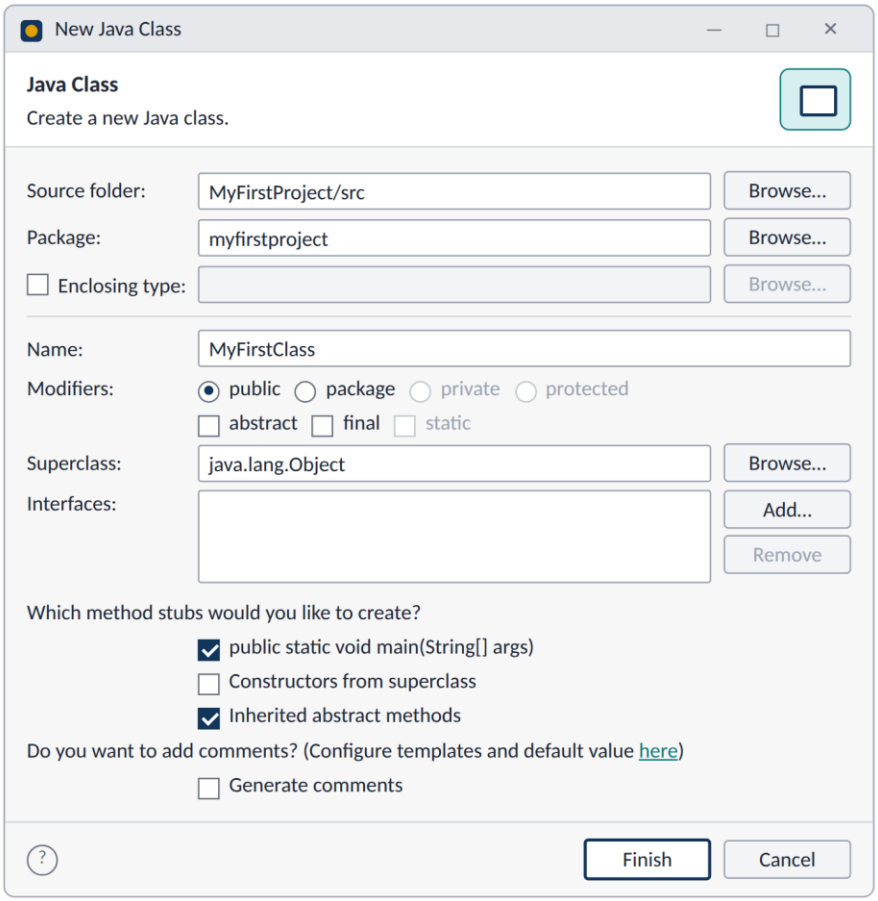
Gambar 2.15 Klik kanan pada src, lalu New > Package.



Gambar 2.16 Dialog New Java Package: folder sumber dan nama paket.

2.6.3 Membuat kelas dan menjalankannya

Klik kanan paket yang baru dibuat dan pilih *New* > *Class*. Pada dialog di **Gambar 2.17**, ketik `MyFirstClass` sebagai nama, centang kerangka **public static void main(String[] args)**, lalu tekan **Finish**. Anda juga dapat mengetik nama paket baru langsung di dialog ini; Eclipse kemudian akan membuatkan paket itu untuk Anda.



Gambar 2.17 Dialog New Java Class, dengan kerangka method main tercentang.

Eclipse membuat berkas itu dan membukanya dengan komentar `// TODO Auto-generated method stub` di dalam `main`. Gantilah komentar itu dengan satu baris, sehingga isi berkas menjadi seperti **Kode 2.3**.

Kode 2.3 `MyFirstClass.java` dalam paket `myfirstproject`: program pertama proyek Eclipse.

```
package myfirstproject;

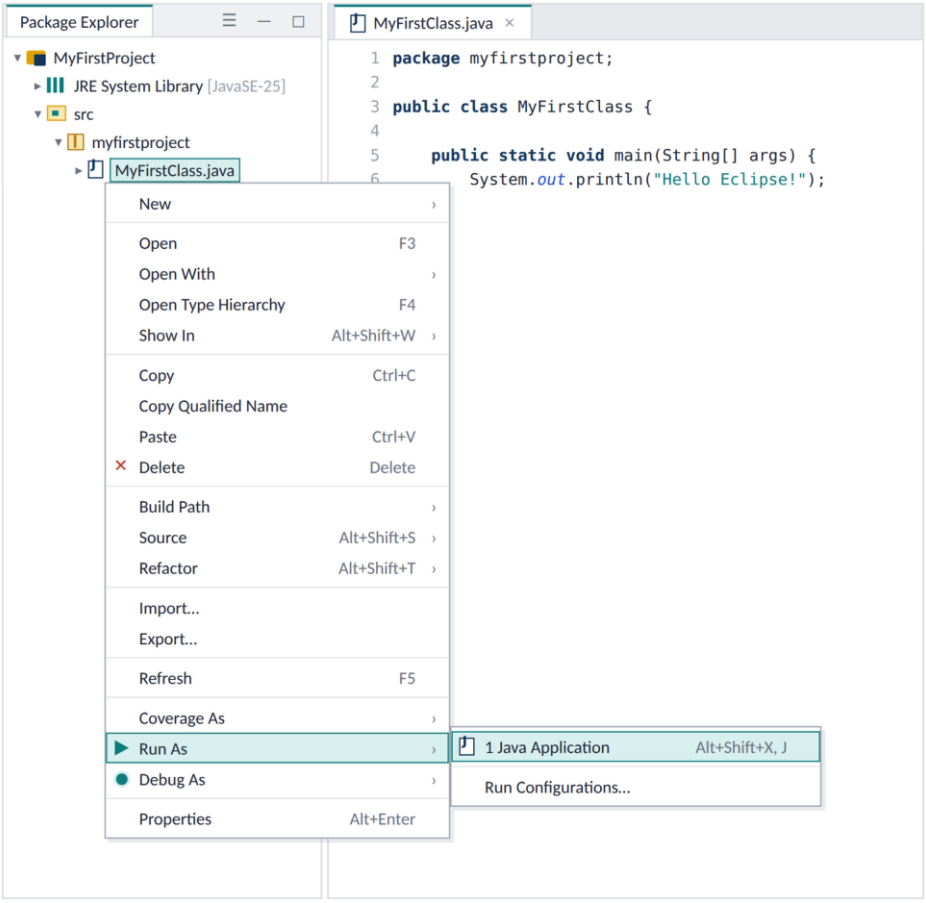
public class MyFirstClass {

    public static void main(String[] args) {
        System.out.println("Hello Eclipse!");
    }
}
```

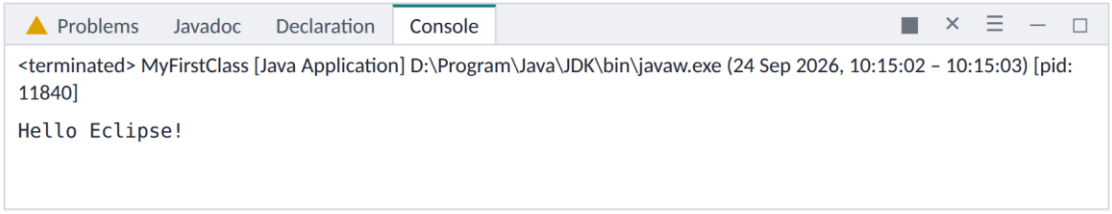
Baris pertama, `package myfirstproject;`, menyatakan paket tempat kelas itu berada; paket ini harus sesuai dengan folder tempat berkasnya disimpan (`src/myfirstproject`). Sisanya adalah bentuk yang sudah Anda kenal dari Bab 1: sebuah kelas publik yang namanya sama dengan nama berkasnya, dan *method* `main` tempat program dimulai.

Untuk menjalankannya, klik kanan kelas itu di Package Explorer, atau klik kanan di mana saja di dalam editornya, lalu pilih *Run As > Java Application* (**Gambar 2.18**). Eclipse menyimpan berkas itu bila perlu, mengompilasinya (sebenarnya ia sudah mengompilasi di latar belakang setiap kali Anda menyimpan), menjalankan sebuah JVM, dan menampilkan keluaran program di *view* **Console** (**Gambar 2.19**):

```
Hello Eclipse!
```



Gambar 2.18 Run As > Java Application dari menu konteks kelas.



Gambar 2.19 View Console setelah program dijalankan. Baris pertama mencatat program mana yang berjalan, di atas Java yang mana, dan kapan.

Eksekusi pertama membuat sebuah **konfigurasi peluncuran** (*launch configuration*), yaitu catatan tersimpan tentang kelas mana yang dijalankan, di atas JRE yang mana, dan dengan argumen apa; tombol **Run** yang hijau mengulang konfigurasi yang paling akhir, dan *Run > Run Configurations…* menyuntingnya.

Kembarannya dalam Python adalah **Kode 2.4**. Untuk satu berkas saja, Python tidak memerlukan proyek maupun paket: buatlah berkas bernama `my_first_module.py` di folder mana pun, ketik programnya, lalu jalankan.

Kode 2.4 `my_first_module.py`: program pertama yang sama dalam Python.

```
"""my_first_module.py: program pertama Bab 2."""


def main():
    print("Hello VS Code!")


if __name__ == "__main__":
    main()
```

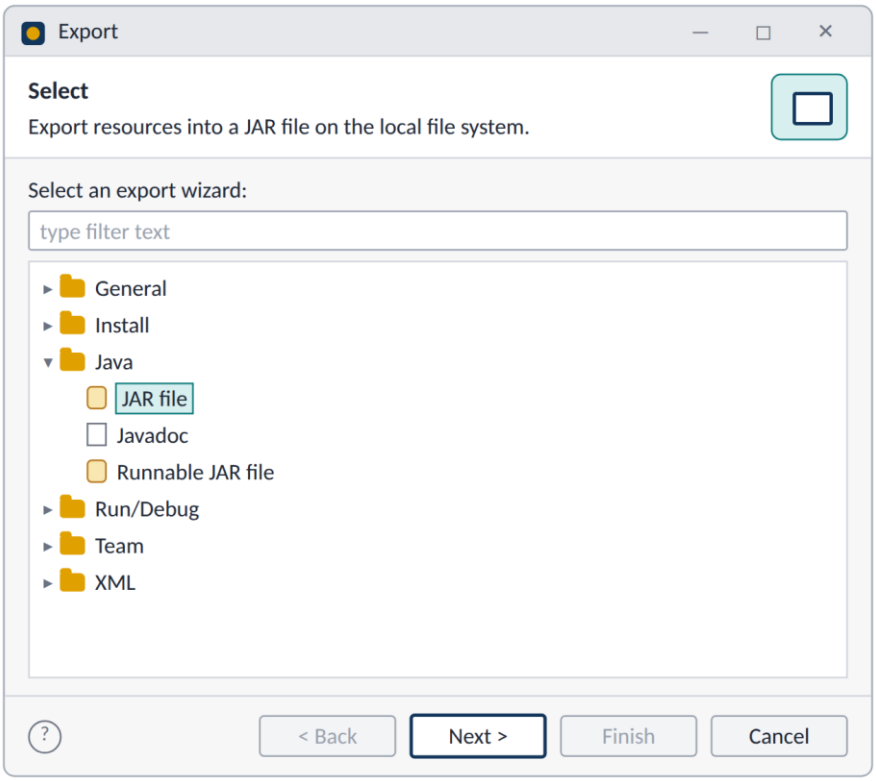
```
Hello VS Code!
```

Dua baris terakhir layak diberi penjelasan, karena Anda akan melihatnya di setiap program Python dalam buku ini. Ketika Python menjalankan sebuah berkas secara langsung, ia mengisi variabel khusus `__name__` dengan string `"__main__"`; ketika berkas yang sama *diimpor* oleh modul lain, `__name__` berisi nama modul itu sendiri. Karena itu, pengujian `if __name__ == "__main__":` berarti "panggil `main()` hanya bila berkas ini dijalankan sebagai program, bukan ketika ia diimpor". Inilah padanan Python bagi aturan Java bahwa eksekusi dimulai di `main`.

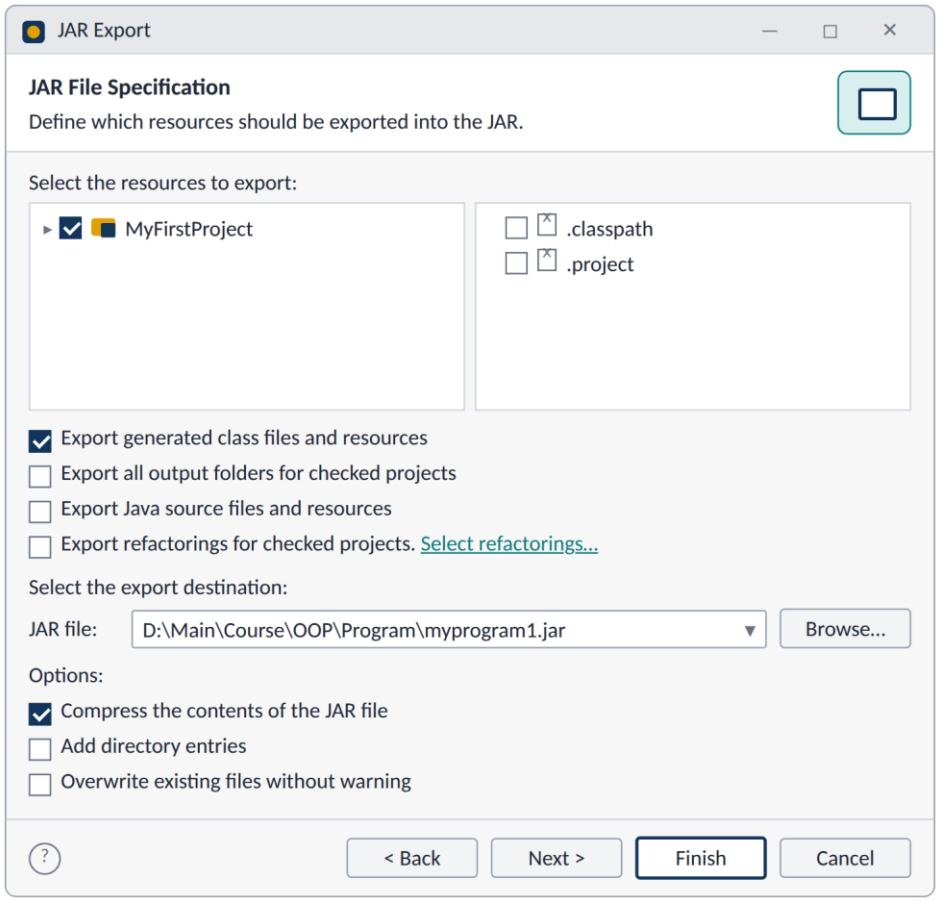
2.6.4 Menjalankan program di luar Eclipse: berkas JAR

Pengguna Anda tidak akan memasang Eclipse hanya untuk menjalankan program Anda. Cara baku untuk mendistribusikan program Java adalah **berkas JAR** (*Java ARchive*): satu berkas berformat ZIP yang memuat berkas-berkas `.class` hasil kompilasi dan sebuah berkas keterangan kecil yang disebut *manifest*.

Untuk membuatnya di Eclipse, klik kanan proyek dan pilih *Export...*, lalu *Java > JAR file* dan **Next** (**Gambar 2.20**). Pada halaman berikutnya (**Gambar 2.21**) centang proyeknya, pilih tujuan seperti `myprogram1.jar` di folder *workspace*, lalu tekan **Finish**.



Gambar 2.20 Wizard Export: JAR file di bawah kategori Java.



Gambar 2.21 Halaman JAR File Specification: sumber daya mana yang diekspor dan ke mana JAR ditulis.

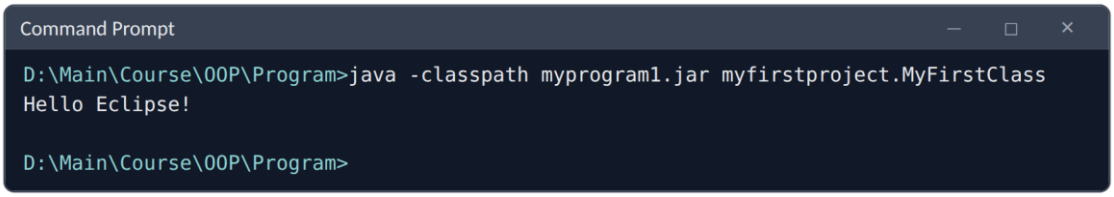
Sekarang bukalah terminal (di Windows, tekan tombol Windows, ketik `cmd`, lalu tekan Enter), pindahlah ke folder yang memuat JAR itu dengan `cd`, dan jalankan kelasnya dari JAR. **Classpath** adalah daftar tempat JVM mencari kelas; opsi `-classpath` (atau bentuk pendeknya, `-cp`) menaruh JAR itu di dalam daftar tersebut, dan kata terakhir menyebut kelas yang dijalankan, *lengkap dengan paketnya*:

```
java -classpath myprogram1.jar myfirstproject.MyFirstClass
```

```
Hello Eclipse!
```

Gambar 2.22 memperlihatkan perintah yang sama di Command Prompt Windows. Bila Anda penasaran dengan isi JAR itu, perkakas `jar` milik JDK mendaftarnya dengan `jar tf myprogram1.jar` [6]:

```
META-INF/
META-INF/MANIFEST.MF
myfirstproject/
myfirstproject/MyFirstClass.class
```

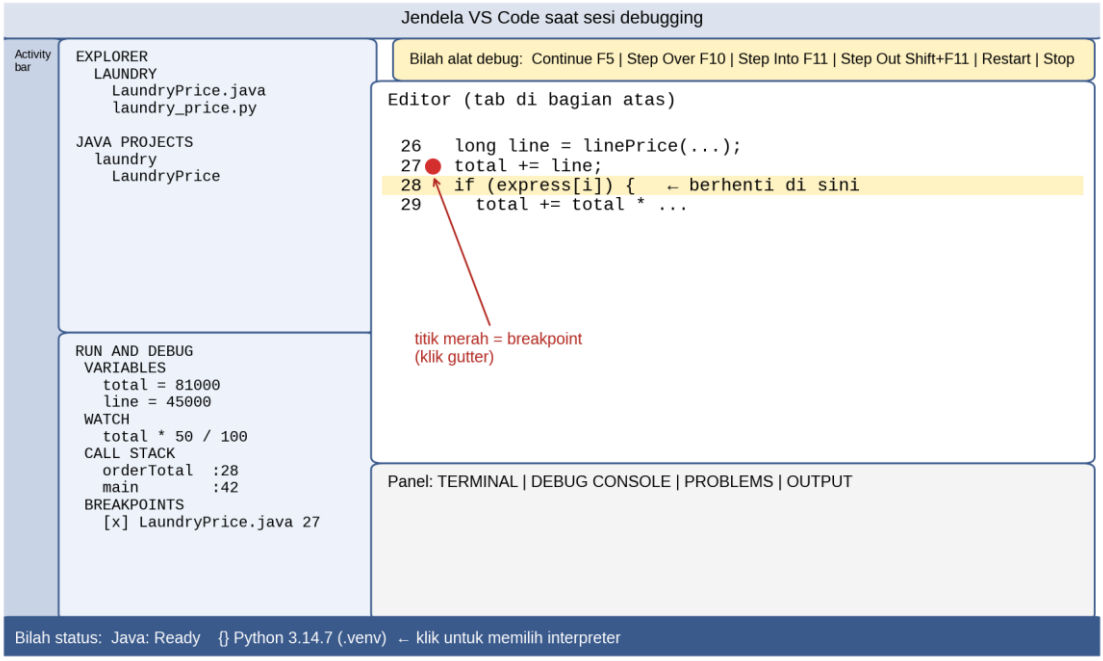


Gambar 2.22 Menjalankan kelas dari JAR di Command Prompt Windows.

Perhatikan folder `myfirstproject/` di dalam JAR: paket kelas itu telah menjadi sebuah folder, persis seperti di folder `src`. Itu bukan kebetulan, sebagaimana dijelaskan Subbab 2.8. (Anda juga dapat membuat JAR yang *dapat dijalankan*, yang *manifest*-nya menyebut kelas utamanya sehingga `java -jar myprogram1.jar` sudah cukup; wizard Export menawarkannya sebagai *Runnable JAR file*.)

2.7 Pekerjaan yang Sama di VS Code

Semua yang baru saja Anda kerjakan di Eclipse memiliki padanan langsung di VS Code, dan karena VS Code memperlakukan Java dan Python dengan cara yang sama, alat inilah yang dipakai buku ini sehari-hari. **Gambar 2.23** adalah peta jendela VS Code selama sesi *debugging*; biarkan gambar itu terlihat selama Anda membaca subbab ini dan Subbab 2.10.8.



Gambar 2.23 Peta jendela VS Code saat men-debug: Explorer dan Java Projects di bilah samping, editor dengan breakpoint di gutter, view Run and Debug, bilah alat debug, panel, dan pemilih interpreter di bilah status.

2.7.1 Folder sebagai pengganti *workspace*

VS Code tidak memiliki dialog *workspace* saat dimulai. Sebagai gantinya Anda membuka sebuah **folder** dengan *File > Open Folder…*, dan folder itulah area kerja Anda: *view* **Explorer** di bilah samping menampilkan berkas-berkasnya, dan pengaturan yang khusus untuk folder itu disimpan dalam subfolder tersembunyi bernama `.vscode`. Bila Anda memerlukan beberapa folder sekaligus, *File > Add Folder to Workspace…* membuat *multi-root workspace* yang disimpan sebagai berkas `.code-workspace`; untuk buku ini satu folder per bab sudah cukup.

Susunan yang baik adalah membuka folder `Code_new/Chapter02` yang menyertai buku ini, yang berisi folder `java` dan folder `python`. Ketika VS Code pertama kali membuka folder yang berisi kode, ia mungkin bertanya apakah Anda memercayai folder itu; jawablah ya untuk folder milik Anda sendiri, karena folder yang tidak dipercaya dijalankan dalam mode terbatas yang menonaktifkan *debugger*.

2.7.2 Java di VS Code: *view* Java Projects

Extension Pack for Java mengenali folder biasa yang berisi berkas-berkas `.java` (proyek *unmanaged*), proyek Eclipse (ia membaca `.project` dan `.classpath`), serta proyek Maven atau Gradle [3]. *View* **Java Projects**, di bagian bawah bilah samping Explorer, menampilkan paket, kelas, dan pustaka setiap proyek, mirip dengan Package Explorer di Eclipse; tombol **+**-nya membuat proyek baru, dan klik kanan pada sebuah paket menawarkan *New Java Class*.

Untuk menjalankan program Java, buka berkas yang memuat `main`. Dua tautan kecil, **Run** dan **Debug**, muncul tepat di atas *method* `main`; tautan ini disebut *CodeLens*. Mengklik **Run** mengompilasi program dan menjalankannya di panel **Terminal** di bagian bawah jendela. Bilah status menampilkan *Java: Ready* setelah *language server* Java selesai membaca proyek Anda; bila tulisan *Java: Loading* bertahan lama, tunggulah, karena pelengkapan kode dan tanda galat belum tersedia sampai ia siap.

JDK mana yang mengompilasi dan menjalankan kode Anda? VS Code memakai JDK yang ditemukannya melalui variabel lingkungan `JAVA_HOME` atau pada `PATH`, dan perintah *Java: Configure Java Runtime* (buka Command Palette dengan Ctrl+Shift+P, atau Cmd+Shift+P di macOS, lalu mulailah mengetik nama perintahnya) menampilkan dan mengubah pilihan itu untuk setiap proyek.

2.7.3 Python di VS Code: memilih interpreter

Sebuah komputer sering memiliki beberapa Python: milik Anda, milik sistem operasi, dan satu di dalam setiap *virtual environment* (lingkungan virtual, yaitu folder pribadi berisi Python beserta paket-paket untuk satu proyek). Banyak masalah "modulnya tidak ditemukan, padahal sudah saya pasang!" berasal dari program yang berjalan di bawah Python yang berbeda dari yang Anda kira.

Aturannya sederhana. Lihatlah ujung kanan **bilah status** (**Gambar 2.23**): di sana tertera interpreter yang dipilih, misalnya *Python 3.14.7 (.venv)*. Klik tulisan itu, atau jalankan *Python: Select Interpreter* dari Command Palette, lalu pilih interpreter yang Anda kehendaki dari daftarnya [4]. Pilihan itu diingat untuk folder tersebut.

Untuk membuat *virtual environment*, jalankan *Python: Create Environment* dan pilih *Venv*; VS Code membuat dan memilih folder `.venv` (dari terminal: `python3 -m venv .venv`, atau `py -m venv .venv` di Windows).

Untuk menjalankan berkas Python, bukalah berkas itu dan klik segitiga **Run Python File** di kanan atas editor, atau klik kanan di dalam editor dan pilih *Run Python File in Terminal*. VS Code mengetikkan perintah lengkapnya ke panel Terminal untuk Anda, termasuk jalur lengkap interpreter yang dipilih, sehingga Terminal juga memberi tahu Anda dengan tepat Python mana yang menjalankan program Anda.

2.7.4 Konfigurasi peluncuran: `launch.json`

Bila sebuah program memerlukan pengaturan khusus, misalnya argumen baris perintah, folder kerja tertentu, atau menjalankan sebuah *modul* dengan `python -m`, Anda menuliskannya dalam **konfigurasi peluncuran** (*launch configuration*), nama yang dipakai VS Code untuk gagasan yang sama dengan milik Eclipse. Buka *view* **Run and Debug** (ikon segitiga berserangga di bilah aktivitas, atau Ctrl+Shift+D), klik *create a launch.json file*, lalu pilih Java atau Python. VS Code menulis berkas `.vscode/launch.json`. Berikut contoh dengan tiga konfigurasi: satu untuk kelas

Java `main.Tester` dari Subbab 2.8, satu untuk berkas Python yang sedang aktif, dan satu yang menjalankan modul paket Python `main.tester` dari folder `python`:

```
{
    "version": "0.2.0",
    "configurations": [
        {
            "type": "java",
            "name": "Tester (Java)",
            "request": "launch",
            "mainClass": "main.Tester"
        },
        {
            "type": "debugpy",
            "name": "Python: current file",
            "request": "launch",
            "program": "${file}",
            "console": "integratedTerminal"
        },
        {
            "type": "debugpy",
            "name": "Python: main.tester module",
            "request": "launch",
            "module": "main.tester",
            "cwd": "${workspaceFolder}/python"
        }
    ]
}
```

`name` setiap konfigurasi muncul dalam daftar tarik-turun di bagian atas *view* Run and Debug; pilih salah satu lalu tekan F5 untuk memulainya di bawah *debugger*, atau Ctrl+F5 untuk menjalankannya tanpa *debugging*. Ungkapan seperti `${file}` (berkas di editor yang aktif) dan `${workspaceFolder}` (folder yang dibuka) diisi oleh VS Code ketika konfigurasi dimulai. Tipe `debugpy` menyebut *debugger* Python yang dipakai ekstensi Python; konfigurasi yang lebih lama mungkin bertuliskan `python`, dan itu masih berfungsi.

2.7.5 Terminal selalu ada

Kedua IDE menjalankan program Anda dengan perkakas baris perintah yang sama seperti di Subbab 2.3, dan VS Code memperlihatkan perintah persisnya di panel Terminal. Setiap kali IDE membuat Anda heran, salinlah perintah itu dan jalankan sendiri.

2.8 Proyek, Paket, Modul, dan *Import*

2.8.1 Mengapa kode perlu dikelompokkan

Program sepanjang lima puluh ribu baris, seperti aplikasi web di balik Profitina (Subbab 2.14), tersebar di ratusan berkas. Tanpa pengelompokan, *nama-nama bertabrakan* (dua orang sama-sama menulis kelas bernama `Counter`) dan *tak seorang pun dapat menemukan apa pun*.

Java dan Python memecahkan masalah ini dengan cara yang sama: berkas-berkas dikelompokkan ke dalam **paket** yang bernama, paket dapat memuat paket lain, dan nama lengkap sesuatu mencakup nama paket tempat ia berada. Kelas bernama `Counter` dalam paket `util` memiliki nama lengkap `util.Counter`; kelas bernama `Counter` dalam paket `org.w3c.dom.css` (pustaka Java memang memilikinya) memiliki nama lengkap `org.w3c.dom.css.Counter`. Keduanya tidak lagi bertabrakan.

> **DEFINISI.** Di Java, *paket* (*package*) adalah sekelompok kelas yang diberi nama, dinyatakan oleh pernyataan `package` pada baris pertama setiap berkas dan disimpan dalam folder yang bernama sama. Di Python, *modul* adalah satu berkas `.py`, dan *paket* adalah folder berisi modul-modul, yang biasanya ditandai oleh berkas bernama `__init__.py`. Di kedua bahasa, pernyataan *import* membuat sebuah nama dari paket lain dapat dipakai di berkas yang sedang ditulis.

2.8.2 Paket adalah folder

Aturan yang membuat paket mudah dipahami adalah: **paket adalah folder**. Di Java, kelas yang dinyatakan `package util;` harus berada dalam berkas di dalam folder bernama `util` (di bawah folder sumber `src`), dan kelas dalam paket `com.profitina.laundry` harus berada di `src/com/profitina/laundry/`. Berkas-berkas `.class` hasil kompilasi mengikuti struktur folder yang sama di folder keluaran (`bin` di Eclipse), dan juga di dalam JAR, seperti yang Anda lihat di akhir Subbab 2.6.4.

Di Python, paket juga berupa folder: folder `util` yang memuat `counter.py` adalah paket `util`, dan modul di dalamnya adalah `util.counter`. Folder itu biasanya memuat berkas bernama `__init__.py`, yang boleh kosong. Kehadirannya menyatakan "folder ini adalah paket reguler", dan kode apa pun di dalamnya dijalankan ketika paket itu pertama kali diimpor. Sejak Python 3.3 folder tanpa `__init__.py` tetap berfungsi sebagai *namespace package*, tetapi pemula sebaiknya selalu menambahkan berkas itu: ia membuat maksud Anda tersurat dan menghindarkan beberapa kasus pinggir yang membingungkan [8].

Konvensi penamaannya sedikit berbeda. Nama paket Java seluruhnya berhuruf kecil, sering dibentuk dari nama domain yang dibalik untuk kode yang akan dibagikan (`com.profitina.laundry`), dan nama kelas diawali huruf kapital (`Counter`). Nama modul dan paket Python pendek, seluruhnya berhuruf kecil, dan boleh memakai garis bawah (`laundry_price`); nama kelas diawali huruf kapital seperti di Java [9], [8].

2.8.3 Proyek dengan dua paket

Sekarang kita membangun proyek yang sedikit lebih besar, tempat satu kelas memakai kelas lain dari paket yang berbeda. Proyek itu bernama `counter` dan memiliki dua paket: `util`, yang memuat kelas `Counter` untuk menjumlahkan bilangan, dan `main`, yang memuat kelas `Tester` untuk memeriksa bahwa `Counter` bekerja.

Di Eclipse: buatlah proyek Java bernama `counter`; klik kanan `src` dan buat paket `util`, lalu paket `main`; klik kanan `util` dan buat kelas `Counter`, lalu klik kanan `main` dan buat kelas `Tester` (kali ini *dengan* kerangka *method* `main`). Di VS Code: di dalam folder `java`, buatlah folder `util` dan `main` beserta kedua berkas di dalamnya; *view* Java Projects menampilkan keduanya sebagai paket secara otomatis.

Proyek Java dan kembarannya dalam Python kemudian tampak seperti ini di disk:

```
java/                                 python/
  util/                                 util/
    Counter.java                          __init__.py      (kosong)
  main/                                   counter.py
    Tester.java                         main/
  bin/            (hasil kompilasi)       __init__.py      (kosong)
    util/Counter.class                    tester.py
    main/Tester.class
```

Kode 2.5 adalah kelas `Counter`. *Method*-nya, `count(x)`, menjumlahkan $1 + 2 + \cdots + x$ ke dalam *field* bernama `result` lalu mengembalikannya; ia menolak nilai `x` di luar rentang 1 sampai 255 dengan *melempar eksepsi* (*throwing an exception*), cara Java untuk mengatakan "saya tidak dapat melakukan ini" (Bab 5 membahas eksepsi secara tuntas).

Kode 2.5 `Counter.java` dalam paket `util`: menjumlahkan $1 + 2 + \cdots + x$ untuk $1 \le x \le 255$.

```
package util;

public class Counter {

    private int result = 0;

    public int getResult() {
        return result;
    }

    // Menjumlahkan 1 + 2 + ... + x; x harus terletak antara 1 dan 255.
    public int count(int x) {
        if (x > 0 && x <= 255) {
            for (int i = 1; i <= x; i++) {
                result += i;
            }
        } else {
            throw new RuntimeException("x should be between 1 and 255");
        }
        return result;
    }
}
```

Dalam notasi matematika, `count(x)` menghitung

$$S(x) = \sum_{i=1}^{x} i = \frac{x(x+1)}{2}, \qquad 1 \le x \le 255, \tag{2.1}$$

sehingga $S(5) = 1 + 2 + 3 + 4 + 5 = 15$, yang sesuai dengan $\frac{5 \cdot 6}{2} = 15$. Persamaan (2.1) memberi kita nilai harapan untuk menguji, dan itulah tepatnya yang dikerjakan `Tester`. (Ketika kelas ini pertama kali dibuat dari templat, badan `count` hanya berisi komentar `TODO` dan `return 0;`, seperti pada **Gambar 2.24**; melengkapinya adalah latihan kecil yang hasilnya ditunjukkan versi jadinya. Eclipse mengumpulkan setiap komentar `TODO` di *view Tasks*, sehingga komentar semacam itu menjadi daftar tugas yang sederhana.)

Counter.java × Tester.java

```java
package util;

public class Counter {
    public int count(int x) {
        // TODO check that x > 0 and <= 255
        // if not throw a new RuntimeException
        // Example for a RuntimeException:

        // throw new RuntimeException("x should be between 1 and 255");

        // TODO calculate the numbers from 1 to x
        // for example if x is 5, calculate
        // 1 + 2 + 3 + 4 + 5


        // TODO return your calculated value
        // instead of 0
        return 0;
    }

}
```

Gambar 2.24 Versi awal Counter.java: komentar TODO menguraikan apa yang harus dikerjakan count(x), dan method itu mengembalikan 0 sampai semuanya selesai.

Kode 2.6 adalah kelas `Tester`. Kelas ini membuat sebuah `Counter`, memeriksa bahwa `count(5)` menghasilkan 15, lalu memeriksa bahwa `count(256)` ditolak, dengan menangkap eksepsinya.

Kode 2.6 `Tester.java` dalam paket `main`: pengujian kecil buatan sendiri untuk `Counter`.

```java
package main;

import util.Counter;

public class Tester {

    public static void main(String[] args) {
        Counter counter = new Counter();
        int result = counter.count(5);
        if (result == 15) {
            System.out.println("Correct");
        } else {
            System.out.println("False");
        }
        try {
            counter.count(256);
        } catch (RuntimeException e) {
            System.out.println("Works as expected");
        }
    }
}
```

```
Correct
Works as expected
```

Baris `import util.Counter;` itulah yang memungkinkan `Tester`, yang berada di paket `main`, merujuk `Counter` dengan nama pendeknya. Tanpa baris itu, `Tester` harus menulis nama lengkap `util.Counter` setiap kali.

Kembarannya dalam Python adalah **Kode 2.7** dan **Kode 2.8**. Kelasnya ditulis dengan sintaksis Python (Bab 3 menjelaskan `class`, `__init__`, dan `self` dengan benar; untuk sementara bacalah `self._result` sebagai "`result` milik objek ini"), dan penolakannya berupa `ValueError`, eksepsi standar Python untuk nilai yang berada di luar rentang.

Kode 2.7 `util/counter.py`: kembaran Python bagi `Counter.java`.

```
"""util/counter.py: kembaran Python dari Counter.java."""


class Counter:

    def __init__(self):
        self._result = 0

    def get_result(self):
        return self._result

    # Menjumlahkan 1 + 2 + ... + x; x harus terletak antara 1 dan 255.
    def count(self, x):
        if 0 < x <= 255:
            for i in range(1, x + 1):
                self._result += i
        else:
            raise ValueError("x should be between 1 and 255")
        return self._result
```

Kode 2.8 `main/tester.py`: kembaran Python bagi `Tester.java`.

```
"""main/tester.py: memeriksa util.counter.Counter; jalankan dengan python -m main.tester"""
from util.counter import Counter


def main():
    counter = Counter()
    result = counter.count(5)
    if result == 15:
        print("Correct")
    else:
        print("False")
    try:
        counter.count(256)
    except ValueError:
        print("Works as expected")


if __name__ == "__main__":
    main()
```

Import di Python menyebut *modul*-nya, lalu sesuatu di dalamnya: `from util.counter import Counter` berarti "dari modul `counter` di dalam paket `util`, bawalah nama `Counter`". Untuk menjalankan modul yang berada di dalam paket, pergilah ke folder yang *memuat* paket-paket itu (di sini `python`) dan pakai sakelar `-m` dengan nama bertitik modul tersebut, tanpa `.py`:

```
cd python
python3 -m main.tester
```

```
Correct
Works as expected
```

Mengapa tidak cukup `python3 main/tester.py`? Karena Python mencari paket yang diimpor mulai dari folder skrip yang diberikan kepadanya. Bila dijalankan sebagai `main/tester.py`, ia akan mencari `util` di dalam `main`, tidak menemukannya, lalu berhenti dengan

`ModuleNotFoundError: No module named 'util'`. Bila dijalankan dengan `-m` dari folder `python`, ia mencari di folder `python`, tempat `util` berada. Satu aturan ini menjelaskan sebagian besar galat "*module not found*" yang dijumpai pemula.

2.8.4 *Import* yang terlupa, di kedua bahasa

Sekarang mari kita buat kesalahan yang dijanjikan Subbab 2.3.5: hapus baris *import* dari kedua *tester* dan lihat bagaimana masing-masing bahasa bereaksi.

Di Eclipse, begitu *import* itu hilang, `Counter` digarisbawahi merah di editor, sebuah penanda merah muncul di *ruler* kiri, dan *view* Problems mendaftar galatnya. Mengompilasi dari terminal memperlihatkan persis apa yang ditanggapi IDE:

```
javac -d bin src/util/Counter.java src/main/Tester.java
```

```
src/main/Tester.java:7: error: cannot find symbol
        Counter counter = new Counter();
        ^
  symbol:   class Counter
  location: class Tester
src/main/Tester.java:7: error: cannot find symbol
        Counter counter = new Counter();
                          ^
  symbol:   class Counter
  location: class Tester
2 errors
```

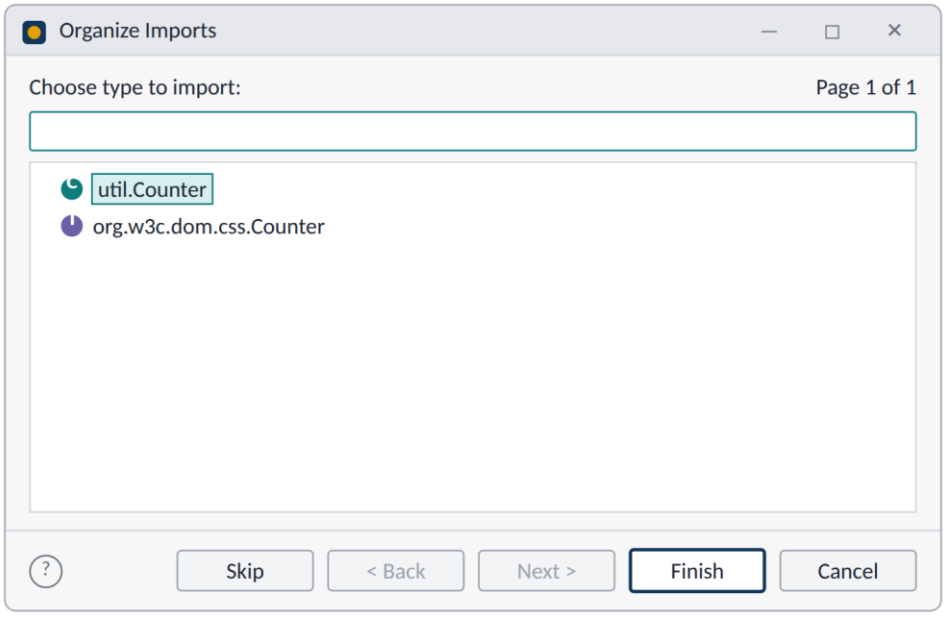
Kompilator menemukan masalah itu pada *waktu kompilasi*, sebelum apa pun berjalan: nama `Counter` pada baris 8 berkas aslinya (kini baris 7, karena baris *import* sudah hilang) tidak dapat ditemukan dalam paket `main`. Ia melaporkannya dua kali, sekali untuk tipe dalam `Counter counter` dan sekali untuk konstruktor dalam `new Counter()`, masing-masing dengan tanda sisipan (*caret*) di bawahnya.

IDE menawarkan untuk memperbaikinya bagi Anda. Arahkan tetikus ke garis bawah merah itu (atau klik penanda merah di *ruler*) dan sebuah kotak mendaftar *quick fix* (**Gambar 2.25**). Yang pertama, *Import 'Counter' (util)*, menambahkan persis baris yang tadi kita hapus. Perhatikan usulan kedua, *Import 'Counter' (org.w3c.dom.css)*: pustaka Java memang memuat kelas lain bernama `Counter`, dan hanya Anda yang tahu mana yang Anda maksud. Inilah tabrakan nama dari Subbab 2.8.1, dan alasan mengapa paket ada.



Gambar 2.25 Counter cannot be resolved to a type: kotak quick fix menawarkan import util.Counter, atau org.w3c.dom.css.Counter yang tidak berkaitan.

Eclipse memiliki cara kedua untuk menambahkan *import*, yang lebih baik bila beberapa *import* hilang sekaligus: *Source* > *Organize Imports* (Ctrl+Shift+O, atau Cmd+Shift+O di macOS). Perintah ini menambahkan setiap *import* yang dapat ditentukan tanpa bertanya, membuang *import* yang tidak dipakai, dan mengurutkan sisanya. Bila sebuah nama pendek cocok dengan beberapa kelas, ia bertanya, seperti pada **Gambar 2.26**. VS Code menawarkan perintah yang sama sebagai *Organize Imports* (Shift+Alt+O) dan *quick fix* yang sama di bawah ikon bola lampu (Ctrl+.).



Gambar 2.26 Organize Imports menanyakan Counter mana yang dimaksud ketika nama pendeknya ambigu.

Python, tanpa *import* itu, mulai berjalan dan baru gagal ketika mencapai baris yang memakai nama tersebut:

```
Traceback (most recent call last):
  File "<frozen runpy>", line 198, in _run_module_as_main
  File "<frozen runpy>", line 88, in _run_code
  File "/home/student/counter/py/main/tester.py", line 18, in <module>
    main()
  File "/home/student/counter/py/main/tester.py", line 5, in main
    counter = Counter()
              ^^^^^^^
NameError: name 'Counter' is not defined. Did you mean: 'counter'?
```

Ini adalah galat *waktu jalan*: program sudah dimulai, memanggil `main()`, dan baru di baris 5 mendapati bahwa `Counter` tidak ada. (Bingkai `<frozen runpy>` milik mesin di balik `python -m`.) Usulan `counter` hanyalah tebakan, bukan jawaban. Ekstensi Python di VS Code menggarisbawahi nama yang tidak terdefinisi sebelum Anda menjalankan program, tetapi itu adalah bantuan editor, bukan bahasanya.

2.8.5 Mengompilasi proyek berpaket secara manual

Bila sebuah proyek Java memiliki paket, perintah terminal di Subbab 2.3.3 memerlukan dua tambahan kecil. Dari folder yang memuat `src`:

```
javac -d bin src/util/Counter.java src/main/Tester.java
java -cp bin main.Tester
```

Opsi `-d bin` menyuruh `javac` menulis berkas-berkas `.class` di bawah folder `bin`, dengan membuat satu subfolder untuk setiap paket. Mendaftar isi `bin` sesudahnya memperlihatkan hasilnya:

```
bin
bin/main
bin/main/Tester.class
bin/util
bin/util/Counter.class
```

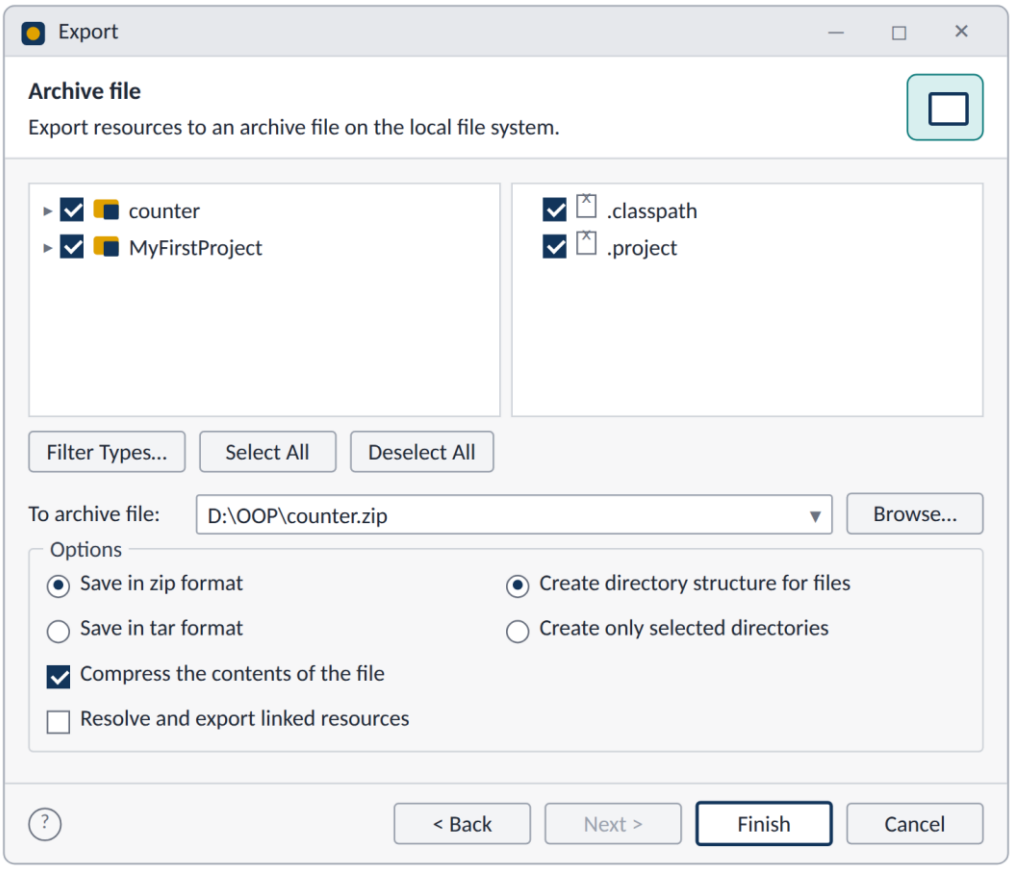
Opsi `-cp bin` menaruh `bin` di *classpath*, dan kelas yang dijalankan disebut lengkap dengan paketnya, `main.Tester`, tidak pernah dengan garis miring atau ekstensi `.class`. Keluarannya sama seperti di Eclipse:

```
Correct
Works as expected
```

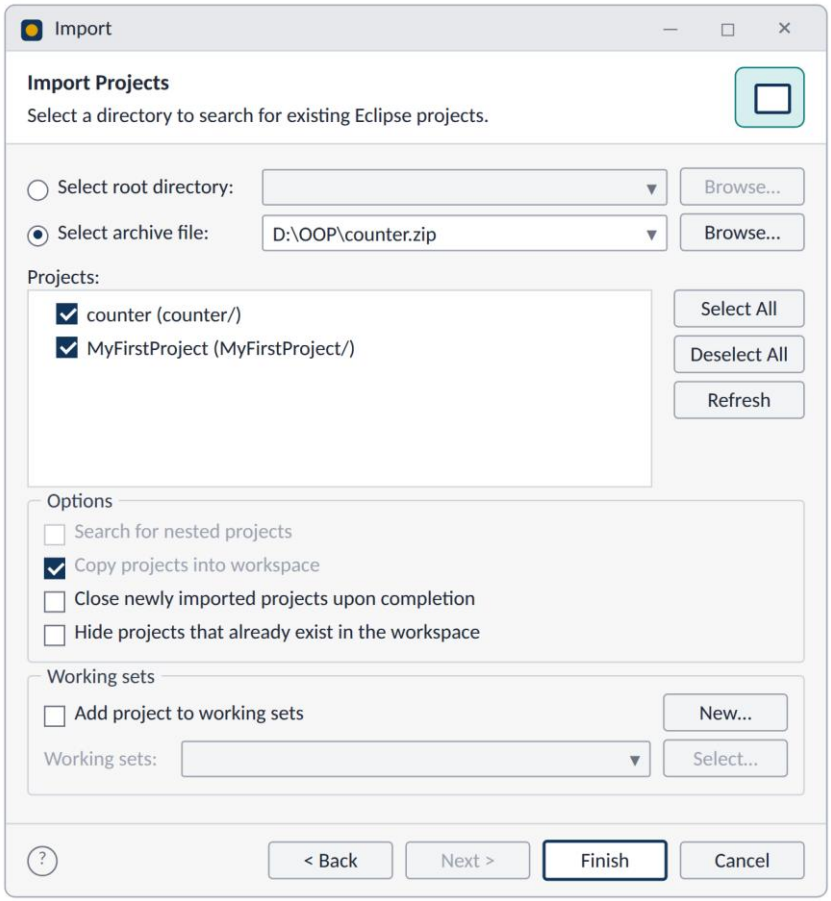
Dua baris itulah yang dikerjakan Eclipse dan VS Code untuk Anda ketika Anda menekan *Run*.

2.8.6 Mengekspor, mengimpor, dan menjelajahi proyek

Mengekspor dan mengimpor. Untuk memberikan seluruh proyek Eclipse kepada orang lain, atau memindahkannya ke *workspace* lain, pakailah *File > Export > General > Archive File*, centang proyek-proyeknya, lalu pilih sebuah berkas `.zip` (**Gambar 2.27**). Untuk memasukkan proyek, pakailah *File > Import > General > Existing Projects into Workspace*, pilih *Select archive file* (atau *Select root directory* bila berkas zip itu sudah diekstrak), centang proyek-proyeknya, lalu tekan **Finish** (**Gambar 2.28**). Sebuah proyek hanya dapat diimpor bila belum ada proyek bernama sama di *workspace*; bila Anda mengimpor ulang proyek Anda sendiri, pastikan berkas zip-nya sudah ada sebelum Anda menghapus salinan yang lama. VS Code tidak memerlukan semua ini: proyek hanyalah sebuah folder, sehingga mengompres dan mengekstrak folder itu sudah cukup.

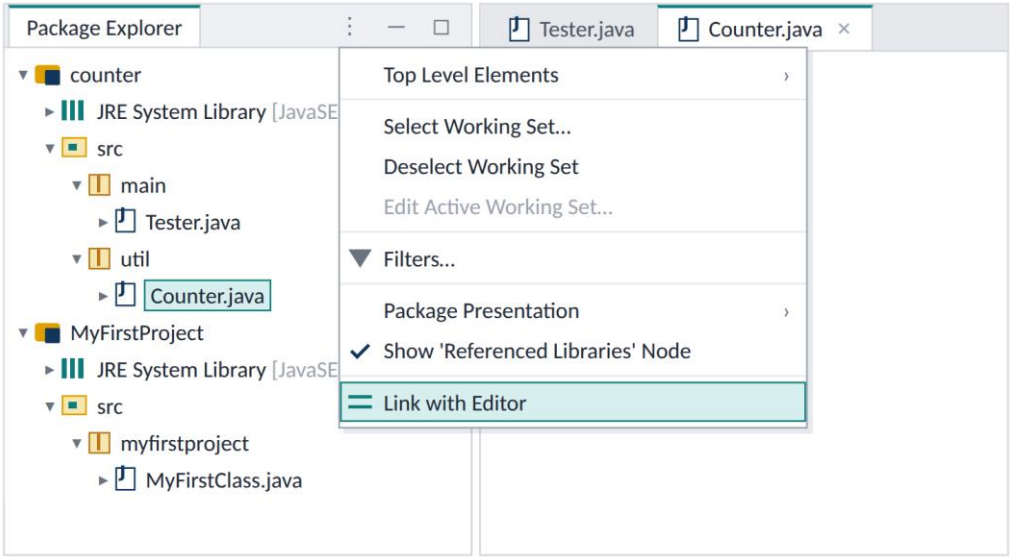


Gambar 2.27 Mengekspor dua proyek ke berkas arsip, dengan struktur foldernya tetap terjaga.



Gambar 2.28 Mengimpor proyek yang sudah ada dari berkas arsip ke dalam workspace.

Menautkan explorer dengan editor. Dalam proyek besar, mudah sekali kehilangan jejak *di mana* berkas yang sedang Anda sunting berada. Menu *view* Package Explorer (tiga titik di kanan atasnya) memiliki butir **Link with Editor** (**Gambar 2.29**); bila butir ini aktif, berpindah ke editor `Counter.java` akan memilih `Counter.java` di pohon proyek. Butir *Filters...*-nya menyembunyikan jenis-jenis berkas yang tidak ingin Anda lihat. VS Code melakukan hal yang sama secara otomatis: Explorer mengikuti editor yang aktif kecuali Anda mematikan pengaturan *Explorer: Auto Reveal*.



Gambar 2.29 Menu view Package Explorer, dengan Link with Editor yang akan dinyalakan.

2.9 Perkakas Andalan Penyuntingan: *Content Assist*, *Quick Fix*, *Refactoring*, dan Pemformatan

Editor pada IDE yang baik mengerjakan jauh lebih banyak daripada sekadar menerima ketikan Anda. Empat perkakasnya akan menghemat waktu Anda berjam-jam sejak minggu pertama.

***Content assist* (IntelliSense).** Ketik awal sebuah nama lalu tekan Ctrl+Space: editor mendaftar setiap variabel, *method*, kelas, dan kata kunci yang cocok di titik itu, beserta tipe dan dokumentasinya, lalu menyisipkan yang Anda pilih. Setelah sebuah titik, ia mendaftar anggota objek di depan titik itu, sehingga `counter.` diikuti Ctrl+Space mendaftar `count(int)` dan `getResult()`. *Content assist* juga menjabarkan *templat*: di Eclipse, ketik `sysout` atau `main` lalu tekan Ctrl+Space untuk memperoleh `System.out.println();` atau sebuah *method* `main` yang lengkap; di VS Code, ketik `sysout` atau `main` lalu pilih usulannya. Di Python, mengetik `if __name__` lalu memilih usulannya menghasilkan seluruh baris `if __name__ == "__main__":`.

***Quick Fix*.** Setiap kali ada yang digarisbawahi, tekan Ctrl+1 di Eclipse atau Ctrl+. di VS Code (Cmd+1 dan Cmd+. di macOS). Anda sudah melihatnya menambahkan *import* yang hilang; ia juga dapat membuat *method* yang Anda panggil tetapi belum Anda tulis, menambahkan `return` yang terlupa, mengubah tipe sebuah variabel, atau membungkus pemanggilan yang berisiko dengan `try` dan `catch` yang diwajibkan Java.

***Refactoring*.** Melakukan *refactoring* berarti mengubah struktur kode tanpa mengubah apa yang dikerjakannya [10]. *Refactoring* yang paling penting adalah **Rename**: taruh kursor pada `count`, pilih *Refactor > Rename* di Eclipse (Alt+Shift+R) atau tekan F2 di VS Code, ketik `sumUpTo`, dan IDE mengubah deklarasinya serta setiap pemakaiannya di setiap berkas, dan tidak ada yang lain, karena ia tahu `count` yang mana adalah yang mana. Menu yang sama menawarkan *Extract*

Method, *Extract Local Variable*, *Inline*, dan *Move* (ke paket lain, sambil memperbarui semua *import*); VS Code menawarkan hal yang sama untuk Python.

Pemformatan. *Source* > *Format* di Eclipse (Ctrl+Shift+F) dan *Format Document* di VS Code (Shift+Alt+F) merapikan ulang indentasi dan spasi seluruh berkas menurut satu gaya yang konsisten. Formatlah kode Anda sebelum meminta siapa pun membacanya, termasuk diri Anda sendiri esok hari: kode yang indentasinya berantakan menyembunyikan bug.

Tabel 2.3 menghimpun tombol-tombol yang layak dipelajari sekarang. Tombol Eclipse berlaku untuk Windows dan Linux; di macOS, Ctrl biasanya menjadi Cmd.

Tabel 2.3 Pintasan papan ketik yang layak dipelajari pada minggu pertama, di Eclipse dan di VS Code.

Tindakan	Eclipse	VS Code
Content assist / pelengkapan	Ctrl+Space	Ctrl+Space
Quick fix	Ctrl+1	Ctrl+.
Merapikan *import*	Ctrl+Shift+O	Shift+Alt+O
Mengganti nama di semua tempat	Alt+Shift+R	F2
Memformat berkas	Ctrl+Shift+F	Shift+Alt+F
Pergi ke definisi	F3	F12
Mencari berkas atau kelas berdasarkan nama	Ctrl+Shift+R / Ctrl+Shift+T	Ctrl+P
Mengomentari/membuka komentar baris	Ctrl+/	Ctrl+/
Menjalankan program terakhir	Ctrl+F11	Ctrl+F5
Men-debug program terakhir	F11	F5
Memasang/melepas *breakpoint* pada baris	Ctrl+Shift+B	F9

> **PERINGATAN:** Perhatikan jebakan pada baris-baris terakhir **Tabel 2.3**: di Eclipse, F11 *memulai debugging* dan F5 melakukan *step into*; di VS Code, F5 *memulai* (atau melanjutkan) *debugging* dan F11 melakukan *step into*. Bila Anda memakai kedua alat, ucapkan tindakannya dalam hati ("*step into*") alih-alih memercayai jari-jari Anda.

2.10 *Debugging* dari Nol

2.10.1 Apa itu bug, dan dua cara mencarinya

Bug adalah perbedaan antara apa yang dikerjakan program dan apa yang seharusnya dikerjakannya. Bug yang "berisik" menghentikan kompilator atau program dengan pesan yang menunjukkan di mana harus mencari (Subbab 2.13). Bug yang "diam" lebih sulit: program berjalan sampai selesai dan mencetak angka yang tampak masuk akal padahal salah. Tagihan cucian sebesar Rp 109.350 tampak sama meyakinkannya dengan tagihan sebesar Rp 93.150.

Ada dua cara klasik untuk melihat ke dalam program guna menemukan bug yang diam.

Cara pertama adalah ***print debugging***: tambahkan baris-baris `System.out.println(...)` atau `print(...)` yang menampilkan variabel pada saat-saat yang menarik, jalankan, lalu baca keluarannya. Cara ini bekerja di mana saja, tetapi setiap pertanyaan baru berarti menyunting dan menjalankan ulang, Anda harus menebak sebelumnya *apa* yang dicetak dan *di mana*, dan baris-baris tambahan itu harus dibuang sesudahnya.

Cara kedua adalah ***debugger***, yang menjalankan program Anda di bawah kendalinya sehingga Anda dapat menjedanya di baris mana pun, melihat *setiap* variabel, bergerak satu pernyataan

demi satu pernyataan masuk ke atau keluar dari pemanggilan *method*, dan bahkan mengubah nilai, semuanya tanpa menyunting program.

> **DEFINISI.** *Debugging* adalah proses menemukan dan menghilangkan penyebab sebuah bug. *Debugger* adalah alat yang menjalankan program secara interaktif, sehingga Anda dapat menghentikannya pada titik-titik pilihan, mengeksekusinya langkah demi langkah, serta memeriksa dan mengubah keadaannya sambil mengamati kode sumbernya.

2.10.2 Kosakatanya

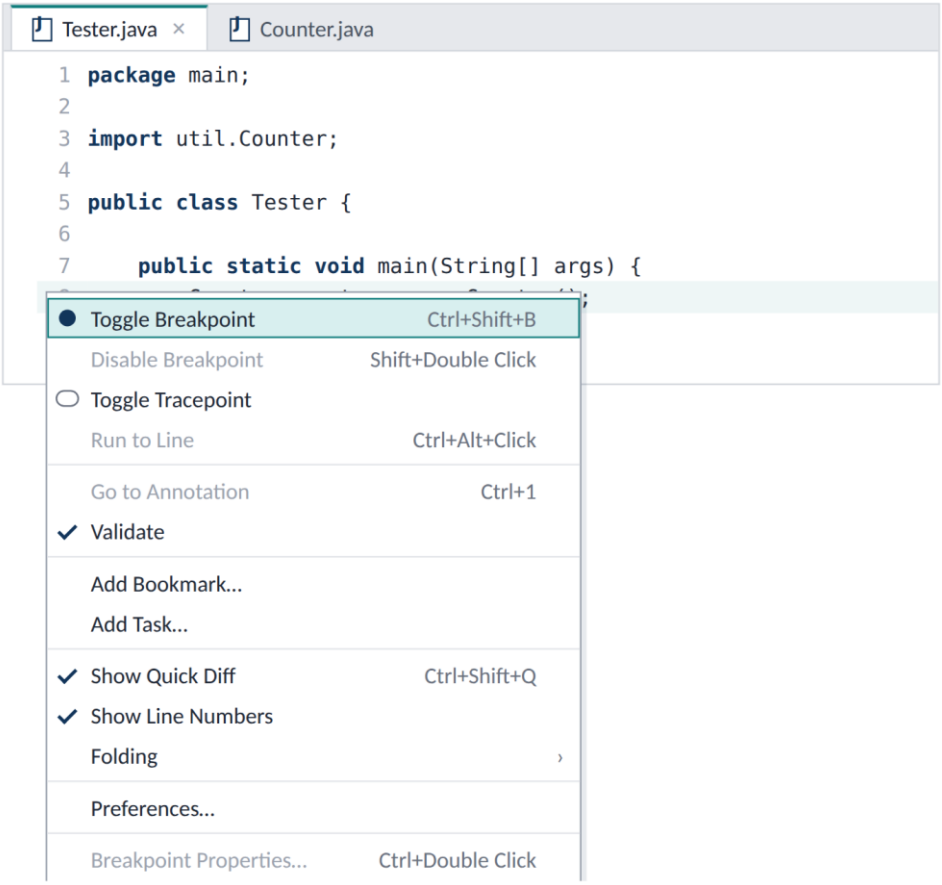
Tujuh kata sudah cukup untuk membicarakan *debugger* apa pun.

- ***Breakpoint*** (titik henti) adalah tanda pada sebuah baris yang berarti "berhentilah *sebelum* menjalankan baris ini"; kedua IDE menampilkannya sebagai titik di *ruler* kiri editor.
- Program yang berhenti pada sebuah *breakpoint* disebut **tertangguhkan** (*suspended*); baris yang akan dijalankan disorot, dan tidak ada yang berjalan sampai Anda memerintahkannya.
- ***Stepping*** (melangkah) mengeksekusi program yang tertangguhkan sedikit demi sedikit: *step over*, *step into*, atau *step out* (Subbab 2.10.4).
- ***Resume*** (*continue*, lanjutkan) menjalankan program dengan bebas sampai *breakpoint* berikutnya atau sampai selesai.
- **Variabel** dari *method* yang sedang berjalan ditampilkan beserta nilainya selama program tertangguhkan.
- ***Watch*** (pantauan) adalah sebuah ungkapan, misalnya `total * 50 / 100`, yang dievaluasi ulang pada setiap perhentian.
- **Tumpukan pemanggilan** (*call stack*) mendaftar *method-method* yang sedang berjalan, masing-masing dipanggil oleh *method* di bawahnya.

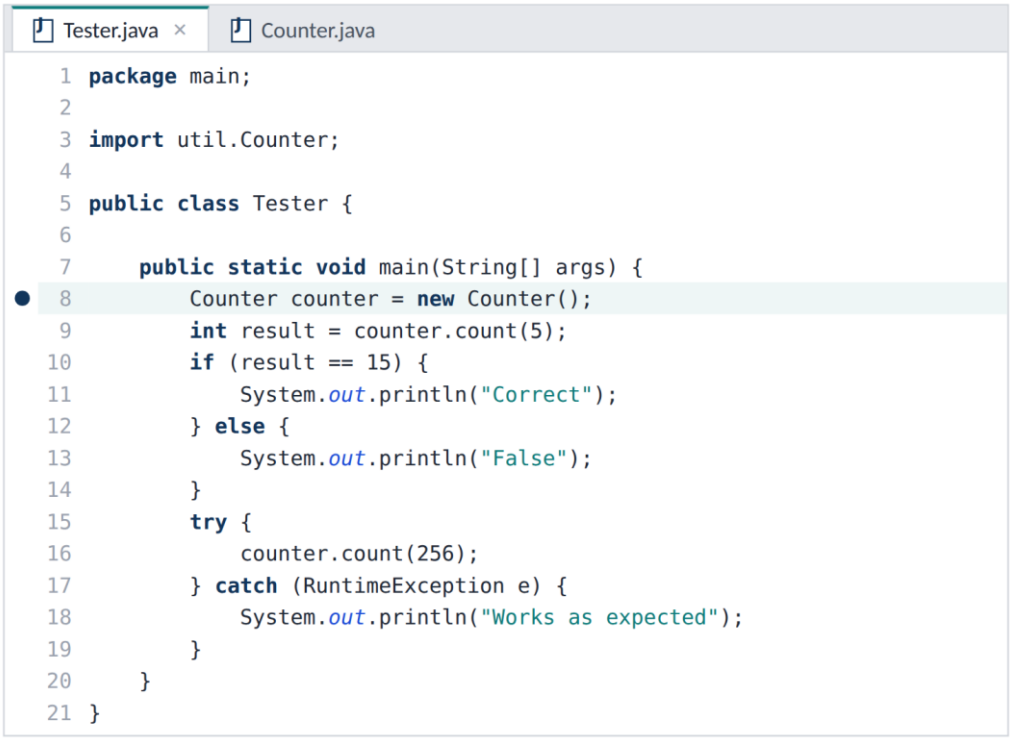
Kerabat *breakpoint*, yaitu ***watchpoint***, berhenti setiap kali sebuah *field* tertentu dibaca atau diubah, baris mana pun yang melakukannya; Eclipse memasangnya dengan klik ganda pada *ruler* di samping deklarasi *field*. *Breakpoint* dan *watchpoint* bersama-sama disebut *titik henti* (*stop points*).

2.10.3 *Breakpoint* dan memulai *debugger* di Eclipse

Kita memakai proyek `counter` dari Subbab 2.8. Bukalah `Tester.java`. Untuk memasang *breakpoint*, klik kanan pada *ruler* kiri di samping sebuah baris lalu pilih **Toggle Breakpoint** (**Gambar 2.30**), atau cukup klik ganda pada *ruler* di posisi itu, atau taruh kursor pada baris itu lalu tekan Ctrl+Shift+B. Pasanglah satu pada baris `Counter counter = new Counter();`. Sebuah titik biru kecil muncul di *ruler* (**Gambar 2.31**). Melakukan hal yang sama sekali lagi akan melepasnya; itulah arti *toggle*.



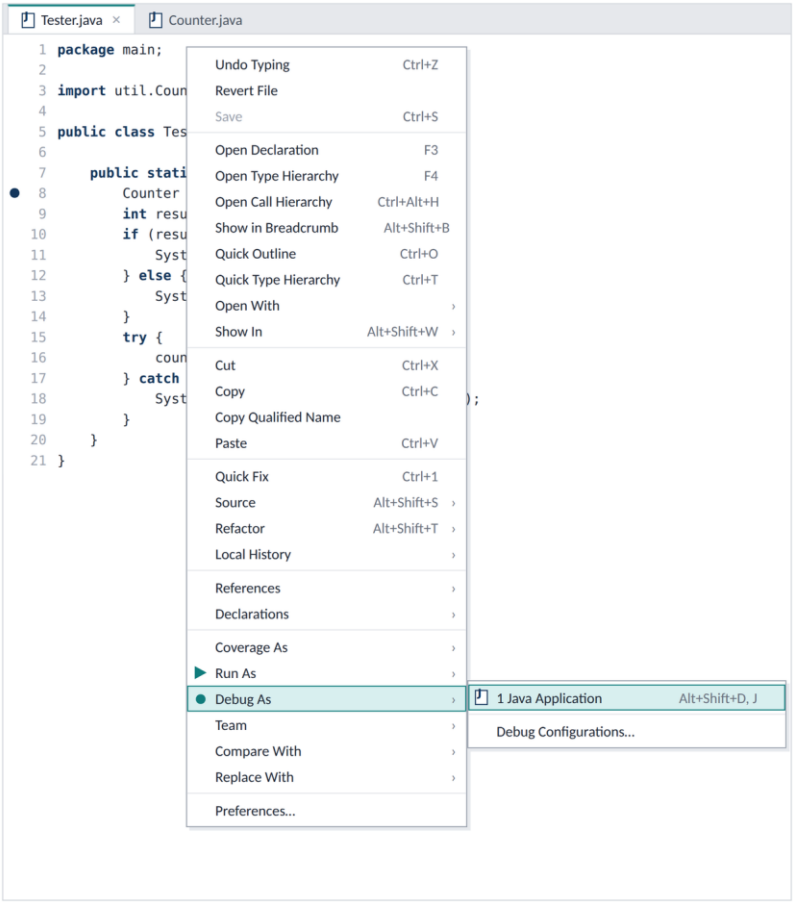
Gambar 2.30 Toggle Breakpoint pada menu konteks ruler kiri editor (Ctrl+Shift+B).



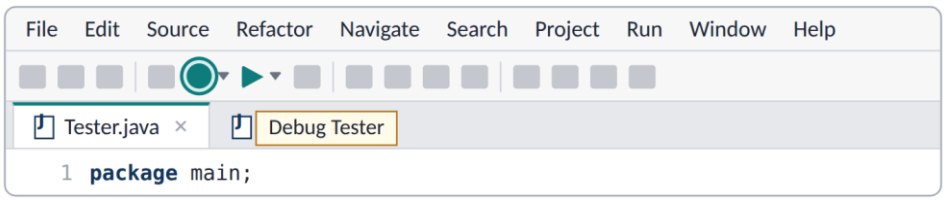
Gambar 2.31 Sebuah breakpoint pada baris 8, Counter counter = new Counter(); titik di ruler menandainya.

Menjalankan program secara biasa mengabaikan *breakpoint*. Untuk menjalankannya *di bawah debugger*, klik kanan kelasnya (atau di dalam editornya) lalu pilih **Debug As > Java Application**

(**Gambar 2.32**). Setelah Anda melakukannya sekali, tombol **Debug** berbentuk serangga pada bilah alat memulai konfigurasi peluncuran yang sama lagi (**Gambar 2.33**).

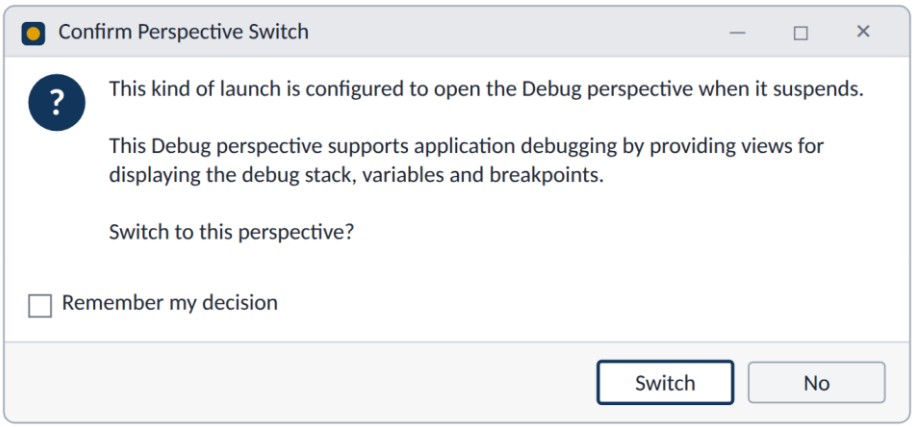


Gambar 2.32 Debug As > Java Application menjalankan program di bawah debugger.



Gambar 2.33 Tombol Debug pada bilah alat menjalankan ulang peluncuran debug terakhir, di sini Tester.

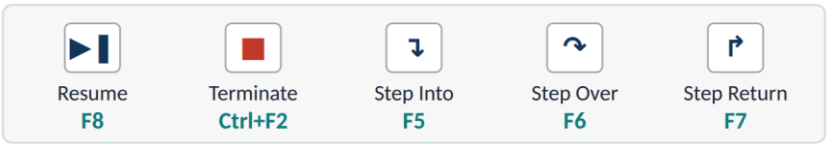
Ketika eksekusi mencapai *breakpoint*, Eclipse bertanya apakah ia perlu berpindah ke *perspective* Debug (**Gambar 2.34**). Jawablah **Switch** (dan centang *Remember my decision* agar ia berhenti bertanya). Bila Anda tidak memasang *breakpoint* sama sekali, program berjalan begitu saja sampai selesai, persis seperti dengan *Run As*: *debugger* hanya berhenti di tempat yang Anda minta.



Gambar 2.34 Eclipse menawarkan untuk membuka perspective Debug ketika program pertama kali tertangguhkan.

2.10.4 Mengendalikan eksekusi: *resume, step over, step into, step out*

Bilah alat *perspective* Debug memiliki tombol-tombol untuk mengendalikan program yang tertangguhkan, dan masing-masing memiliki tombol fungsi (**Gambar 2.35**). Jauh lebih cepat bila Anda menghafal tombol-tombolnya.



Gambar 2.35 Tombol eksekusi debugger Eclipse beserta tombol papan ketiknya: Resume (F8), Terminate (Ctrl+F2), Step Into (F5), Step Over (F6), dan Step Return (F7).

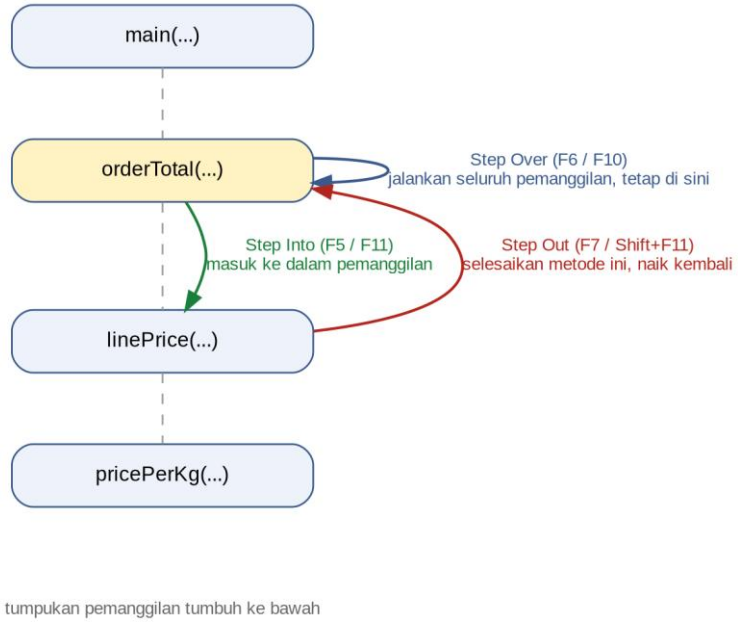
Tabel 2.4 memberikan arti setiap kendali beserta tombolnya di kedua IDE dan di dua *debugger* terminal yang dipakai pada Subbab 2.11, yaitu `jdb` untuk Java dan `pdb` untuk Python.

Tabel 2.4 Kendali eksekusi debugger beserta tombol dan perintahnya di Eclipse, VS Code, jdb, dan pdb.

Kendali	Apa yang dikerjakannya	Eclipse	VS Code	Jdb	Pdb
Step Into	Mengeksekusi baris saat ini; bila baris itu memanggil *method* buatan Anda, berhenti pada baris pertama di dalam *method* tersebut	F5	F11	`step`	`step`
Step Over	Mengeksekusi baris saat ini sepenuhnya, termasuk *method* apa pun yang dipanggilnya, lalu berhenti pada baris berikutnya dalam *method* yang sama	F6	F10	`next`	`next`
Step Out (Step Return)	Menuntaskan *method* saat ini dan berhenti di pemanggilnya, tepat sesudah pemanggilan	F7	Shift+F11	`step up`	`return`
Resume (Continue)	Berjalan sampai *breakpoint* berikutnya, atau sampai selesai	F8	F5	`cont`	`continue`
Terminate (Stop)	Mengakhiri program seketika	Ctrl+F2	Shift+F5	`exit`	`quit`

Ketiga langkah itu paling mudah dipahami sebagai gerakan pada tumpukan pemanggilan, seperti pada **Gambar 2.36**. Misalkan program tertangguhkan di dalam *method* `orderTotal` pada baris yang memanggil `linePrice(...)`, yang selanjutnya memanggil `pricePerKg(...)`. **Step Over** memperlakukan pemanggilan itu sebagai satu tindakan: `linePrice` dan semua yang dipanggilnya berjalan sampai tuntas, dan program berhenti pada baris berikutnya dari `orderTotal`. **Step Into** turun *ke dalam* pemanggilan dan berhenti pada baris pertama `linePrice`. **Step Out** melakukan

kebalikannya: ia membiarkan *method* saat ini selesai dan berhenti kembali di *method* yang memanggilnya.



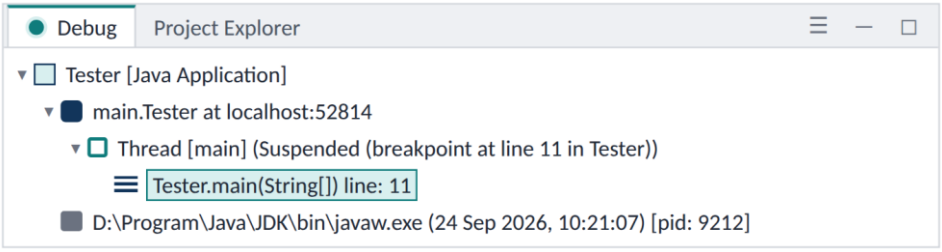
Gambar 2.36 Langkah-langkah debugger dilihat pada tumpukan pemanggilan. Step Into turun ke method yang dipanggil, Step Out kembali ke pemanggil, dan Step Over menjalankan seluruh pemanggilan dan tetap pada tingkat yang sama.

Pegangan praktisnya: lakukan *step over* pada kode yang Anda percayai, *step into* pada kode yang Anda curigai, dan *step out* begitu Anda sadar telah masuk ke sesuatu yang tidak menarik. *Method* pustaka seperti `Math.max` atau `System.out.println` biasanya dilewati oleh Step Into, karena kedua IDE menerapkan *step filter* yang menjauhkan Anda dari pustaka Java dan, pada *debugger* Python di VS Code, dari kode yang bukan milik Anda (pengaturan `justMyCode`). Hampir selalu itulah yang Anda inginkan.

2.10.5 Tumpukan pemanggilan

> **DEFINISI.** *Tumpukan pemanggilan* (*call stack*) adalah daftar pemanggilan *method* yang sudah dimulai tetapi belum selesai. *Method* yang sedang dieksekusi berada paling atas; di bawahnya ada *method* yang memanggilnya, lalu yang memanggil *method* itu, terus sampai `main` (dan, di bawah `main`, kode awal milik *runtime* itu sendiri).

Eclipse menampilkan tumpukan pemanggilan di *view* **Debug**, di kiri atas *perspective* Debug (**Gambar 2.37**). Pohonnya dibaca sebagai berikut: peluncurannya (`Tester [Java Application]`), JVM yang dijalankannya, *thread* `main` (tertangguhkan), dan di bawah *thread* itu satu baris untuk setiap *frame* (bingkai) tumpukan, di sini hanya `Tester.main(String[]) line: 11`. Mengklik *frame* yang lebih bawah menampilkan, di editor dan di *view* Variables, baris dan variabel *method itu* pada saat ia melakukan pemanggilannya; dengan cara inilah Anda mengetahui *mengapa* sebuah *method* dipanggil dengan argumen yang diterimanya.

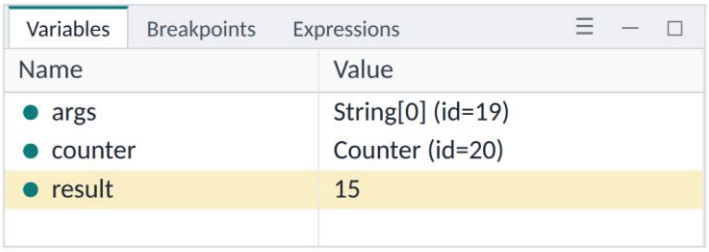


Gambar 2.37 View Debug menampilkan thread main yang tertangguhkan di Tester.main pada baris 11; setiap baris di bawah thread adalah satu frame tumpukan pemanggilan.

Ia disebut *tumpukan* karena berperilaku seperti tumpukan piring: setiap pemanggilan menaruh satu *frame* di atas, setiap pengembalian mengambil *frame* teratas, dan hanya yang teratas yang aktif.

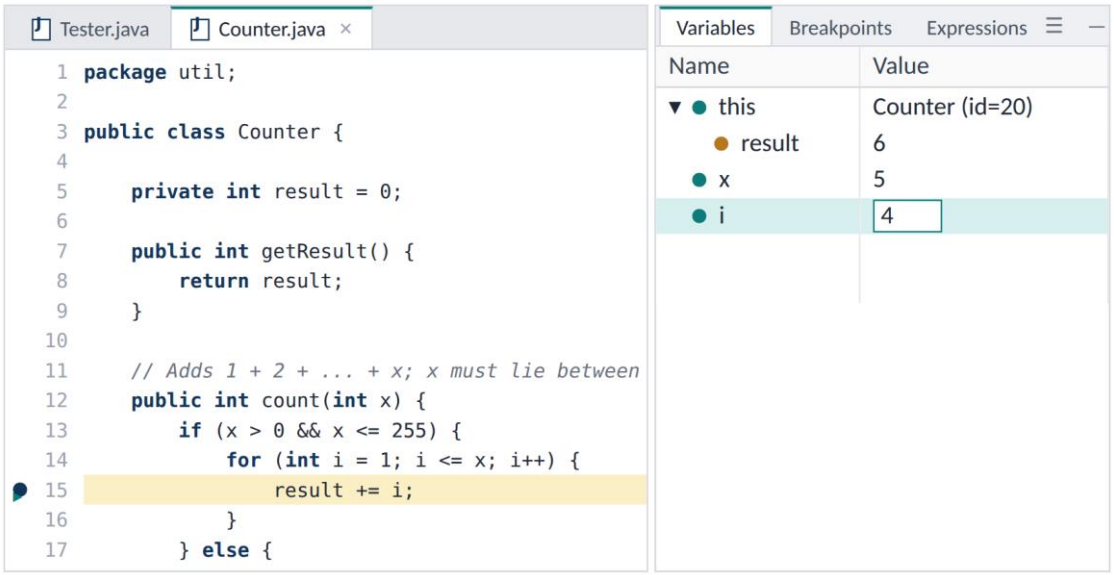
2.10.6 Melihat variabel, dan mengubahnya

View **Variables**, di kanan atas *perspective* Debug, mendaftar parameter dan variabel lokal dari *frame* yang dipilih selama program tertangguhkan (**Gambar 2.38**). Nilai yang diubah oleh langkah terakhir disorot, cara tercepat untuk melihat apa yang dikerjakan sebuah langkah, dan membuka sebuah objek seperti `counter` memperlihatkan *field-field*-nya. Menu *view* itu juga dapat menampilkan variabel statis dan, melalui *Layout > Select Columns…*, *tipe aktual* setiap nilai, yang berguna mulai Bab 7.



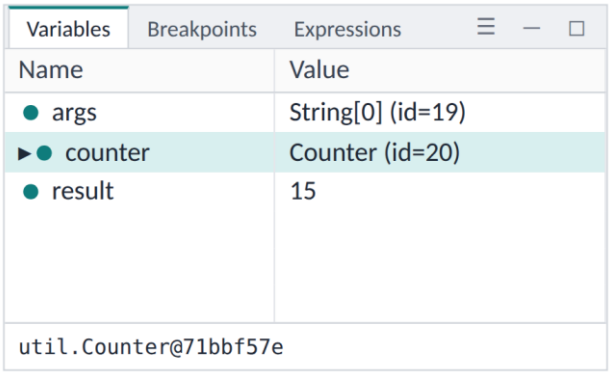
Gambar 2.38 View Variables sesudah count(5) mengembalikan nilainya: args, counter, dan result = 15.

Anda juga dapat *mengubah* sebuah nilai selama program tertangguhkan: klik nilai itu di *view* Variables, ketik nilai baru, lalu tekan Enter (**Gambar 2.39**). Di VS Code, klik kanan sebuah variabel lalu pilih *Set Value*; di `pdb` cukup ketik sebuah penugasan seperti `!total = 0`. Dengan begitu Anda dapat mengajukan pertanyaan "bagaimana jika?" tanpa menyunting program: bagaimana jika pesanan itu bukan ekspres, bagaimana jika pelanggannya bukan anggota? Perubahan itu hanya berlaku untuk eksekusi kali ini.

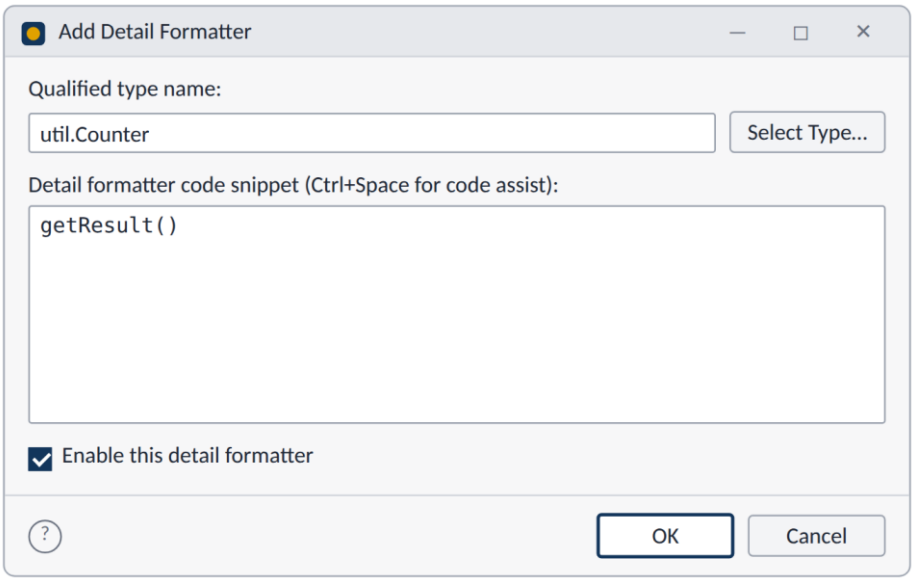


Gambar 2.39 Mengubah nilai selama debugging: variabel gelung i di dalam count(int) sedang disunting di view Variables.

Bagaimana sebuah objek ditampilkan. Secara bawaan *view* Variables menampilkan objek memakai *method* `toString()`-nya. Untuk kelas yang tidak mendefinisikannya, hasilnya kira-kira `util.Counter@71bbf57e`, nama kelas dan sebuah bilangan, yang tidak memberi tahu apa-apa (**Gambar 2.40**). Bab 3 memperlihatkan cara menulis `toString()`; sampai saat itu, Eclipse memungkinkan Anda mendefinisikan **Detail Formatter**: klik kanan variabelnya, pilih *New Detail Formatter...*, lalu ketik ungkapan Java yang ditampilkan sebagai gantinya, misalnya `getResult()` (**Gambar 2.41**). Sejak itu setiap `Counter` ditampilkan sebagai hasilnya.



Gambar 2.40 Tanpa method toString(), panel detail hanya menampilkan util.Counter@71bbf57e.



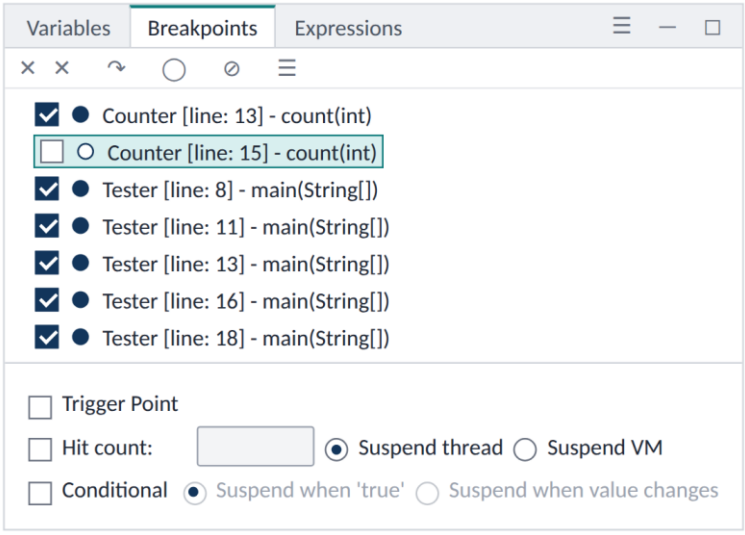
Gambar 2.41 Detail Formatter untuk util.Counter yang menampilkan getResult() sebagai gantinya.

***Watch* dan ungkapan.** *View Expressions* (Eclipse) dan bagian *Watch* (VS Code) menyimpan ungkapan-ungkapan yang dievaluasi ulang pada setiap perhentian. Klik kanan sebuah variabel atau ungkapan yang diseleksi di editor lalu pilih *Watch*, atau klik **+** di bagian Watch lalu ketik, misalnya, `total * 50 / 100`. Subbab 2.11 memakai sebuah *watch* untuk melihat nilai yang salah sebelum nilai itu ditambahkan ke total.

2.10.7 Mengelola *breakpoint*: menonaktifkan, melewati, *hit count*, dan kondisi

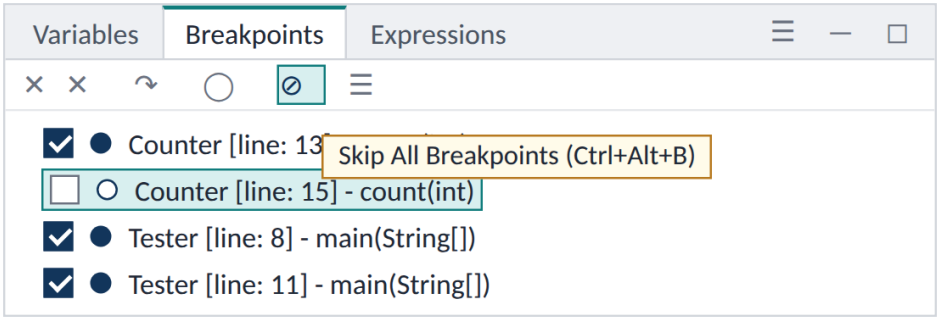
Setelah satu jam men-debug, Anda mungkin memiliki selusin *breakpoint* di beberapa berkas. *View* **Breakpoints** (**Gambar 2.42**) mendaftar semuanya. Menghapus centang pada kotak sebuah *breakpoint* akan *menonaktifkannya* tanpa menghapusnya, sehingga Anda dapat menyalakannya kembali nanti; tombol-tombol **X** menghapus *breakpoint* yang dipilih atau semuanya. Bagian bawah *view* itu menampilkan properti *breakpoint* yang dipilih, dan dua di antaranya sangat ampuh:

- ***Hit count***: berhenti hanya pada kali ke-*n* baris itu dicapai. Dalam gelung yang berjalan 1.000 kali, *hit count* 500 membawa Anda langsung ke tengahnya.
- ***Conditional*** (bersyarat): berhenti hanya bila sebuah ungkapan Boolean yang Anda ketik bernilai benar, misalnya `i == 7` atau `billedKg < kg`. Subbab 2.12 dikhususkan untuk hal ini.



Gambar 2.42 View Breakpoints mendaftar setiap breakpoint; kotak centangnya menyalakan atau mematikan breakpoint, dan bagian bawahnya memuat properti Hit count dan Conditional.

Tombol **Skip All Breakpoints** (Ctrl+Alt+B) pada bilah alat *view* yang sama (**Gambar 2.43**) mematikan semua *breakpoint* sekaligus; menekannya lagi menyalakan semuanya kembali. Tombol ini berguna ketika Anda ingin program berjalan bebas untuk sementara tanpa kehilangan *breakpoint* yang sudah Anda tempatkan dengan cermat.



Gambar 2.43 Skip All Breakpoints (Ctrl+Alt+B) menonaktifkan setiap breakpoint tanpa menghapus satu pun.

2.10.8 *Debugger* yang sama di VS Code, untuk Java dan untuk Python

Debugger VS Code bekerja dengan cara yang sama untuk kedua bahasa, dan **Gambar 2.23** memperlihatkan bagian-bagiannya [3], [4].

- **Memasang *breakpoint*.** Klik pada *gutter* di sebelah kiri nomor baris; sebuah titik merah muncul. F9 memasang atau melepas *breakpoint* pada baris tempat kursor berada.
- **Memulai.** Untuk Java, klik CodeLens **Debug** di atas `main`, atau tekan F5. Untuk Python, pilih *Python Debugger: Debug Python File* dari daftar tarik-turun di samping segitiga Run di kanan atas editor, atau tekan F5 lalu pilih *Python File*. Bila ada `launch.json`, F5 memulai konfigurasi yang sedang dipilih di dalamnya.
- ***View*-nya.** Bilah samping **Run and Debug** menampilkan bagian *Variables*, *Watch*, *Call Stack*, dan *Breakpoints*, yang berpadanan satu lawan satu dengan *view* Variables, Expressions, Debug, dan Breakpoints di Eclipse. Bagian Breakpoints juga memiliki kotak centang untuk berhenti pada *Uncaught Exceptions* dan *Raised Exceptions* (Python) atau *Uncaught Exceptions* dan *Caught Exceptions* (Java).

- **Bilah alat.** Sebuah bilah alat melayang muncul di bagian atas editor dengan Continue (F5), Step Over (F10), Step Into (F11), Step Out (Shift+F11), Restart, dan Stop.
- ***Debug Console*** mengevaluasi ungkapan apa pun yang Anda ketik, misalnya `line * 50 // 100`, di dalam *frame* saat ini.
- ***Breakpoint* bersyarat.** Klik kanan titik *breakpoint* (atau *gutter*) lalu pilih *Edit Breakpoint…* atau *Add Conditional Breakpoint…*; pilih *Expression*, *Hit Count*, atau *Log Message*. *Logpoint* (pesan log) mencetak pesan seperti `total is {total}` setiap kali baris itu dicapai, tanpa berhenti: sebuah pernyataan cetak yang tidak pernah perlu dihapus dari kode sumber.

> **WAWASAN KUNCI:** Setiap *debugger* grafis, di setiap IDE, untuk setiap bahasa, menawarkan seperangkat kecil kendali yang sama: *breakpoint* (biasa, bersyarat, *hit count*), *resume*, *step over*, *step into*, *step out*, variabel, *watch*, dan tumpukan pemanggilan. Pelajarilah gagasannya sekali saja, pada satu alat, dan Anda akan dapat memakai *debugger* mana pun dalam hitungan menit.

2.11 Perburuan Terbimbing: Tagihan Cucian yang Terlalu Mahal

2.11.1 Gejalanya

Profitina Laundry, usaha binatu kecil yang menyertai buku ini dari awal sampai akhir, menarik biaya berdasarkan berat. Daftar harganya sederhana:

- *wash* (cuci): Rp 6.000 per kg; *iron* (setrika): Rp 4.000 per kg; *wash+iron* (cuci dan setrika): Rp 9.000 per kg;
- cucian yang lebih ringan dari 3 kg ditagih sebagai 3 kg (mesinnya tetap berputar);
- layanan *express* (ekspres) 50% lebih mahal untuk butir cucian tersebut;
- anggota yang tagihannya mencapai Rp 100.000 mendapat potongan 10% dari seluruh tagihan.

Bila ditulis sebagai rumus, harga butir ke-i, dengan tarif layanan $r(s_i)$, berat w_i kilogram, dan $e_i = 1$ bila butir itu ekspres serta $e_i = 0$ bila tidak, adalah

$$p_i = \left(1 + \frac{e_i}{2}\right) \cdot r(s_i) \cdot \max(w_i, 3), \tag{2.2}$$

dan tagihan untuk n butir adalah

$$T = \begin{cases} \frac{9}{10} P & \text{jika pelanggan adalah anggota dan } P \geq 100\,000, \\ P & \text{selain itu,} \end{cases} \qquad \text{denga} \quad \sum_{i=1}^{n} p_i. \tag{2.3}$$

Bu Sari, seorang anggota, membawa tiga butir cucian: 4 kg untuk dicuci, 2 kg untuk disetrika, dan 5 kg untuk dicuci dan disetrika, yang terakhir ini ekspres. Program kasir, **Kode 2.9** dalam Java dan **Kode 2.10** dalam Python, menghitung tagihannya. Program itu hanya memakai apa yang sudah diperkenalkan Bab 1 dan bab ini: konstanta, beberapa *method* (fungsi di Python), larik (*list* di Python) yang memuat satu entri untuk setiap butir, dan sebuah gelung. Program mencetak total yang dihitungnya dan, sebagai pembanding, total yang menurut struk Bu Sari minggu lalu seharusnya muncul.

Kode 2.9 `LaundryPrice.java`: kalkulator harga kasir, lengkap dengan sebuah bug.

```java
// LaundryPrice.java: menghitung harga satu pesanan Profitina Laundry. Versi ini mengandung bug.
public class LaundryPrice {

    static final double MIN_KG = 3.0;            // cucian yang lebih ringan ditagih 3 kg
    static final long EXPRESS_PERCENT = 50;      // layanan ekspres 50% lebih mahal
    static final long MEMBER_DISCOUNT_PERCENT = 10;
    static final long DISCOUNT_THRESHOLD = 100_000;

    static long pricePerKg(String service) {
        return switch (service) {
            case "wash" -> 6_000;
            case "iron" -> 4_000;
            case "wash+iron" -> 9_000;
            default -> throw new IllegalArgumentException("unknown service: " + service);
        };
    }

    static long linePrice(String service, double kg) {
        double billedKg = Math.max(kg, MIN_KG);
        return (long) (billedKg * pricePerKg(service));
    }

    static long orderTotal(String[] services, double[] kgs, boolean[] express, boolean member) {
        long total = 0;
        for (int i = 0; i < services.length; i++) {
            long line = linePrice(services[i], kgs[i]);
            total += line;
            if (express[i]) {
                total += total * EXPRESS_PERCENT / 100;
            }
        }
        if (member && total >= DISCOUNT_THRESHOLD) {
            total -= total * MEMBER_DISCOUNT_PERCENT / 100;
        }
        return total;
    }
```

Kode 2.9 (Lanjutan) *LaundryPrice.java: kalkulator harga kasir, lengkap dengan sebuah bug.*

```java
    public static void main(String[] args) {
        String[] services = {"wash", "iron", "wash+iron"};
        double[] kgs = {4.0, 2.0, 5.0};
        boolean[] express = {false, false, true};
        long total = orderTotal(services, kgs, express, true);
        System.out.println("Customer : Bu Sari (member)");
        System.out.println("Total    : Rp " + total);
        System.out.println("Expected : Rp 93150");
    }
}
```

```
Customer : Bu Sari (member)
Total    : Rp 109350
Expected : Rp 93150
```

Kode 2.10 `laundry_price.py`: kalkulator yang sama dalam Python, dengan bug yang sama.

```
"""laundry_price.py: menghitung harga satu pesanan Profitina Laundry. Versi ini mengandung
bug."""

MIN_KG = 3.0                      # cucian yang lebih ringan ditagih 3 kg
EXPRESS_PERCENT = 50              # layanan ekspres 50% lebih mahal
MEMBER_DISCOUNT_PERCENT = 10
DISCOUNT_THRESHOLD = 100_000


def price_per_kg(service):
    if service == "wash":
        return 6_000
    elif service == "iron":
        return 4_000
    elif service == "wash+iron":
        return 9_000
    raise ValueError("unknown service: " + service)


def line_price(service, kg):
    billed_kg = max(kg, MIN_KG)
    return int(billed_kg * price_per_kg(service))


def order_total(services, kgs, express, member):
    total = 0
    for i in range(len(services)):
        line = line_price(services[i], kgs[i])
        total += line
        if express[i]:
            total += total * EXPRESS_PERCENT // 100
    if member and total >= DISCOUNT_THRESHOLD:
        total -= total * MEMBER_DISCOUNT_PERCENT // 100
    return total


def main():
    services = ["wash", "iron", "wash+iron"]
    kgs = [4.0, 2.0, 5.0]
    express = [False, False, True]
    total = order_total(services, kgs, express, True)
    print("Customer : Bu Sari (member)")
    print("Total    : Rp", total)
    print("Expected : Rp 93150")


if __name__ == "__main__":
    main()
```

Kode 2.10 (Lanjutan) *laundry_price.py: kalkulator yang sama dalam Python, dengan bug yang sama.*

```
Customer : Bu Sari (member)
Total    : Rp 109350
Expected : Rp 93150
```

Kedua program saling sepakat, dan keduanya salah: Bu Sari ditagih Rp 109.350, bukan Rp 93.150, kelebihan tagih sebesar Rp 16.200. Inilah jenis bug yang paling buruk, diam dan masuk akal, dan bug ini merugikan pelanggan sungguhan dengan uang sungguhan.

Beberapa kata tentang kodenya sebelum kita berburu. `static final` di Java (dan, menurut konvensi, nama berhuruf kapital di Python) menandai sebuah *konstanta*, yaitu nilai yang tidak pernah berubah selama program berjalan. `switch` di dalam `pricePerKg` memilih nilai kembalian dengan membandingkan `service` dengan setiap `case`; cabang `default`-nya melempar eksepsi

untuk layanan yang tidak dikenal, yang akan sengaja kita picu di Subbab 2.13. Garis bawah dalam `100_000` hanya untuk memudahkan pembacaan; nilainya seratus ribu. `Math.max(kg, MIN_KG)` (Python: `max(kg, MIN_KG)`) adalah yang lebih besar di antara keduanya, dan itulah yang mewujudkan batas minimum 3 kg. *Cast* `(long)` (Python: `int(...)`) mengubah hasil kali bertipe `double` menjadi bilangan bulat rupiah; untuk berat yang dinyatakan dalam kelipatan setengah kilogram, seperti di sini, hasil kalinya selalu bilangan bulat, sehingga tidak ada yang hilang.

2.11.2 Langkah 1: ketahui jawaban yang seharusnya

Anda tidak dapat mengenali nilai yang salah bila tidak mengetahui nilai yang benar. Sebelum menyentuh *debugger*, hitunglah tagihan itu secara manual dengan Persamaan (2.2) dan Persamaan (2.3), satu baris untuk setiap butir, seperti pada **Tabel 2.5**.

Tabel 2.5 Tagihan Bu Sari yang dihitung secara manual. Inilah nilai-nilai yang harus dicapai program, butir demi butir.

Butir	Layanan	Berat w_i	Kg ditagih $\max(w_i,3)$	Tarif $r(s_i)$	Harga dasar	Ekspres?	p_i
1	wash	4,0	4,0	6.000	24.000	tidak	24.000
2	iron	2,0	3,0	4.000	12.000	tidak	12.000
3	wash+iron	5,0	5,0	9.000	45.000	ya, +22.500	67.500
						P	103.500
						potongan 10%	−10,350
						T	93.150

Tabel itu memberi lebih dari sekadar jawaban di baris terbawah: ia memberi nilai-nilai *antara* yang benar. Sesudah butir 1, total berjalan seharusnya 24.000; sesudah butir 2, 36.000; sesudah butir 3, $36{,}000 + 67{,}500 = 103{,}500$. Tempat mana pun total berjalan program menyimpang dari kolom ini, di situlah bug-nya berada.

2.11.3 Langkah 2: susun hipotesis, lalu pasang *breakpoint*

Total yang salah itu terlalu *tinggi*, sebesar Rp 16.200. Apa yang dapat membuat tagihan terlalu tinggi? Potongannya mungkin tidak diterapkan (tetapi potongan yang terlewat akan menghasilkan 103.500, bukan 109.350); berat minimum mungkin diterapkan secara keliru; biaya tambahan ekspres mungkin terlalu besar. Kita belum tahu, dan itu wajar: hipotesis hanyalah dugaan tentang *di mana harus mencari*. Gelung di dalam `orderTotal` adalah tempat semua harga butir bertemu, jadi di sanalah kita memasang *breakpoint*: pada baris yang memanggil `linePrice` (baris 26 di Java, baris 27 di Python).

Di Eclipse, klik ganda pada *ruler* baris 26 lalu pilih *Debug As > Java Application*; di VS Code, klik *gutter* baris 26 (Java) atau 27 (Python) lalu tekan F5. Program berhenti sebelum pemanggilan itu, dengan `i = 0` dan `total = 0` di *view* Variables.

2.11.4 Langkah 3: tur terbimbing dengan *step into*, *step over*, dan *step out*

Pertama-tama kita memeriksa bahwa `linePrice` menghargai butir 1 dengan benar, sambil berlatih ketiga langkah di sepanjang jalan: Step Into tiga kali (masuk ke `linePrice`, melewati pemanggilan pustaka `Math.max`, masuk ke `pricePerKg`), Step Out dua kali (kembali ke `linePrice`, kembali ke `orderTotal`), dan Step Over sekali untuk menuntaskan penugasannya. Di IDE Anda

menekan F5, F5, F5, F7, F7, F6 di Eclipse atau F11, F11, F11, Shift+F11, Shift+F11, F10 di VS Code, sambil mengamati *view* Variables dan tumpukan pemanggilan sesudah setiap tekanan.

Tangkapan layar hanya dapat memperlihatkan satu saat dari proses ini, tetapi JDK dan Python sama-sama menyertakan *debugger terminal* yang sederhana, `jdb` untuk Java [6] dan `pdb` untuk Python [11], yang mencetak setiap langkah sebagai teks. Berikut `jdb` (kompilasi dengan `javac -g LaundryPrice.java` agar berkas kelasnya menyimpan nama-nama variabel lokal, lalu jalankan `jdb -classpath . LaundryPrice`); perintah sesudah *prompt* `>` dan `main[1]` adalah apa yang diketik:

```
Initializing jdb ...
> stop at LaundryPrice:26
Deferring breakpoint LaundryPrice:26.
It will be set after the class is loaded.
> run
run LaundryPrice
Set uncaught java.lang.Throwable
Set deferred uncaught java.lang.Throwable
>
VM Started: Set deferred breakpoint LaundryPrice:26

Breakpoint hit: "thread=main", LaundryPrice.orderTotal(), line=26 bci=13
26                long line = linePrice(services[i], kgs[i]);

main[1] print i
 i = 0
main[1] step
>
Step completed: "thread=main", LaundryPrice.linePrice(), line=19 bci=0
19            double billedKg = Math.max(kg, MIN_KG);

main[1] locals
Method arguments:
service = "wash"
kg = 4.0
Local variables:
main[1] step
>
Step completed: "thread=main", LaundryPrice.linePrice(), line=20 bci=8
20            return (long) (billedKg * pricePerKg(service));

main[1] step
>
Step completed: "thread=main", LaundryPrice.pricePerKg(), line=10 bci=0
10            return switch (service) {

main[1] where
  [1] LaundryPrice.pricePerKg (LaundryPrice.java:10)
  [2] LaundryPrice.linePrice (LaundryPrice.java:20)
  [3] LaundryPrice.orderTotal (LaundryPrice.java:26)
  [4] LaundryPrice.main (LaundryPrice.java:42)
main[1] step up
>
Step completed: "thread=main", LaundryPrice.linePrice(), line=20 bci=13
20            return (long) (billedKg * pricePerKg(service));

main[1] step up
>
Step completed: "thread=main", LaundryPrice.orderTotal(), line=26 bci=24
26                long line = linePrice(services[i], kgs[i]);
```

```
main[1] next
>
Step completed: "thread=main", LaundryPrice.orderTotal(), line=27 bci=26
27                total += line;

main[1] print line
 line = 24000
main[1] clear LaundryPrice:26
Removed: breakpoint LaundryPrice:26
main[1] cont
> Customer : Bu Sari (member)
Total    : Rp 109350
Expected : Rp 93150

The application exited
```

Ikutilah baris demi baris. `stop at LaundryPrice:26` memasang *breakpoint*; `jdb` menyatakannya *deferred* (ditangguhkan) karena kelasnya belum dimuat. `run` memulai program, yang berhenti di baris 26 dengan `i = 0`. `step` yang pertama masuk ke `linePrice` di baris 19, dan `locals` menampilkan kedua parameternya dan belum ada variabel lokal. `step` berikutnya menjalankan baris 19 (tanpa masuk ke `Math.max`), yang sesudahnya masuk ke `pricePerKg` di baris 10, dan `where` mencetak tumpukan pemanggilan, *frame* teratas lebih dulu: tepat keempat *frame* pada **Gambar 2.36**, beserta baris yang telah dicapai masing-masing. Dua perintah `step up` memanjat keluar kembali, satu *frame* setiap kali, dan `next` menuntaskan baris 26, sesudah itu `print line` menampilkan 24000. Akhirnya `clear` melepas *breakpoint* dan `cont` membiarkan program berjalan sampai selesai. (*Prompt* `>` yang tersasar pada baris-baris yang selebihnya kosong adalah `jdb` yang mencetak *prompt*-nya sebelum peristiwa yang ditunggunya tiba; semua itu tidak berbahaya.)

Tur yang sama di `pdb`, dimulai dengan `python3 -m pdb laundry_price.py`, dengan perintah yang diketik sesudah `(Pdb)`:

```
> /home/student/laundry/laundry_price.py(1)<module>()
-> """laundry_price.py: prices one Profitina Laundry order. This version has a bug."""
(Pdb) break 27
Breakpoint 1 at /home/student/laundry/laundry_price.py:27
(Pdb) continue
> /home/student/laundry/laundry_price.py(27)order_total()
-> line = line_price(services[i], kgs[i])
(Pdb) p i, total
(0, 0)
(Pdb) step
--Call--
> /home/student/laundry/laundry_price.py(19)line_price()
-> def line_price(service, kg):
(Pdb) step
> /home/student/laundry/laundry_price.py(20)line_price()
-> billed_kg = max(kg, MIN_KG)
(Pdb) p service, kg
('wash', 4.0)
(Pdb) step
> /home/student/laundry/laundry_price.py(21)line_price()
-> return int(billed_kg * price_per_kg(service))
(Pdb) p billed_kg
4.0
(Pdb) step
--Call--
> /home/student/laundry/laundry_price.py(9)price_per_kg()
-> def price_per_kg(service):
(Pdb) where
  /usr/lib/python3.11/bdb.py(600)run()
-> exec(cmd, globals, locals)
  <string>(1)<module>()
  /home/student/laundry/laundry_price.py(47)<module>()
-> main()
  /home/student/laundry/laundry_price.py(40)main()
-> total = order_total(services, kgs, express, True)
  /home/student/laundry/laundry_price.py(27)order_total()
-> line = line_price(services[i], kgs[i])
  /home/student/laundry/laundry_price.py(21)line_price()
-> return int(billed_kg * price_per_kg(service))
> /home/student/laundry/laundry_price.py(9)price_per_kg()
-> def price_per_kg(service):
(Pdb) return
--Return--
> /home/student/laundry/laundry_price.py(11)price_per_kg()->6000
-> return 6_000
(Pdb) return
--Return--
> /home/student/laundry/laundry_price.py(21)line_price()->24000
-> return int(billed_kg * price_per_kg(service))
(Pdb) next
> /home/student/laundry/laundry_price.py(28)order_total()
-> total += line
(Pdb) p line
24000
(Pdb) quit
```

`pdb` berhenti sebelum baris pertama berkas, jadi kita memasang *breakpoint* di baris 27 lalu `continue` untuk mencapainya. `step` masuk ke `line_price` (pdb mula-mula menampilkan `--Call--` pada baris `def`, lalu pernyataan pertamanya). `p service, kg` mencetak nilai, dan `p billed_kg` sesudah langkah berikutnya menampilkan 4.0. `step` berikutnya masuk ke `price_per_kg`, dan `where` menampilkan tumpukannya, kali ini *frame tertua lebih dulu*, dengan *frame* saat ini ditandai `>` di bagian bawah. (Dua *frame* teratas, di `bdb.py` dan `<string>`, milik `pdb` sendiri.) Perintah *step out* pada `pdb` adalah `return`: ia berjalan sampai fungsi saat ini hendak mengembalikan nilai dan

menampilkan nilai itu, `->6000` lalu `->24000`. `next` menuntaskan penugasannya dan `p line` memastikan 24000.

Butir 1 dihargai dengan benar. Untuk butir ini bug-nya tidak berada di `linePrice`, dan kita sudah mengenal letak segala sesuatunya. Sekarang kita berburu.

2.11.5 Langkah 4: amati total berjalan

Pindahkan *breakpoint* ke baris `total += line;` (baris 27 di Java, baris 28 di Python), sehingga program berhenti sekali untuk setiap butir, tepat sebelum butir itu ditambahkan. Tambahkan dua *watch*: `total`, dan ungkapan `total * EXPRESS_PERCENT / 100` (di Python, `total * EXPRESS_PERCENT // 100`), yaitu biaya tambahan yang akan ditambahkan program bila sebuah butir ekspres. Sekarang tekan **Resume** (F8 di Eclipse, F5 di VS Code) untuk berpindah dari butir ke butir, dan bandingkan setiap nilai dengan **Tabel 2.5**.

- Perhentian 1: `line = 24000`, `total = 0`. Benar.
- Perhentian 2: `line = 12000`, `total = 24000`. Benar: setrika 2 kg ditagih sebagai 3 kg.
- Perhentian 3: `line = 45000`, `total = 36000`, dan `express[i]` bernilai `true`. Sejauh ini benar. Sekarang lakukan **step over** satu baris setiap kali. Sesudah `total += line`, `total = 81000`: masih konsisten, karena harga dasar butir 3 adalah 45.000. Baris berikutnya menguji `express[i]` dan masuk ke cabangnya. *Watch* menampilkan biaya tambahan yang akan ditambahkan: $81{,}000 \times 50/100 = 40{,}500$. Padahal **Tabel 2.5** menyatakan bahwa biaya tambahan untuk butir 3 adalah $45{,}000 \times 50/100 = 22{,}500$. Satu *step over* lagi, dan `total` melonjak menjadi 121.500, bukan 103.500.

Itulah dia. Biaya tambahan dihitung dari `total`, total berjalan *semua* butir sejauh ini, bukan dari `line`, harga *butir ini*. Butir 1 dan 2 ikut dikenai biaya tambahan ekspres, padahal keduanya tidak ekspres: $36{,}000 \times 50/100 = 18{,}000$ terlalu banyak sebelum potongan, dan $18{,}000 \times 90/100 = 16{,}200$ terlalu banyak sesudahnya, persis sebesar kelebihan tagih yang kita amati. Bug itu terjelaskan sepenuhnya, sampai rupiah terakhir, dan dengan cara itulah Anda tahu bahwa inilah *sang* bug, bukan sekadar *sebuah* bug.

Bila ditulis sebagai rumus, gelung itu menghitung, untuk $i = 1, \ldots, n$ dimulai dari $t_0 = 0$,

$$t_i = (t_{i-1} + p'_i)\left(1 + \frac{e_i}{2}\right), \tag{2.4}$$

dengan $p'_i = r(s_i)\max(w_i, 3)$ adalah harga dasar tanpa biaya tambahan, sedangkan Persamaan (2.2) dan Persamaan (2.3) meminta $t_i = t_{i-1} + p'_i\left(1 + \frac{e_i}{2}\right)$. Faktor $\left(1 + \frac{e_i}{2}\right)$ diterapkan pada hal yang salah. Persamaan (2.4) juga memberi tahu kita kapan bug itu tidak terlihat: bila satu-satunya butir ekspres adalah butir *pertama*, $t_{i-1} = 0$ dan kedua rumus itu sepakat. Pengujian dengan satu butir ekspres saja tidak akan pernah menangkapnya, dan itulah sebabnya bug ini bertahan hidup.

Berikut perburuan itu di `jdb`, dengan *breakpoint* di baris 27; sesudah tiga perhentian kita melakukan *step over* melewati cabang ekspres dan mengamati `total` berubah:

```
Initializing jdb ...
> stop at LaundryPrice:27
Deferring breakpoint LaundryPrice:27.
It will be set after the class is loaded.
> run
run LaundryPrice
Set uncaught java.lang.Throwable
Set deferred uncaught java.lang.Throwable
>
VM Started: Set deferred breakpoint LaundryPrice:27

Breakpoint hit: "thread=main", LaundryPrice.orderTotal(), line=27 bci=26
27                total += line;

main[1] locals
Method arguments:
services = instance of java.lang.String[3] (id=449)
kgs = instance of double[3] (id=450)
express = instance of boolean[3] (id=451)
member = true
Local variables:
total = 0
i = 0
line = 24000
main[1] cont
>
Breakpoint hit: "thread=main", LaundryPrice.orderTotal(), line=27 bci=26
27                total += line;

main[1] print line
 line = 12000
main[1] print total
 total = 24000
main[1] cont
>
Breakpoint hit: "thread=main", LaundryPrice.orderTotal(), line=27 bci=26
27                total += line;

main[1] print i
 i = 2
main[1] print express[i]
 express[i] = true
main[1] next
>
Step completed: "thread=main", LaundryPrice.orderTotal(), line=28 bci=33
28                if (express[i]) {
```

```
main[1] print total
 total = 81000
main[1] next
>
Step completed: "thread=main", LaundryPrice.orderTotal(), line=29 bci=40
29                    total += total * EXPRESS_PERCENT / 100;

main[1] print total
 total = 81000
main[1] next
>
Step completed: "thread=main", LaundryPrice.orderTotal(), line=25 bci=55
25            for (int i = 0; i < services.length; i++) {

main[1] print total
 total = 121500
main[1] where
  [1] LaundryPrice.orderTotal (LaundryPrice.java:25)
  [2] LaundryPrice.main (LaundryPrice.java:42)
main[1] cont
> Customer : Bu Sari (member)
Total    : Rp 109350
Expected : Rp 93150

The application exited
```

`locals` pada perhentian pertama menampilkan semuanya sekaligus: keempat parameter (larik tampil sebagai `instance of double[3]` dengan sebuah id internal) serta variabel lokal `total`, `i`, dan `line`. Pada perhentian ketiga, `print i` dan `print express[i]` memastikan bahwa inilah butir yang ekspres, dan tiga perintah `next` membawa `total` dari 81000 (sesudah baris 27) ke 121500 (sesudah baris 29). `where` memastikan bahwa kita berada di `orderTotal`, yang dipanggil dari `main` di baris 42.

Dan di `pdb`, perintah `display` memainkan peran *watch*: setiap kali program berhenti, `pdb` mencetak ulang setiap ungkapan yang ditampilkan *yang nilainya berubah*, beserta nilai lamanya:

```
> /home/student/laundry/laundry_price.py(1)<module>()
-> """laundry_price.py: prices one Profitina Laundry order. This version has a bug."""
(Pdb) break 28
Breakpoint 1 at /home/student/laundry/laundry_price.py:28
(Pdb) continue
> /home/student/laundry/laundry_price.py(28)order_total()
-> total += line
(Pdb) display total
display total: 0
(Pdb) display line
display line: 24000
(Pdb) next
> /home/student/laundry/laundry_price.py(29)order_total()
-> if express[i]:
display total: 24000  [old: 0]
(Pdb) next
> /home/student/laundry/laundry_price.py(26)order_total()
-> for i in range(len(services)):
(Pdb) continue
> /home/student/laundry/laundry_price.py(28)order_total()
-> total += line
display line: 12000  [old: 24000]
(Pdb) next
> /home/student/laundry/laundry_price.py(29)order_total()
-> if express[i]:
display total: 36000  [old: 24000]
(Pdb) next
> /home/student/laundry/laundry_price.py(26)order_total()
-> for i in range(len(services)):
(Pdb) continue
> /home/student/laundry/laundry_price.py(28)order_total()
-> total += line
display line: 45000  [old: 12000]
(Pdb) p i, express[i]
(2, True)
(Pdb) next
> /home/student/laundry/laundry_price.py(29)order_total()
-> if express[i]:
display total: 81000  [old: 36000]
(Pdb) next
> /home/student/laundry/laundry_price.py(30)order_total()
-> total += total * EXPRESS_PERCENT // 100
(Pdb) next
> /home/student/laundry/laundry_price.py(26)order_total()
-> for i in range(len(services)):
display total: 121500  [old: 81000]
(Pdb) quit
```

Baris terakhir sesi itu sudah menjelaskan segalanya: `display total: 121500  [old: 81000]`.

2.11.6 Langkah 5: perbaiki, lalu periksa

Perbaikannya adalah menambahkan biaya tambahan pada harga butir itu sendiri sebelum butir itu ditambahkan ke total, persis seperti yang dinyatakan Persamaan (2.2). Pada **Kode 2.11** dan **Kode 2.12** hanya gelung di dalam `orderTotal` yang berubah: biaya tambahan kini mengalikan `line`, dan `total += line` pindah ke sesudahnya.

Kode 2.11 `LaundryPriceFixed.java`: biaya tambahan ekspres kini dikenakan pada harga butir itu sendiri.

```java
// LaundryPriceFixed.java: menghitung harga satu pesanan Profitina Laundry. Bug sudah
diperbaiki.
public class LaundryPriceFixed {

    static final double MIN_KG = 3.0;            // cucian yang lebih ringan ditagih 3 kg
    static final long EXPRESS_PERCENT = 50;      // layanan ekspres 50% lebih mahal
    static final long MEMBER_DISCOUNT_PERCENT = 10;
    static final long DISCOUNT_THRESHOLD = 100_000;

    static long pricePerKg(String service) {
        return switch (service) {
            case "wash" -> 6_000;
            case "iron" -> 4_000;
            case "wash+iron" -> 9_000;
            default -> throw new IllegalArgumentException("unknown service: " + service);
        };
    }

    static long linePrice(String service, double kg) {
        double billedKg = Math.max(kg, MIN_KG);
        return (long) (billedKg * pricePerKg(service));
    }

    static long orderTotal(String[] services, double[] kgs, boolean[] express, boolean
member) {
        long total = 0;
        for (int i = 0; i < services.length; i++) {
            long line = linePrice(services[i], kgs[i]);
            if (express[i]) {
                line += line * EXPRESS_PERCENT / 100;   // biaya tambahan hanya untuk baris
ini
            }
            total += line;
        }
        if (member && total >= DISCOUNT_THRESHOLD) {
            total -= total * MEMBER_DISCOUNT_PERCENT / 100;
        }
        return total;
    }
```

Kode 2.11 (Lanjutan) *LaundryPriceFixed.java: biaya tambahan ekspres kini dikenakan pada harga butir itu sendiri.*

```java
    public static void main(String[] args) {
        String[] services = {"wash", "iron", "wash+iron"};
        double[] kgs = {4.0, 2.0, 5.0};
        boolean[] express = {false, false, true};
        long total = orderTotal(services, kgs, express, true);
        System.out.println("Customer : Bu Sari (member)");
        System.out.println("Total    : Rp " + total);
        System.out.println("Expected : Rp 93150");
    }
}
```

```
Customer : Bu Sari (member)
Total    : Rp 93150
Expected : Rp 93150
```

Kode 2.12 `laundry_price_fixed.py`: perbaikan yang sama dalam Python.

```
"""laundry_price_fixed.py: menghitung harga satu pesanan Profitina Laundry. Bug sudah
diperbaiki."""

MIN_KG = 3.0                      # cucian yang lebih ringan ditagih 3 kg
EXPRESS_PERCENT = 50              # layanan ekspres 50% lebih mahal
MEMBER_DISCOUNT_PERCENT = 10
DISCOUNT_THRESHOLD = 100_000


def price_per_kg(service):
    if service == "wash":
        return 6_000
    elif service == "iron":
        return 4_000
    elif service == "wash+iron":
        return 9_000
    raise ValueError("unknown service: " + service)


def line_price(service, kg):
    billed_kg = max(kg, MIN_KG)
    return int(billed_kg * price_per_kg(service))


def order_total(services, kgs, express, member):
    total = 0
    for i in range(len(services)):
        line = line_price(services[i], kgs[i])
        if express[i]:
            line += line * EXPRESS_PERCENT // 100   # biaya tambahan hanya untuk baris ini
        total += line
    if member and total >= DISCOUNT_THRESHOLD:
        total -= total * MEMBER_DISCOUNT_PERCENT // 100
    return total


def main():
    services = ["wash", "iron", "wash+iron"]
    kgs = [4.0, 2.0, 5.0]
    express = [False, False, True]
    total = order_total(services, kgs, express, True)
    print("Customer : Bu Sari (member)")
    print("Total    : Rp", total)
    print("Expected : Rp 93150")
```

Kode 2.12 (Lanjutan) *laundry_price_fixed.py: perbaikan yang sama dalam Python.*

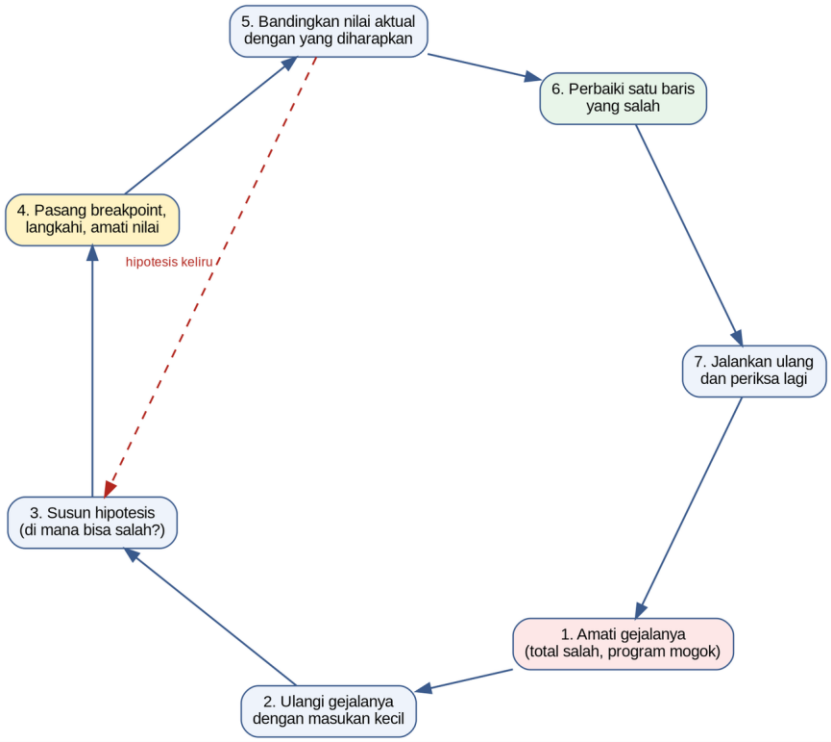
```
if __name__ == "__main__":
    main()
```

```
Customer : Bu Sari (member)
Total    : Rp 93150
Expected : Rp 93150
```

Kedua program kini mencetak Rp 93.150. Namun, jangan berhenti pada satu pengujian yang tadi gagal. Jalankan program yang sudah diperbaiki itu di bawah *debugger* sekali lagi dengan *watch* yang sama dan pastikan bahwa total berjalannya melewati 24.000, 36.000, dan 103.500, seperti yang dinyatakan **Tabel 2.5**; perbaikan yang menghasilkan total yang benar dengan alasan yang salah adalah bug yang sedang menunggu pelanggan berikutnya. Bab 6 memperlihatkan cara mengubah pemeriksaan semacam itu menjadi pengujian otomatis yang berjalan setiap kali kode berubah.

2.11.7 Metode di balik perburuan

Tengoklah kembali apa yang telah kita kerjakan, karena tujuh langkah yang sama berlaku untuk setiap bug diam yang akan Anda jumpai. **Gambar 2.44** menggambarkannya sebagai sebuah siklus.



Gambar 2.44 Siklus debugging. Bila perbandingan pada langkah 5 membantah hipotesis, kembalilah ke langkah 3 dengan membawa apa yang sudah Anda pelajari, bukan kembali menebak-nebak.

Amati gejalanya dengan cermat; ulangi gejala itu dengan masukan sekecil mungkin (di sini tiga butir); susun hipotesis tentang *di mana* harus mencari, dengan mengutamakan tempat-tempat nilai bertemu; pasang *breakpoint* dan melangkahlah; bandingkan setiap nilai yang teramati dengan nilai yang dihitung secara manual, karena perbedaan pertama itulah bug-nya; perbaiki satu baris yang salah dan jelaskan keluaran salah persis yang disebabkannya; jalankan ulang kasus yang gagal beserta kasus-kasus yang sebelumnya lulus. Bila perbandingan memperlihatkan semuanya benar di tempat yang Anda periksa, hipotesisnya keliru; itu tetap kemajuan, karena Anda telah menyingkirkan sebagian program dari kecurigaan. Pindahkan *breakpoint* dan ulangi. Pencarian yang sistematis seperti ini selalu berakhir, sedangkan menebak-nebak tidak [12], [13].

2.12 *Breakpoint* Bersyarat: Menemukan Jarum dalam Tumpukan Jerami

Breakpoint di dalam gelung berhenti pada *setiap* putaran. Dengan tiga butir, itu tidak masalah; dengan pesanan sehari, atau setahun, melangkahi setiap putaran adalah mustahil. ***Breakpoint* bersyarat** (*conditional breakpoint*) hanya berhenti bila sebuah ungkapan pilihan Anda bernilai benar.

Setiap akhir hari Profitina Laundry menjumlahkan pesanan hari itu. **Kode 2.13** dan **Kode 2.14** melakukannya untuk sepuluh pesanan. (Agar contohnya tetap kecil, pesanan-pesanan ini tidak memuat butir ekspres maupun potongan.) Totalnya kurang Rp 6.000 dari jumlah uang yang dihitung kasir di laci.

Kode 2.13 `DailyOrders.java`: pesanan sehari, kurang Rp 6.000.

```java
// DailyOrders.java: menjumlahkan pesanan Profitina Laundry selama satu hari. Versi ini mengandung bug.
public class DailyOrders {

    static final double MIN_KG = 3.0;            // cucian yang lebih ringan ditagih 3 kg

    static long pricePerKg(String service) {
        return switch (service) {
            case "wash" -> 6_000;
            case "iron" -> 4_000;
            case "wash+iron" -> 9_000;
            default -> throw new IllegalArgumentException("unknown service: " + service);
        };
    }

    static long linePrice(String service, double kg) {
        long billedKg = (long) Math.max(kg, MIN_KG);
        return billedKg * pricePerKg(service);
    }

    public static void main(String[] args) {
        String[] services = {"wash", "iron", "wash+iron", "wash", "iron",
                             "wash+iron", "wash", "iron", "wash", "wash+iron"};
        double[] kgs = {3.0, 5.0, 2.0, 7.5, 4.0, 6.0, 3.5, 8.0, 1.5, 5.0};
        long day = 0;
        for (int i = 0; i < services.length; i++) {
            long price = linePrice(services[i], kgs[i]);
            System.out.printf("Order %2d  %-9s %4.1f kg  Rp %6d%n", i + 1, services[i],
kgs[i], price);
            day += price;
        }
        System.out.println("Day total : Rp " + day);
        System.out.println("Expected  : Rp 296000");
    }
}
```

```
Order  1  wash       3.0 kg  Rp  18000
Order  2  iron       5.0 kg  Rp  20000
Order  3  wash+iron  2.0 kg  Rp  27000
Order  4  wash       7.5 kg  Rp  42000
Order  5  iron       4.0 kg  Rp  16000
Order  6  wash+iron  6.0 kg  Rp  54000
Order  7  wash       3.5 kg  Rp  18000
Order  8  iron       8.0 kg  Rp  32000
Order  9  wash       1.5 kg  Rp  18000
Order 10  wash+iron  5.0 kg  Rp  45000
Day total : Rp 290000
Expected  : Rp 296000
```

Kode 2.14 `daily_orders.py`: pesanan hari yang sama dalam Python, dengan kekurangan yang sama.

```python
"""daily_orders.py: menjumlahkan pesanan Profitina Laundry selama satu hari. Versi ini
mengandung bug."""

MIN_KG = 3.0                    # cucian yang lebih ringan ditagih 3 kg


def price_per_kg(service):
    if service == "wash":
        return 6_000
    elif service == "iron":
        return 4_000
    elif service == "wash+iron":
        return 9_000
    raise ValueError("unknown service: " + service)


def line_price(service, kg):
    billed_kg = int(max(kg, MIN_KG))
    return billed_kg * price_per_kg(service)


def main():
    services = ["wash", "iron", "wash+iron", "wash", "iron",
                "wash+iron", "wash", "iron", "wash", "wash+iron"]
    kgs = [3.0, 5.0, 2.0, 7.5, 4.0, 6.0, 3.5, 8.0, 1.5, 5.0]
    day = 0
    for i in range(len(services)):
        price = line_price(services[i], kgs[i])
        print(f"Order {i + 1:2d}  {services[i]:<9} {kgs[i]:4.1f} kg  Rp {price:6d}")
        day += price
    print("Day total : Rp", day)
    print("Expected  : Rp 296000")


if __name__ == "__main__":
    main()
```

```
Order  1  wash       3.0 kg  Rp  18000
Order  2  iron       5.0 kg  Rp  20000
Order  3  wash+iron  2.0 kg  Rp  27000
Order  4  wash       7.5 kg  Rp  42000
Order  5  iron       4.0 kg  Rp  16000
Order  6  wash+iron  6.0 kg  Rp  54000
Order  7  wash       3.5 kg  Rp  18000
Order  8  iron       8.0 kg  Rp  32000
Order  9  wash       1.5 kg  Rp  18000
Order 10  wash+iron  5.0 kg  Rp  45000
Day total : Rp 290000
Expected  : Rp 296000
```

(`printf` di Java dan *f-string* di Python mencetak kolom: `%2d` adalah bilangan bulat selebar dua karakter, `%-9s` string rata kiri dalam sembilan karakter, `%4.1f` bilangan desimal dengan satu angka di belakang titik; Bab 3 menjelaskan pemformatan secara tuntas.)

Pesanan mana yang salah? Anda bisa saja memasang *breakpoint* di `linePrice` lalu memeriksa sepuluh perhentian, tetapi pikirkanlah sebaliknya seperti apa "salah" itu *terlihat* di dalam `linePrice`. Aturannya, berat yang ditagih adalah yang lebih besar di antara berat sebenarnya dan 3 kg, sehingga berat yang ditagih tidak boleh *kurang* dari berat sebenarnya. *Breakpoint* pada baris `return` di `linePrice` (baris 17 di Java, baris 18 di Python) dengan kondisi

```
billedKg < kg          (Java)
billed_kg < kg         (Python)
```

hanya akan berhenti untuk pesanan-pesanan yang melanggar aturan itu.

- **Eclipse:** pasang *breakpoint*-nya, lalu klik kanan dan pilih *Breakpoint Properties...* (atau pilih *breakpoint* itu di *view* Breakpoints); centang **Conditional**, pilih *Suspend when 'true'*, lalu ketik kondisinya di kotak di bawahnya (bagian bawah **Gambar 2.42**).
- **VS Code:** klik kanan pada *gutter* baris itu dan pilih *Add Conditional Breakpoint...*, pilih *Expression*, ketik kondisinya, lalu tekan Enter. Titiknya kini bertanda sama dengan kecil.
- **pdb:** tambahkan kondisinya sesudah sebuah koma: `break 18, billed_kg < kg`.

Berikut sesi `pdb`-nya:

```
> /home/student/laundry/daily_orders.py(1)<module>()
-> """daily_orders.py: adds up one day of Profitina Laundry orders. This version has a
bug."""
(Pdb) break 18, billed_kg < kg
Breakpoint 1 at /home/student/laundry/daily_orders.py:18
(Pdb) continue
Order  1  wash       3.0 kg  Rp  18000
Order  2  iron       5.0 kg  Rp  20000
Order  3  wash+iron  2.0 kg  Rp  27000
> /home/student/laundry/daily_orders.py(18)line_price()
-> return billed_kg * price_per_kg(service)
(Pdb) p service, kg, billed_kg
('wash', 7.5, 7)
(Pdb) continue
Order  4  wash       7.5 kg  Rp  42000
Order  5  iron       4.0 kg  Rp  16000
Order  6  wash+iron  6.0 kg  Rp  54000
> /home/student/laundry/daily_orders.py(18)line_price()
-> return billed_kg * price_per_kg(service)
(Pdb) p service, kg, billed_kg
('wash', 3.5, 3)
(Pdb) continue
Order  7  wash       3.5 kg  Rp  18000
Order  8  iron       8.0 kg  Rp  32000
Order  9  wash       1.5 kg  Rp  18000
Order 10  wash+iron  5.0 kg  Rp  45000
Day total : Rp 290000
Expected  : Rp 296000
The program finished and will be restarted
> /home/student/laundry/daily_orders.py(1)<module>()
-> """daily_orders.py: adds up one day of Profitina Laundry orders. This version has a
bug."""
(Pdb) quit
```

Program berjalan bebas melewati pesanan 1 sampai 3 (baris-barisnya tercetak) dan hanya berhenti dua kali: untuk cucian 7,5 kg yang ditagih sebagai 7 kg, dan cucian 3,5 kg yang ditagih sebagai 3 kg. Kekurangannya terjelaskan sampai ke rupiah terakhir: $0.5 \times 6{,}000 + 0.5 \times 6{,}000 = 6{,}000$. Penyebabnya adalah konversi ke bilangan bulat: `(long) Math.max(kg, MIN_KG)` di Java dan `int(max(kg, MIN_KG))` di Python sama-sama *memotong* (*truncate*), yaitu membuang pecahannya, sehingga setengah kilogram cucian diberikan secara cuma-cuma. Pesanan 9, 1,5 kg, *tidak* menghentikan program, karena batas minimum 3 kg sudah menaikkannya menjadi tepat 3.0 sebelum pemotongan. Perbaikannya adalah mengalikan lebih dahulu lalu mengonversi harganya, bukan beratnya, persis seperti yang dikerjakan `LaundryPrice`; Soal Latihan 2.6 meminta Anda membuat perbaikan itu dan memeriksa total yang baru.

Dua perkakas yang berkaitan: ***hit count*** hanya berhenti pada kedatangan ke-n (VS Code: *Hit Count*; `pdb`: `ignore 1 6` melewati enam kedatangan berikutnya pada *breakpoint* 1), dan ***exception breakpoint*** (*breakpoint* eksepsi) berhenti pada saat sebuah eksepsi dilempar, di mana pun (Eclipse: tombol *J!* pada *view* Breakpoints; VS Code: centang *Uncaught Exceptions* atau *Raised Exceptions*). Hal itu membawa kita ke bug-bug yang berisik.

2.13 Membaca *Stack Trace*

Ketika program Java atau Python menghadapi keadaan yang tidak dapat ditanganinya, ia *melempar* (Python: *membangkitkan*, *raise*) sebuah **eksepsi** (*exception*). Bila tidak ada yang menangkapnya, program berhenti dan mencetak ***stack trace*** (jejak tumpukan; di Python disebut *traceback*): pesan yang menyebut eksepsinya beserta tumpukan pemanggilan pada saat eksepsi itu dilempar. Pemula sering memandang *stack trace* sebagai tembok teks merah lalu menggulirnya begitu saja. Padahal itulah pesan paling berguna yang pernah diberikan *runtime* kepada Anda, karena ia menyatakan *apa* yang salah, *di mana*, dan *bagaimana program sampai ke sana*.

Profitina Laundry mulai menawarkan layanan *dry cleaning* (cuci kering), tetapi belum ada yang menambahkannya ke daftar harga. **Kode 2.15** dan **Kode 2.16** menghargai sebuah pesanan yang memuat layanan itu.

Kode 2.15 `UnknownService.java`: pesanan yang menyebut layanan yang tidak dikenal daftar harga.

```java
// UnknownService.java: pesanan yang menyebut layanan yang tidak dikenal daftar harga.
public class UnknownService {

    static final double MIN_KG = 3.0;

    static long pricePerKg(String service) {
        return switch (service) {
            case "wash" -> 6_000;
            case "iron" -> 4_000;
            case "wash+iron" -> 9_000;
            default -> throw new IllegalArgumentException("unknown service: " + service);
        };
    }

    static long linePrice(String service, double kg) {
        double billedKg = Math.max(kg, MIN_KG);
        return (long) (billedKg * pricePerKg(service));
    }

    public static void main(String[] args) {
        String[] services = {"wash", "dry-clean"};
        double[] kgs = {4.0, 2.0};
        for (int i = 0; i < services.length; i++) {
            System.out.println(services[i] + ": Rp " + linePrice(services[i], kgs[i]));
        }
    }
}
```

```
wash: Rp 24000
Exception in thread "main" java.lang.IllegalArgumentException: unknown service: dry-clean
    at UnknownService.pricePerKg(UnknownService.java:11)
    at UnknownService.linePrice(UnknownService.java:17)
    at UnknownService.main(UnknownService.java:24)
```

Kode 2.16 `unknown_service.py`: pesanan yang sama dalam Python.

```
"""unknown_service.py: pesanan yang menyebut layanan yang tidak dikenal daftar harga."""

MIN_KG = 3.0


def price_per_kg(service):
    if service == "wash":
        return 6_000
    elif service == "iron":
        return 4_000
    elif service == "wash+iron":
        return 9_000
    raise ValueError("unknown service: " + service)


def line_price(service, kg):
    billed_kg = max(kg, MIN_KG)
    return int(billed_kg * price_per_kg(service))


def main():
    services = ["wash", "dry-clean"]
    kgs = [4.0, 2.0]
    for i in range(len(services)):
        print(services[i] + ": Rp", line_price(services[i], kgs[i]))


if __name__ == "__main__":
    main()
```

```
wash: Rp 24000
Traceback (most recent call last):
  File "/home/student/laundry/unknown_service.py", line 29, in <module>
    main()
  File "/home/student/laundry/unknown_service.py", line 25, in main
    print(services[i] + ": Rp", line_price(services[i], kgs[i]))
                               ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/home/student/laundry/unknown_service.py", line 18, in line_price
    return int(billed_kg * price_per_kg(service))
                           ^^^^^^^^^^^^^^^^^^^^^
  File "/home/student/laundry/unknown_service.py", line 13, in price_per_kg
    raise ValueError("unknown service: " + service)
ValueError: unknown service: dry-clean
```

2.13.1 Membaca *stack trace* Java

Bacalah *stack trace* Java dari atas.

1. `Exception in thread "main"`: eksepsi itu tidak tertangkap di mana pun, di dalam *thread* yang menjalankan `main`.
2. `java.lang.IllegalArgumentException`: *tipe* eksepsinya, lengkap dengan paketnya. Namanya saja sering sudah bercerita: sebuah argumen tidak sah.
3. `unknown service: dry-clean`: *pesannya*, yaitu teks yang diberikan ketika eksepsi itu dibuat. Inilah sebabnya pesan eksepsi yang baik mencantumkan nilai yang menjadi biang masalah.
4. Baris-baris yang diawali `at` adalah tumpukan pemanggilan, **pemanggilan terbaru lebih dulu**. Yang teratas, `UnknownService.pricePerKg(UnknownService.java:11)`, adalah tempat eksepsi dilempar: *method* `pricePerKg`, berkas `UnknownService.java`, baris 11 (cabang `default`). Di bawahnya, `linePrice` di baris 17 memanggil `pricePerKg`, dan `main` di baris 24 memanggil `linePrice`.

Baris yang *melempar* tidak selalu merupakan baris yang *salah*: `pricePerKg` benar ketika menolak layanan yang tidak dikenal, dan pertanyaan sebenarnya adalah mengapa `main` memintanya. Dalam program sungguhan, lewati *frame-frame* pustaka (`java.base/...`) dan carilah *frame* yang menyebut kelas-kelas Anda sendiri. Di kedua IDE, setiap `(File.java:11)` di Console adalah tautan ke baris tersebut.

2.13.2 Membaca *traceback* Python

Python mencetak informasi yang sama dalam *urutan terbalik*, sebagaimana diperingatkan baris pertamanya: `Traceback (most recent call last)`. Jadi bacalah *traceback* Python **dari bawah**.

1. Baris terakhir, `ValueError: unknown service: dry-clean`, memberikan tipe dan pesan eksepsinya.
2. Tepat di atasnya, *frame* terdalam: `File ".../unknown_service.py", line 13, in price_per_kg`, diikuti baris sumbernya sendiri, `raise ValueError(...)`.
3. Di atasnya lagi, `line_price` di baris 18 memanggilnya, di atasnya `main` di baris 25, dan paling atas pemanggilan tingkat modul `main()` di baris 29.

Python 3.11 dan sesudahnya menambahkan baris-baris tanda sisipan (`^^^^`) di bawah bagian persis sebuah baris yang sedang dievaluasi, yang sangat membantu bila satu baris memuat beberapa pemanggilan: di sini tanda sisipan di bawah `line_price(services[i], kgs[i])` dan `price_per_kg(service)` menunjuk pemanggilan itu sendiri, bukan `print` atau `int` di sekelilingnya. (Python 3.13 dan 3.14 juga mencetak *traceback* berwarna di terminal.) Jalur yang ditampilkan untuk setiap berkas adalah jalur lengkap di mesin tempat program berjalan, sehingga di mesin Anda jalurnya akan berbeda.

Baris pertama, `wash: Rp 24000`, memperlihatkan bahwa program sempat mengerjakan sesuatu yang berguna sebelum gagal. *Stack trace* dikirim ke aliran *standard error*, sehingga terminal mencampurnya dengan keluaran biasa, tetapi mengalihkan keluaran ke sebuah berkas hanya akan menyimpan baris-baris yang normal.

> **WAWASAN KUNCI:** *Stack trace* adalah tumpukan pemanggilan dari Subbab 2.10.5 yang dicetak pada saat program mogok. Java mendaftarnya dari yang terbaru, Python dari yang tertua, tetapi keduanya memuat tiga fakta yang sama: tipe eksepsi, pesannya, dan rantai pemanggilan beserta nama berkas dan nomor baris. Bacalah ketiganya sebelum Anda mengubah satu baris pun.

Debugger dapat membawa Anda selangkah lebih jauh. Dengan *exception breakpoint* (Subbab 2.12) pada `IllegalArgumentException` di Eclipse atau VS Code, atau dengan *Raised Exceptions* tercentang untuk Python, program berhenti di baris `throw` atau `raise` *sebelum* tumpukannya terurai, dengan setiap variabel di setiap *frame* masih dapat diperiksa. `pdb` melakukan hal yang sama secara otomatis dalam mode *post-mortem*: `python3 -m pdb unknown_service.py`, lalu `continue`, berhenti di baris yang membangkitkan eksepsi sehingga Anda dapat mengetik `where` dan `p service`.

2.14 Profitina dalam Praktik

Profitina, platform *AI Visibility* di `app.profitina.com`, adalah aplikasi web Python yang dibangun dengan FastAPI dan dilayani oleh Uvicorn, dengan PostgreSQL di lingkungan produksi dan dependensi yang dikunci dalam sebuah *lock file*. Tiga kebiasaan dari bab ini tampak dalam cara produk semacam itu dijaga tetap sehat.

Ulangi dulu sebelum memperbaiki. Ketika seorang pemilik merek melaporkan bahwa sebuah skor visibilitas "tampak keliru", langkah pertama bukanlah membaca kode, melainkan mengulangi pemindaian pada salinan uji dengan masukan yang sama, seperti langkah 2 pada **Gambar 2.44**. Hanya bug yang dapat diulangi yang dapat dilangkahi dengan *breakpoint*, dan hanya bug yang sudah diperbaiki lalu dijadikan pengujian otomatis yang akan tetap terperbaiki. Profitina memperlakukan seluruh rangkaian pengujian otomatisnya, jauh lebih dari seribu pengujian, sebagai gerbang rilis: tidak ada yang dipasang ke produksi sampai semuanya lulus.

***Stack trace* adalah data.** Server web tidak dapat berhenti pada sebuah *breakpoint* di hadapan pengguna sungguhan, sehingga ketika sebuah permintaan gagal, server mencatat *traceback*-nya di log alih-alih menampilkannya kepada pengguna. Membaca *traceback* itu dari bawah ke atas, persis seperti pada Subbab 2.13.2, adalah cara menelusuri permintaan yang gagal sampai ke *frame* tempat keputusan yang keliru diambil. *Traceback* dicatat tanpa data pribadi apa pun, sejalan dengan Undang-Undang Pelindungan Data Pribadi (UU PDP 27/2022).

Ketahui mesin mana menjalankan apa. Fitur-fitur AI Profitina berjalan secara lokal di server milik Profitina sendiri, dengan Qwen2.5 yang dilayani melalui Ollama, sehingga tidak ada data pengguna yang meninggalkan server untuk pemrosesan AI; bila tidak ada model yang tersedia, aplikasi mengatakannya terus terang alih-alih berpura-pura. Men-debug sistem semacam itu dimulai dengan pertanyaan Subbab 2.7.3 dalam bentuk yang lebih besar: interpreter mana, model mana, dan konfigurasi mana yang sebenarnya sedang berjalan? *Endpoint* kesehatan (*health endpoint*) yang melaporkan versi yang terpasang, dan proses rilis yang baru dinyatakan berhasil setelah *endpoint* itu menjawab, adalah padanan di lingkungan produksi bagi kebiasaan melirik bilah status sebelum menekan F5.

Seperti biasa, aplikasi memberi saran dan pengguna yang memutuskan; perkakas *debugging* membantu pengembang membuat saran itu benar, tetapi tidak pernah menjadikannya jaminan.

2.15 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya

Menjalankan program yang lama. Anda menyunting berkas tetapi tidak menyimpannya (carilah `*` pada tabnya), atau di terminal Anda menyunting `LaundryPrice.java` lalu menjalankan `java LaundryPrice` tanpa menjalankan `javac` lagi. *Cara menghindari:* simpan sebelum menjalankan; di terminal, kompilasi dan jalankan bersama-sama, atau pakai `java LaundryPrice.java`.

Python yang salah. VS Code menjalankan interpreter yang berbeda dari tempat Anda memasang sebuah paket, sehingga *import* gagal. *Cara menghindari:* periksa interpreter di bilah status, dan bacalah jalur lengkap interpreter yang diketikkan VS Code ke Terminal.

Menjalankan modul paket sebagai skrip. `python3 main/tester.py` gagal dengan `ModuleNotFoundError: No module named 'util'`. *Cara menghindari:* pergilah ke folder yang memuat paket-paket itu dan jalankan `python3 -m main.tester`.

Paket dan folder tidak sesuai. Berkas Java yang menyatakan `package util;` tetapi berada di `src/tools/` tidak dapat dikompilasi di Eclipse dan tidak dapat ditemukan oleh `java`. *Cara menghindari:* buatlah kelas melalui *New* > *Class* di paket yang benar, dan pindahkan dengan *Refactor* > *Move*, yang memperbarui folder dan setiap *import*.

Menekan Run alih-alih Debug, atau menyebut nama kelas secara keliru. *Breakpoint* diabaikan di bawah *Run As*, dan `java main/Tester.class` gagal. *Cara menghindari:* pakailah *Debug As* atau F5 di VS Code, dan sebutlah nama kelas dengan titik: `java -cp bin main.Tester`.

Masuk ke segala hal. Melakukan *step into* pada setiap pemanggilan membawa Anda jauh ke dalam kode yang sebenarnya Anda percayai. *Cara menghindari: step over* pada yang Anda percayai, *step into* pada yang Anda curigai, *step out* begitu Anda tersesat.

Berhenti pada nilai salah pertama tanpa menjelaskannya. Memperbaiki baris mencurigakan yang pertama tanpa memeriksa apakah baris itu menjelaskan seluruh gejala sering kali memperbaiki *sebuah* bug tetapi bukan *sang* bug. *Cara menghindari:* kerjakan hitungannya, seperti pada Subbab 2.11.5, sampai perbaikan itu menjelaskan kesalahannya hingga rupiah terakhir.

Membaca *stack trace* dari ujung yang salah. *Cara menghindari:* Java mendaftar pemanggilan terbaru lebih dulu, Python paling akhir; temukan tipe dan pesan eksepsinya, lalu *frame* pertama yang merupakan kode Anda sendiri.

Meninggalkan cetakan *debug*. *Cara menghindari:* utamakan *watch* dan *logpoint*, dan carilah sisa-sisa pernyataan cetak sebelum Anda selesai.

2.16 Ringkasan Bab

- **IDE** memadukan penyuntingan, pembangunan, eksekusi, *debugging*, dan navigasi. **Tabel 2.2** menerjemahkan antara Eclipse dan VS Code.
- CPU hanya menjalankan **kode mesin**. Java mengompilasi kode sumber menjadi ***bytecode*** sekali saja dengan `javac`, dan **JVM** menginterpretasikan serta mengompilasi-JIT *bytecode* itu setiap kali program dijalankan. Python mengompilasi kode sumber menjadi *bytecode* di dalam interpreter setiap kali program dimulai, menyimpan modul-modul yang diimpor di `__pycache__`, dan mesin virtualnya menjalankan *bytecode* tersebut.
- Java menemukan nama yang tidak ada dan tipe yang salah pada **waktu kompilasi**; Python menemukan galat sintaksis sebelum program berjalan, tetapi nama yang tidak ada baru pada **waktu jalan**, ketika barisnya dieksekusi.
- Eclipse memiliki ***workspace*** berisi **proyek** dan jendela berisi ***view*** serta **editor** yang ditata dalam ***perspective***; VS Code bekerja pada sebuah **folder** dan menjalankan Python dengan interpreter yang dipilih di bilah status.
- **Paket** adalah folder berisi kode yang saling berkaitan; ***import*** membawa nama-nama dari paket lain. Kelas Java dijalankan sebagai `java -cp bin paket.Kelas`; modul Python di dalam paket dijalankan sebagai `python -m paket.modul` dari folder di atas paket-paket itu.
- ***Debugger*** menangguhkan program pada ***breakpoint*** dan memungkinkan Anda melakukan ***step over***, ***step into***, dan ***step out***, memeriksa dan mengubah **variabel**, menyimpan ***watch***, serta membaca **tumpukan pemanggilan**. *Breakpoint* **bersyarat** dan ***hit count*** hanya berhenti ketika perlu; ***exception breakpoint*** berhenti di tempat sebuah eksepsi dilempar.
- Men-debug-lah dengan metode: amati, ulangi, susun hipotesis, pasang *breakpoint*, bandingkan nilai aktual dengan nilai harapan yang dihitung secara manual, perbaiki, jalankan ulang (**Gambar 2.44**). Tagihan cucian itu terlalu mahal karena biaya tambahan ekspres mengalikan total berjalan, bukan harga butirnya (Persamaan (2.4)).
- ***Stack trace*** memberikan tipe eksepsi, pesannya, dan rantai pemanggilan; bacalah milik Java dari atas dan milik Python dari bawah.

2.17 Soal Latihan

Soal Latihan 2.1. Dalam satu kalimat untuk masing-masing, jelaskan apa itu *workspace*, *project*, *package*, *view*, *editor*, dan *perspective* di Eclipse, lalu sebutkan padanan terdekat masing-masing di VS Code.

Soal Latihan 2.2. Untuk setiap kesalahan berikut, nyatakan apakah Java melaporkannya pada waktu kompilasi atau waktu jalan, dan apakah Python melaporkannya sebelum program dimulai atau selama program berjalan: (a) `;` yang hilang di Java atau `:` yang hilang di Python; (b) nama variabel yang salah eja di dalam cabang `if` yang hanya dijalankan pada hari Minggu; (c) memanggil `count(300)` pada `Counter` di **Kode 2.5** dan **Kode 2.7**; (d) *import* `Counter` yang terlupa.

Soal Latihan 2.3. Berkas `src/com/profitina/laundry/Receipt.java` diawali dengan `package com.profitina.laundry;` dan memuat `main`. Tuliskan perintah-perintah untuk mengompilasinya ke folder `bin` dan menjalankannya, lalu nyatakan di mana `Receipt.class` berakhir.

Soal Latihan 2.4. Di folder `python` dari Subbab 2.8.3, seorang mahasiswa mengetik `python3 main/tester.py` dan memperoleh `ModuleNotFoundError: No module named 'util'`. Jelaskan sebabnya, dan berikan perintah yang benar.

Soal Latihan 2.5. Program **Kode 2.11** tertangguhkan pada perhentian pertamanya di baris 26. Dimulai dari sana, Anda menekan Step Into tiga kali, lalu Step Out dua kali. Sesudah setiap tekanan dari kelima tekanan itu, sebutkan *method* dan baris tempat program tertangguhkan serta banyaknya *frame* kode Anda sendiri pada tumpukan pemanggilan.

Soal Latihan 2.6. Perbaiki **Kode 2.13** dan **Kode 2.14** sehingga total harian menjadi Rp 296.000. *Breakpoint* bersyarat mana yang memperlihatkan bahwa perbaikan itu berhasil, dan apa yang dilakukannya sesudah perbaikan?

Soal Latihan 2.7. Sebuah *tester* memanggil `count(5)` dua kali pada `Counter` yang sama dari **Kode 2.5** dan memperoleh 15, lalu 30. Pakailah *debugger* (variabel mana yang akan Anda pantau dengan *watch*, dan di mana Anda akan memasang *breakpoint*?) untuk menjelaskan sebabnya, lalu tulislah versi yang pemanggilan keduanya juga mengembalikan 15.

Soal Latihan 2.8. Profitina Laundry menambahkan layanan *dry cleaning* seharga Rp 15.000 per kg, dengan batas minimum 3 kg yang sama. Tulislah program dalam kedua bahasa yang mencetak struk terperinci untuk pesanan seorang anggota berupa 4 kg cuci, 2 kg *dry-clean* ekspres, dan 3,5 kg setrika, diikuti subtotal, potongan anggota, dan totalnya. Hitunglah totalnya secara manual lebih dahulu.

Soal Latihan 2.9. Dengan memakai Persamaan (2.4), hitunglah berapa yang akan ditagihkan **Kode 2.9** yang *mengandung bug* kepada Bu Sari bila ketiga butirnya dimasukkan dengan urutan wash+iron (ekspres), wash, iron. Jelaskan apa yang disampaikan hasil ini tentang pengujian.

Soal Latihan 2.10. Program **Kode 2.9** yang mengandung bug tertangguhkan pada perhentian ketiganya di baris 27, lalu Anda memakai *view* Variables untuk mengubah `express[2]` dari `true` menjadi `false`, kemudian melanjutkan eksekusi. Total berapa yang dicetak program, dan mengapa percobaan ini berguna?

Soal Latihan 2.11. Jelaskan perbedaan antara *breakpoint*, *breakpoint* bersyarat, *breakpoint hit count*, *exception breakpoint*, *watchpoint*, dan *logpoint* VS Code, lalu berikan satu situasi di binatu yang untuknya masing-masing merupakan pilihan terbaik.

2.18 Penyelesaian Soal Latihan

Penyelesaian 2.1. *Workspace* adalah folder yang menyimpan metadata Eclipse dan, secara bawaan, proyek-proyek Anda (VS Code: folder yang dibuka atau berkas `.code-workspace`). *Project* adalah satuan yang dapat dibangun dan memiliki berkas `.project` (VS Code: proyek yang ditampilkan di *view* Java Projects). *Package* adalah sekelompok kelas bernama yang disimpan sebagai folder (sama di keduanya). *View* menampilkan informasi terstruktur seperti Package Explorer (VS Code: *view* di bilah samping dan panel). *Editor* memuat satu berkas dan harus disimpan (VS Code: tab editor). *Perspective* menata *view* untuk suatu tugas (VS Code: satu tata letak yang menambahkan *view* Run and Debug ketika men-debug).

Penyelesaian 2.2. (a) Java: waktu kompilasi; Python: sebelum program dimulai (`SyntaxError`), karena Python mengompilasi seluruh berkas lebih dahulu. (b) Java: waktu kompilasi (`cannot find symbol`), walaupun cabangnya jarang dijalankan; Python: pada waktu jalan, dan hanya pada hari Minggu, ketika cabang itu dijalankan (`NameError`). (c) Keduanya pada waktu jalan: nilai 300 baru diketahui ketika pemanggilan terjadi, dan *method*-nya melempar eksepsi. (d) Java: waktu kompilasi; Python: pada waktu jalan, ketika nama itu pertama kali dipakai (Subbab 2.8.4).

Penyelesaian 2.3. Dari folder yang memuat `src`: `javac -d bin src/com/profitina/laundry/Receipt.java`, lalu `java -cp bin com.profitina.laundry.Receipt`. Berkas kelasnya adalah `bin/com/profitina/laundry/Receipt.class`, karena `-d` membuat satu folder untuk setiap tingkat paket.

Penyelesaian 2.4. Bila dimulai sebagai jalur skrip, Python mencari *import* di folder skrip itu sendiri, `python/main`, yang tidak memuat `util`. Jalankanlah sebagai modul dari folder yang memuat kedua paket: `cd python`, lalu `python3 -m main.tester`.

Penyelesaian 2.5. Mulai di `orderTotal` pada baris 26 (dua *frame*: `orderTotal`, `main`). (1) Step Into: `linePrice`, baris 19, tiga *frame*. (2) Step Into: `linePrice`, baris 20, tiga *frame*; `Math.max` dilewati oleh *step filter*. (3) Step Into: `pricePerKg`, baris 10, empat *frame*. (4) Step Out: kembali di `linePrice`, baris 20, tiga *frame*, masih menuntaskan perkaliannya. (5) Step Out: kembali di `orderTotal`, baris 26, dua *frame*; satu Step Over kemudian menuntaskan penugasan `line = 24000`. Inilah jalur tur `jdb` pada Subbab 2.11.4, yang nomor-nomor barisnya sama karena program yang diperbaiki hanya mengubah baris 27 sampai 30.

Penyelesaian 2.6. Konversikan *harganya*, bukan beratnya. Di Java, ganti kedua baris `linePrice` dengan `double billedKg = Math.max(kg, MIN_KG);` dan `return (long) (billedKg * pricePerKg(service));`; di Python, dengan `billed_kg = max(kg, MIN_KG)` dan `return int(billed_kg * price_per_kg(service))`. Pesanan 4 dan 7 menjadi Rp 45.000 dan Rp 21.000, dan total harian menjadi $290{,}000 + 3{,}000 + 3{,}000 = 296{,}000$. *Breakpoint* bersyarat yang sama, `billedKg < kg` pada baris `return`, kini tidak pernah berhenti: program berjalan langsung sampai selesai, yang memastikan bahwa tidak ada pesanan yang ditagih di bawah berat sebenarnya.

Penyelesaian 2.7. Pasang *breakpoint* pada baris `result += i;` (baris 15 pada **Kode 2.5**) lalu pantau `result` dan `i`. Pada pemanggilan kedua, perhentian pertama sudah menampilkan `result = 15`: `result` adalah sebuah *field*, milik objek dan bertahan di antara pemanggilan, bukan variabel lokal yang dimulai lagi dari 0. Perbaikannya adalah menjumlahkan di dalam variabel lokal, seperti yang dilakukan `countFresh` pada **Kode 2.17** dan **Kode 2.18**, yang memperlihatkan kedua perilaku itu berdampingan.

Kode 2.17 `CounterTwice.java`: sebuah field mempertahankan nilainya di antara pemanggilan; variabel lokal dimulai dari awal.

```java
// CounterTwice.java: mengapa panggilan kedua count(5) menghasilkan 30, dan perbaikannya.
public class CounterTwice {

    private int result = 0;              // sebuah field: nilainya bertahan antarpanggilan

    int countKeepingTotal(int x) {       // perilaku semula
        for (int i = 1; i <= x; i++) {
            result += i;
        }
        return result;
    }

    int countFresh(int x) {              // perbaikannya: variabel lokal, baru setiap
panggilan
        int sum = 0;
        for (int i = 1; i <= x; i++) {
            sum += i;
        }
        return sum;
    }

    public static void main(String[] args) {
        CounterTwice c = new CounterTwice();
        System.out.println("original, 1st call: " + c.countKeepingTotal(5));
        System.out.println("original, 2nd call: " + c.countKeepingTotal(5));
        System.out.println("fixed,    1st call: " + c.countFresh(5));
        System.out.println("fixed,    2nd call: " + c.countFresh(5));
    }
}
```

```
original, 1st call: 15
original, 2nd call: 30
fixed,    1st call: 15
fixed,    2nd call: 15
```

Kode 2.18 `counter_twice.py`: perbandingan yang sama dalam Python, dengan sebuah atribut dan sebuah variabel lokal.

```
"""counter_twice.py: mengapa panggilan kedua count(5) menghasilkan 30, dan perbaikannya."""


class CounterTwice:

    def __init__(self):
        self.result = 0                  # sebuah atribut: nilainya bertahan antarpanggilan

    def count_keeping_total(self, x):    # perilaku semula
        for i in range(1, x + 1):
            self.result += i
        return self.result

    def count_fresh(self, x):            # perbaikannya: variabel lokal, baru setiap
panggilan
        total = 0
        for i in range(1, x + 1):
            total += i
        return total


def main():
    c = CounterTwice()
    print("original, 1st call:", c.count_keeping_total(5))
    print("original, 2nd call:", c.count_keeping_total(5))
    print("fixed,    1st call:", c.count_fresh(5))
    print("fixed,    2nd call:", c.count_fresh(5))


if __name__ == "__main__":
    main()
```

```
original, 1st call: 15
original, 2nd call: 30
fixed,    1st call: 15
fixed,    2nd call: 15
```

Penyelesaian 2.8. Secara manual, dengan Persamaan (2.2): cuci $6{,}000 \times 4 = 24{,}000$; *dry-clean* $15{,}000 \times \max(2{,}3) = 45{,}000$, ditambah 50% ekspres, menjadi 67,500; setrika $4{,}000 \times 3.5 = 14{,}000$. Subtotalnya $105{,}500 \geq 100{,}000$, sehingga potongan anggotanya 10,550 dan, menurut Persamaan (2.3), totalnya 94,950. **Kode 2.19** dan **Kode 2.20** menghitung biaya tambahan di dalam `linePrice`, tempat yang semestinya, sehingga bug dari Subbab 2.11 tidak dapat kembali.

Kode 2.19 `LaundryReceipt.java`: struk terperinci dengan tambahan layanan dry cleaning.

```java
// LaundryReceipt.java: struk terperinci, dengan tambahan layanan dry-clean.
public class LaundryReceipt {

    static final double MIN_KG = 3.0;            // cucian yang lebih ringan ditagih 3 kg
    static final long EXPRESS_PERCENT = 50;      // layanan ekspres 50% lebih mahal
    static final long MEMBER_DISCOUNT_PERCENT = 10;
    static final long DISCOUNT_THRESHOLD = 100_000;

    static long pricePerKg(String service) {
        return switch (service) {
            case "wash" -> 6_000;
            case "iron" -> 4_000;
            case "wash+iron" -> 9_000;
            case "dry-clean" -> 15_000;
            default -> throw new IllegalArgumentException("unknown service: " + service);
        };
    }

    static long linePrice(String service, double kg, boolean express) {
        double billedKg = Math.max(kg, MIN_KG);
        long line = (long) (billedKg * pricePerKg(service));
        if (express) {
            line += line * EXPRESS_PERCENT / 100;
        }
        return line;
    }

    public static void main(String[] args) {
        String[] services = {"wash", "dry-clean", "iron"};
        double[] kgs = {4.0, 2.0, 3.5};
        boolean[] express = {false, true, false};
        boolean member = true;
        long subtotal = 0;
        for (int i = 0; i < services.length; i++) {
            long line = linePrice(services[i], kgs[i], express[i]);
            System.out.printf("%-9s %4.1f kg %-7s Rp %7d%n",
                    services[i], kgs[i], express[i] ? "express" : "", line);
            subtotal += line;
        }
        long discount = 0;
        if (member && subtotal >= DISCOUNT_THRESHOLD) {
            discount = subtotal * MEMBER_DISCOUNT_PERCENT / 100;
        }
        System.out.printf("Subtotal                 Rp %7d%n", subtotal);
        System.out.printf("Member discount          Rp %7d%n", -discount);
        System.out.printf("Total                    Rp %7d%n", subtotal - discount);
```

Kode 2.19 (Lanjutan) *LaundryReceipt.java: struk terperinci dengan tambahan layanan dry cleaning.*

```java
    }
}
```

```
wash       4.0 kg         Rp   24000
dry-clean  2.0 kg express Rp   67500
iron       3.5 kg         Rp   14000
Subtotal                  Rp  105500
Member discount           Rp  -10550
Total                     Rp   94950
```

Kode 2.20 `laundry_receipt.py`: struk yang sama dalam Python; pemformatan `%` memakai kode-kode yang sama dengan `printf` di Java.

```
"""laundry_receipt.py: struk terperinci, dengan tambahan layanan dry-clean."""

MIN_KG = 3.0                    # cucian yang lebih ringan ditagih 3 kg
EXPRESS_PERCENT = 50            # layanan ekspres 50% lebih mahal
MEMBER_DISCOUNT_PERCENT = 10
DISCOUNT_THRESHOLD = 100_000


def price_per_kg(service):
    if service == "wash":
        return 6_000
    elif service == "iron":
        return 4_000
    elif service == "wash+iron":
        return 9_000
    elif service == "dry-clean":
        return 15_000
    raise ValueError("unknown service: " + service)


def line_price(service, kg, express):
    billed_kg = max(kg, MIN_KG)
    line = int(billed_kg * price_per_kg(service))
    if express:
        line += line * EXPRESS_PERCENT // 100
    return line


def main():
    services = ["wash", "dry-clean", "iron"]
    kgs = [4.0, 2.0, 3.5]
    express = [False, True, False]
    member = True
    subtotal = 0
    for i in range(len(services)):
        line = line_price(services[i], kgs[i], express[i])
        print("%-9s %4.1f kg %-7s Rp %7d"
              % (services[i], kgs[i], "express" if express[i] else "", line))
        subtotal += line
    discount = 0
    if member and subtotal >= DISCOUNT_THRESHOLD:
        discount = subtotal * MEMBER_DISCOUNT_PERCENT // 100
    print("Subtotal                  Rp %7d" % subtotal)
    print("Member discount           Rp %7d" % -discount)
    print("Total                     Rp %7d" % (subtotal - discount))
```

Kode 2.20 (Lanjutan) *laundry_receipt.py: struk yang sama dalam Python; pemformatan % memakai kode-kode yang sama dengan printf di Java.*

```
if __name__ == "__main__":
    main()
```

```
wash       4.0 kg         Rp   24000
dry-clean  2.0 kg express Rp   67500
iron       3.5 kg         Rp   14000
Subtotal                  Rp  105500
Member discount           Rp  -10550
Total                     Rp   94950
```

Penyelesaian 2.9. Dengan $t_0 = 0$: $t_1 = (0 + 45{,}000) \times 1.5 = 67{,}500$; $t_2 = 67{,}500 + 24{,}000 = 91{,}500$; $t_3 = 91{,}500 + 12{,}000 = 103{,}500$. Sesudah potongan, tagihannya 93,150, jumlah yang *benar*. Bug itu tidak terlihat setiap kali satu-satunya butir ekspres berada paling depan, sehingga

sebuah pengujian harus memuat butir ekspres sesudah butir-butir lain. Rangkaian pengujian harus dirancang untuk membuat bug terlihat, bukan sekadar untuk menjalankan kode.

Penyelesaian 2.10. Dengan penanda ekspres dimatikan, cabangnya dilewati, sehingga `total` tetap 81.000; karena 81,000 < 100,000, tidak ada potongan yang berlaku dan program mencetak `Total     : Rp 81000`. Hasil itu sesuai dengan Persamaan (2.2) dan Persamaan (2.3) untuk tiga butir yang tidak ekspres (24,000 + 12,000 + 45,000), yang memperlihatkan bahwa semua bagian selain cabang ekspres sudah benar. Mengubah nilai di dalam *debugger* menjawab pertanyaan "bagaimana jika" semacam itu tanpa menyunting atau mengompilasi ulang program.

Penyelesaian 2.11. *Breakpoint* berhenti setiap kali sebuah baris dicapai (baris yang menambahkan sebuah butir ke tagihan). *Breakpoint bersyarat* berhenti bila sebuah ungkapan bernilai benar (`billedKg < kg` pada Subbab 2.12). *Breakpoint hit count* berhenti pada kedatangan ke-n (pesanan ke-200 dalam sebulan). *Exception breakpoint* berhenti di tempat sebuah eksepsi dilempar (layanan tak dikenal `dry-clean`, dengan semua variabel masih utuh). *Watchpoint* berhenti ketika sebuah *field* dibaca atau ditulis (*method* mana yang mengubah saldo seorang pelanggan?). *Logpoint* mencetak tanpa berhenti (setiap total berjalan dari pesanan sehari).

Daftar Pustaka

[1] Eclipse Foundation, "Eclipse IDE documentation: Java development user guide," 2025. [Online]. Available: https://help.eclipse.org/

[2] L. Vogel, "Eclipse IDE for Java development: tutorial," vogella GmbH, 2024. [Online]. Available: https://www.vogella.com/tutorials/Eclipse/article.html

[3] Microsoft, "Java in Visual Studio Code" and "Running and debugging Java," Visual Studio Code documentation, 2025. [Online]. Available: https://code.visualstudio.com/docs/java/java-tutorial

[4] Microsoft, "Python in Visual Studio Code" and "Python debugging in VS Code," Visual Studio Code documentation, 2025. [Online]. Available: https://code.visualstudio.com/docs/python/python-tutorial

[5] T. Lindholm, F. Yellin, G. Bracha, A. Buckley, and D. Smith, *The Java Virtual Machine Specification, Java SE 21 Edition*. Redwood City, CA, USA: Oracle, 2023.

[6] Oracle, *Java Development Kit Version 21 Tool Specifications* (javac, java, jar, javap, jdb), 2023. [Online]. Available: https://docs.oracle.com/en/java/javase/21/docs/specs/man/

[7] Python Software Foundation, "dis — Disassembler for Python bytecode," *The Python Standard Library*, version 3.14, 2025. [Online]. Available: https://docs.python.org/3/library/dis.html

[8] Python Software Foundation, "Modules," *The Python Tutorial*, sec. 6, version 3.14, 2025. [Online]. Available: https://docs.python.org/3/tutorial/modules.html

[9] J. Gosling, B. Joy, G. Steele, G. Bracha, A. Buckley, D. Smith, and G. Bierman, *The Java Language Specification, Java SE 21 Edition*, ch. 7, "Packages and Modules." Redwood City, CA, USA: Oracle, 2023.

[10] M. Fowler, *Refactoring: Improving the Design of Existing Code*, 2nd ed. Boston, MA, USA: Addison-Wesley, 2018.

[11] Python Software Foundation, "pdb — The Python Debugger," *The Python Standard Library*, version 3.14, 2025. [Online]. Available: https://docs.python.org/3/library/pdb.html

[12] A. Zeller, *Why Programs Fail: A Guide to Systematic Debugging*, 2nd ed. Burlington, MA, USA: Morgan Kaufmann, 2009.

[13] D. J. Agans, *Debugging: The 9 Indispensable Rules for Finding Even the Most Elusive Software and Hardware Problems*. New York, NY, USA: AMACOM, 2002.

Bab 3

Kelas, Objek, String, Tipe, Percabangan, dan Perulangan

Setelah menyelesaikan bab ini, Anda akan mampu membaca kelas Java atau Python yang kecil dan menyebut setiap bagiannya: *field*, konstruktor, metode, serta kata `this` (atau `self`) yang merangkai semuanya. Anda akan membuat objek dan menggambarkannya di memori, mengolah teks dengan percaya diri, memilih tipe bilangan yang tepat dan mengenali tiga jebakan aritmetika, membuat keputusan dengan `if` dan `switch`, mengulang pekerjaan dengan perulangan, serta memeriksa sebuah perulangan dengan tangan memakai tabel penelusuran. Terakhir, Anda akan membangun dua kelas pertama Profitina Laundry, `Customer` dan `Service`, yang menghitung harga sekantong cucian, dan menjalankan setiap program dalam Java maupun Python.

3.1 Titik Berangkat Kita

Bab 1 menyiapkan meja kerja Anda, yaitu Java Development Kit, penerjemah (*interpreter*) Python, dan Visual Studio Code, dan Anda menjalankan program pertama dalam masing-masing bahasa. Bab 2 membuat meja kerja itu nyaman: proyek dan paket, Eclipse dan VS Code berdampingan, serta *debugger* dengan *breakpoint*, eksekusi langkah demi langkah, dan pengamatan variabel. Namun, program-program kita sejauh ini masih sangat kecil. Bab ini membongkar sebuah program dan menjajarkan setiap bagiannya di atas meja kerja.

Bagian-bagian itu terbagi menjadi dua kelompok. Kelompok pertama menyangkut *struktur*: **kelas** dan **objek**. Kelas menggambarkan suatu jenis benda: anjing, pelanggan, layanan laundry. Objek adalah satu benda tertentu dari jenis itu: anjing bernama Fred, pelanggan bernama Siti. Pemrograman berorientasi objek adalah keterampilan memilih kelas yang baik dan membiarkan objek-objek bekerja sama.

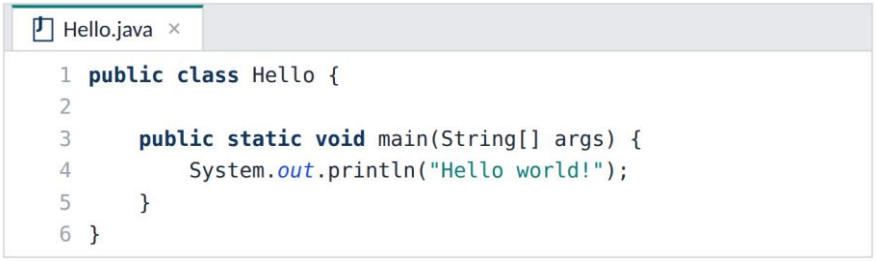
Kelompok kedua menyangkut *apa yang terjadi di dalam sebuah metode*: **nilai** yang diolah program, **keputusan** yang diambilnya, dan **pengulangan** yang dilakukannya. Pemrogram yang belum menguasainya akan menulis kelas yang indah tetapi menghitung jawaban yang salah. Di akhir bab, Profitina Laundry, usaha kecil yang perangkat lunaknya kita bangun bab demi bab, mendapat dua kelas pertamanya.

> **WAWASAN KUNCI:** Kelas adalah *deskripsi*; objek adalah *benda*. Segala hal lain dalam bab ini (string, tipe, percabangan, perulangan) adalah isi metode-metode yang memberi objek perilakunya.

3.2 Menengok ke Dalam Sebuah Kelas

3.2.1 Hello, sekali lagi

Hal paling sederhana yang dapat dilakukan program adalah mencetak sebaris teks. Anda sudah bertemu program ini di Bab 1; sekarang mari kita membacanya perlahan, kata demi kata. **Gambar 3.1** memperlihatkannya terbuka di editor Eclipse, dan **Kode 3.1** memberikan teks yang sama sebagai kode yang dapat Anda salin.



Gambar 3.1 Kelas `Hello` terbuka di editor Eclipse: satu kelas, satu metode `main`, satu pernyataan.

Kode 3.1 `Hello.java`: program Java lengkap yang paling kecil.

```java
public class Hello {

    public static void main(String[] args) {
        System.out.println("Hello world!");
    }
}
```

```
Hello world!
```

Setiap kata dalam **Kode 3.1** punya alasan untuk berada di sana.

- `public class Hello` mendeklarasikan sebuah **kelas** bernama `Hello`. Di Java, *semua* kode tinggal di dalam suatu kelas. Kata `public` berarti kode lain mana pun boleh memakai kelas ini. Berkasnya harus bernama `Hello.java`, persis sama dengan nama kelas, termasuk huruf kapital `H`.
- Kurung kurawal `{` dan `}` menandai awal dan akhir kelas. Segala sesuatu di antaranya milik `Hello`.
- `public static void main(String[] args)` mendeklarasikan sebuah **metode** bernama `main`, yaitu blok instruksi yang diberi nama. Peluncur Java memulai program dari metode yang bentuknya persis seperti ini ketika Anda mengetik `java Hello`. Kata `static` dijelaskan dengan tuntas pada bab selanjutnya; untuk sementara, bacalah sebagai "metode ini milik kelas itu sendiri, sehingga Java dapat memanggilnya sebelum ada objek apa pun". `void` berarti metode tidak mengembalikan nilai, dan `String[] args` menampung kata-kata yang diketik sesudah nama program, yang kita pakai sebentar lagi.
- `System.out.println("Hello world!");` adalah sebuah **pernyataan** (*statement*): satu instruksi. Pernyataan ini meminta objek `System.out` (keluaran standar, biasanya jendela terminal Anda) mencetak satu baris.

Perhatikan bagian yang rewel. Huruf kapital `S` pada `System`, titik-titiknya, tanda petik ganda, dan titik koma penutup semuanya wajib; salah satu saja keliru, kompilator menolak program itu. Ketelitian semacam inilah yang memungkinkan Java menangkap kesalahan sebelum program Anda berjalan.

Berikut program yang sama dalam Python, pada **Kode 3.2**.

Kode 3.2 `hello.py`: program yang sama dalam Python, dengan fungsi `main` yang lazim.

```python
def main():
    print("Hello world!")


if __name__ == "__main__":
    main()
```

```
Hello world!
```

Python tidak memerlukan kelas untuk program sekecil ini: satu baris `print("Hello world!")` saja sudah berfungsi. Kita membungkusnya dalam fungsi `main` dan memanggilnya di bawah `if __name__ == "__main__":`, yang bernilai benar hanya ketika berkas itu dijalankan langsung, bukan ketika diimpor oleh berkas lain. Nanti dalam bab ini kita mengimpor kelas dari satu berkas ke berkas lain, dan saat itulah kebiasaan ini terbayar.

Dua perbedaan antarbahasa sudah tampak. Java menandai blok dengan kurung kurawal; Python menandainya dengan **indentasi**: badan `main` adalah baris-baris yang menjorok di bawah `def main():`. Selain itu, Java mengakhiri pernyataan dengan titik koma, sedangkan Python mengakhirinya di ujung baris.

3.2.2 Membaca kata dari baris perintah

Program yang hanya bisa menulis baru separuh program. Cara paling sederhana memberi program masukan adalah mengetik kata-kata sesudah namanya ketika menjalankannya. Java mengumpulkan kata-kata itu ke dalam `args`. **Gambar 3.2** memperlihatkan versi kedua program ini, `Hello2`, yang memakainya.



Gambar 3.2 `Hello2` di Eclipse: program membaca kata pertama dari baris perintah melalui `args[0]` bila kata itu diberikan.

Kode 3.3 `Hello2.java`: mencetak kata pertama yang diketik di baris perintah.

```
public class Hello2 {

    public static void main(String[] args) {
        String aLineOfText = "";
        if (args.length > 0) {
            aLineOfText = args[0];
            System.out.println("She said, " + aLineOfText + "!");
        } else {
            System.out.println("Hello world!");
        }
    }
}
```

Jika Anda menjalankannya dengan kata-kata `Welcome to Profitina Laundry` sesudah nama kelas,

```
java Hello2 Welcome to Profitina Laundry
```

maka `args` berisi empat kata: `args[0]` adalah `Welcome`, `args[1]` adalah `to`, `args[2]` adalah `Profitina`, dan `args[3]` adalah `Laundry`. Program hanya mencetak yang pertama:

```
She said, Welcome!
```

Jalankan tanpa kata sama sekali, maka `args.length` bernilai `0`, cabang `else` dijalankan, dan program kembali ke salam bawaannya:

```
Hello world!
```

Tiga hal dalam **Kode 3.3** patut dicermati.

Pertama, **penomoran dimulai dari 0**. Kata pertama adalah `args[0]`, bukan `args[1]`. Hampir semua hal dalam komputasi dihitung dari nol: posisi dalam daftar, karakter dalam string, baris dalam tabel. Rasanya aneh selama seminggu, lalu menjadi wajar.

Kedua, `String aLineOfText = "";` mendeklarasikan sebuah **variabel**, yaitu kotak bernama yang menyimpan sebuah nilai. Tipenya `String`, tipe Java untuk teks. `String` adalah *tipe objek*: nilai `String` adalah objek yang memiliki metode sendiri, seperti yang akan kita lihat di Subbab 3.5. Untuk saat ini cukup diketahui bahwa kita menyimpan teks `args[0]` dalam variabel `aLineOfText` lalu mencetaknya.

Ketiga, tanda `+` merekatkan teks, dan operasi ini disebut **konkatenasi** (penyambungan): `"She said, " + aLineOfText + "!"` membangun satu string dari tiga string.

Pengujian `if (args.length > 0)` melindungi kita. Tanpanya, menjalankan program tanpa kata akan meminta `args[0]` yang tidak ada, dan Java akan berhenti dengan `ArrayIndexOutOfBoundsException`, jenis galat yang dibahas tuntas di Bab 5.

Versi Python, **Kode 3.4**, membaca kata-kata yang sama dari `sys.argv`. Satu perbedaan yang perlu diingat: `sys.argv[0]` adalah nama skrip itu sendiri, sehingga kata pertama yang sesungguhnya adalah `sys.argv[1]`.

Kode 3.4 `hello2.py`: kata-kata baris perintah tersimpan dalam `sys.argv`, sesudah nama skrip.

```python
import sys


def main():
    a_line_of_text = ""
    if len(sys.argv) > 1:
        a_line_of_text = sys.argv[1]
        print("She said, " + a_line_of_text + "!")
    else:
        print("Hello world!")


if __name__ == "__main__":
    main()
```

```
She said, Welcome!
```

3.2.3 Nama dan konvensi

Program jauh lebih sering dibaca daripada ditulis, sehingga setiap komunitas bahasa menyepakati rupa nama-nama di dalamnya. Kompilator tidak memaksakan konvensi ini, tetapi setiap pembaca berpengalaman mengharapkannya.

Java. Nama kelas diawali huruf kapital: `Hello`, `Dog`, `LaundryDemo`. Kelas publik `MyClass` harus berada dalam berkas bernama `MyClass.java`. Variabel dan metode diawali huruf kecil: `aLineOfText`, `getAge`, `myIntValue`. Bila sebuah nama tersusun dari beberapa kata, setiap kata

baru diawali huruf kapital tanpa spasi atau garis bawah: `myIntValue`, `kilogramsPerPound`. Gaya ini disebut **camel case**, karena huruf-huruf kapitalnya menonjol seperti punuk unta. Konstanta yang tidak pernah berubah ditulis dengan huruf kapital dan garis bawah: `MAX_VALUE`.

Python. Nama kelas juga memakai huruf kapital, `Dog`, `LaundryService` (gaya ini kadang disebut *CapWords*). Variabel dan fungsi ditulis dengan huruf kecil dan garis bawah di antara kata: `a_line_of_text`, `get_age`, `kilograms_per_pound`. Gaya ini disebut **snake case**. Panduan gaya resmi untuk kode Python, PEP 8 [1], menguraikan aturan-aturan ini dan banyak lagi.

Beberapa hal bukan konvensi, melainkan aturan. Nama-nama milik Java sendiri harus dieja dengan huruf kapital yang benar: `System` dan `String` diawali huruf kapital, `int` dan `char` tidak. Jika Anda menulis `string s = "...";`, kompilator melaporkan bahwa ia tidak menemukan tipe bernama `string`. Python sama ketatnya: `print` adalah fungsi, sedangkan `Print` adalah nama yang tidak dikenal.

3.2.4 Komentar yang menolong pembaca

Komentar adalah teks dalam program yang diabaikan komputer tetapi tidak diabaikan pembaca manusia. Komentar menjelaskan *mengapa* kode melakukan apa yang dilakukannya, sesuatu yang tidak dapat dikatakan oleh kode itu sendiri. Kebiasaan ini terbayar pertama kali Anda membuka program yang Anda tulis tiga bulan lalu.

Java memiliki tiga jenis komentar dan Python dua, semuanya diperlihatkan dalam **Kode 3.5** dan **Kode 3.6**.

Kode 3.5 `CommentDemo.java`: komentar Javadoc, komentar baris, dan komentar blok.

```java
/**
 * Mencetak satu baris nota singkat untuk satu pesanan laundry.
 * Komentar Javadoc seperti ini menjelaskan sebuah kelas atau metode
 * bagi orang yang akan memakainya.
 */
public class CommentDemo {

    public static void main(String[] args) {
        String customer = "Budi Santoso";
        String shorterName = customer.substring(0, 4); // empat karakter pertama, untuk
label

        /* Label yang dicetak pada kantong laundry harus muat dalam kotak kecil,
         * jadi kita memakai bentuk pendek nama diikuti beratnya.
         */
        double weightKg = 3.5;
        System.out.println("Bag tag: " + shorterName + " / " + weightKg + " kg");
    }
}
```

```
Bag tag: Budi / 3.5 kg
```

Kode 3.6 `comment_demo.py`: docstring dan komentar baris dalam Python.

```
"""Mencetak satu baris nota singkat untuk satu pesanan laundry.

Docstring seperti ini menjelaskan sebuah modul, kelas, atau fungsi
bagi orang yang akan memakainya.
"""


def main():
    customer = "Budi Santoso"
    shorter_name = customer[0:4]  # empat karakter pertama, untuk label

    # Label yang dicetak pada kantong laundry harus muat dalam kotak kecil,
    # jadi kita memakai bentuk pendek nama diikuti beratnya.
    weight_kg = 3.5
    print("Bag tag: " + shorter_name + " / " + str(weight_kg) + " kg")


if __name__ == "__main__":
    main()
```

```
Bag tag: Budi / 3.5 kg
```

- **Komentar baris** diawali `//` di Java dan `#` di Python, dan berlaku sampai akhir baris. Komentar ini cocok untuk catatan singkat tentang satu baris, misalnya "empat karakter pertama, untuk label".
- **Komentar blok** di Java diawali `/*` dan diakhiri `*/`, dan boleh terbentang beberapa baris. Biasanya komentar ini diletakkan *sebelum* kode yang dijelaskannya. Python tidak memiliki komentar blok; Anda cukup menulis beberapa baris `#`.
- **Komentar dokumentasi** menjelaskan sebuah kelas atau metode bagi orang yang akan *memakainya*, bukan membaca isinya. Di Java komentar ini diawali `/**` dan disebut komentar *Javadoc*; alat `javadoc` mengubahnya menjadi halaman web. Di Python, komentar dokumentasi adalah string yang diletakkan sebagai pernyataan pertama sebuah modul, kelas, atau fungsi, dan disebut **docstring**. Bab 6 membahas keduanya secara mendalam.

Bagian tersulit dari berkomentar adalah menimbang. Bandingkan:

```
int i;                                  // Deklarasikan int bernama i
i = 3;                                  // Isi i dengan 3
System.out.println("i is equal to " + i); // Cetak nilai i
```

Setiap komentar hanya mengulang kode dalam bahasa manusia dan tidak menambahkan apa pun. Aturan yang baik: *jika teman sekelas dapat dengan mudah memahami apa yang dilakukan sebuah baris, jangan beri komentar*. Berilah komentar pada aturan bisnis dan alasannya: "anggota mendapat potongan 10%" adalah komentar yang baik, karena kode hanya memperlihatkan *bahwa* sepersepuluh dikurangkan, bukan *mengapa*.

3.3 Kelas dan Objek

3.3.1 Cetak biru dan rumah

Bayangkan **kelas** sebagai cetak biru. Cetak biru rumah dari seorang arsitek menyatakan bahwa rumah itu memiliki satu pintu depan, tiga jendela, dan dapur berukuran tertentu, tetapi Anda tidak dapat tinggal di dalam cetak biru. Dari satu cetak biru, tukang dapat membangun banyak rumah, masing-masing bangunan nyata yang dapat Anda masuki. Setiap rumah dapat dicat dengan warna berbeda, dan masing-masing memiliki perabotnya sendiri. **Objek** adalah salah satu rumah itu: benda nyata yang dibangun dari kelas dan menyimpan datanya sendiri [2].

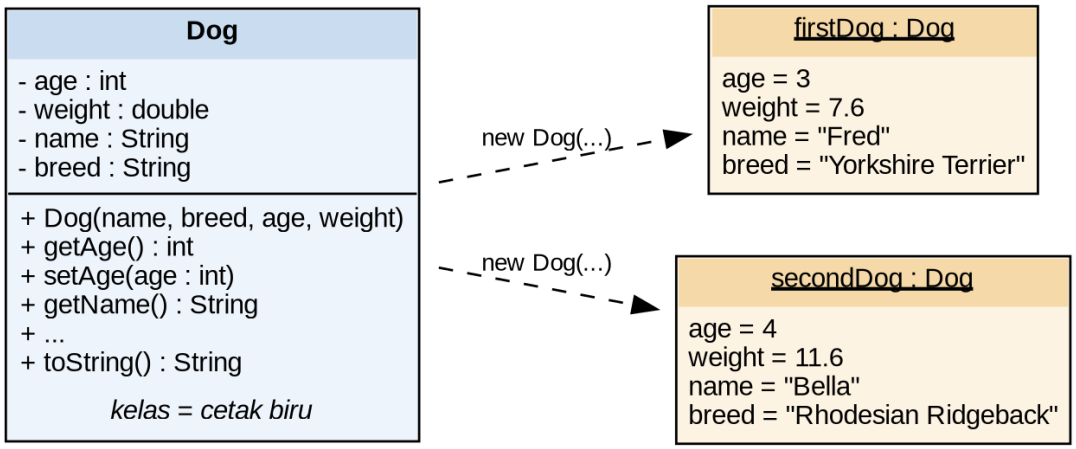
Ambil contoh seekor anjing. Kelas `Dog` menggambarkan apa yang dimiliki setiap anjing dalam program kita: nama, ras, umur, dan berat. Namun, tidak ada orang yang memelihara "anjing pada umumnya": setiap anjing sungguhan adalah anjing tertentu, misalnya Fred, Yorkshire Terrier berumur tiga tahun dengan berat 7,6 kg, atau Bella, Rhodesian Ridgeback berumur empat tahun dengan berat 11,6 kg. Fred dan Bella adalah dua **instans** (*instance*) dari kelas `Dog`; *objek* dan *instans* berarti hal yang sama.

Gambaran kedua membantu banyak pembaca: **lembar kerja** (*spreadsheet*). Kelas serupa dengan baris judul kolom, *nama*, *ras*, *umur*, *berat*. Setiap objek serupa dengan satu baris nilai di bawahnya. Judul kolom menyatakan apa yang dapat dicatat; baris-barisnya adalah catatan itu sendiri.

Tabel 3.1 Kelas dipandang sebagai lembar kerja: baris judul adalah kelas `Dog`, dan setiap baris di bawahnya adalah satu objek `Dog`.

Name	Breed	Age	Weight
Fred	Yorkshire Terrier	3	7.6
Bella	Rhodesian Ridgeback	4	11.6

Tabel 3.1 memperlihatkan gambaran lembar kerja itu, sedangkan **Gambar 3.3** memperlihatkan gagasan yang sama dalam gaya gambar yang dipakai pemrogram sehari-hari: **diagram kelas UML** di kiri dan dua **diagram objek** di kanan. Kotak kelas mendaftar *field* (dengan tanda `-`, artinya privat) dan metode (dengan tanda `+`, artinya publik). Setiap kotak objek memiliki judul bergaris bawah, *nama variabel : nama kelas*, dan mendaftar nilai terkini setiap *field*.



Gambar 3.3 Kelas `Dog` sebagai cetak biru (kiri) dan dua objek yang dibangun darinya dengan `new` (kanan); setiap objek memiliki salinan sendiri dari setiap *field*.

Hal terpenting dalam **Gambar 3.3** mudah terlewat: *setiap objek memiliki salinan sendiri dari setiap field*. `age` milik Fred dan `age` milik Bella adalah dua kotak berbeda di memori. Mengubah yang satu tidak menyentuh yang lain.

3.3.2 Anatomi sebuah kelas

Sekarang mari kita menulis kelasnya. **Kode 3.7** adalah kelas `Dog` lengkap dalam Java.

Kode 3.7 `Dog.java`: empat *field* privat, satu konstruktor, sebuah *getter* dan *setter* untuk setiap *field*, serta `toString`.

```
public class Dog {
    // Field: data yang dibawa setiap objek Dog
    private int age;
    private double weight;
    private String name;
    private String breed;

    /* Tidak ada metode main di sini: Dog bukan program yang berdiri sendiri.
     * Ia hanya memerlukan konstruktor, yang membangun Dog baru.
     */
    public Dog(String name, String breed, int age, double weight) {
        this.name = name;
        this.age = age;
        this.weight = weight;
        this.breed = breed;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

    public double getWeight() {
        return weight;
    }

    public void setWeight(double weight) {
        this.weight = weight;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getBreed() {
        return breed;
    }
```

Kode 3.7 (Lanjutan) *Dog.java: empat field privat, satu konstruktor, sebuah getter dan setter untuk setiap field, serta toString.*

```
    public void setBreed(String breed) {
        this.breed = breed;
    }

    @Override
    public String toString() {
        return "Dog name: " + getName() + "; breed: " + getBreed()
                + "; age: " + getAge() + "; weight: " + getWeight() + " kg";
    }
}
```

Kelas yang sama, sebagaimana tampak di editor Eclipse, ada pada **Gambar 3.4** dan **Gambar 3.5**; nomor baris di tepi kiri memudahkan kita menunjuk bagian-bagiannya sambil membaca.

Dog.java × DogTest.java

```java
public class Dog {
    // Fields: the data every Dog object carries
    private int age;
    private double weight;
    private String name;
    private String breed;

    /* No main method here: Dog is not a program on its own.
     * It only needs a constructor, which builds a new Dog.
     */
    public Dog(String name, String breed, int age, double weight) {
        this.name = name;
        this.age = age;
        this.weight = weight;
        this.breed = breed;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

    public double getWeight() {
        return weight;
    }

    public void setWeight(double weight) {
        this.weight = weight;
    }

```

Gambar 3.4 Bagian pertama `Dog.java` di Eclipse: *field*, konstruktor, serta *getter* dan *setter* yang pertama.

Dog.java × DogTest.java

```java
    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getBreed() {
        return breed;
    }

    public void setBreed(String breed) {
        this.breed = breed;
    }

    @Override
    public String toString() {
        return "Dog name: " + getName() + "; breed: " + getBreed()
                + "; age: " + getAge() + "; weight: " + getWeight() + " kg";
    }
}
```

Gambar 3.5 Sisa `Dog.java` di Eclipse: *getter* dan *setter* yang tersisa, dan `toString`.

Badan kelas berisi tiga jenis **anggota** (*member*). Hafalkan namanya sekarang; setiap bab selanjutnya memakainya.

Field (disebut juga *variabel instans* atau *atribut*) adalah variabel yang dideklarasikan di dalam kelas tetapi di luar metode mana pun. *Field* menyimpan data objek, `int age`, `double weight`, dan setiap objek mendapat salinannya sendiri. Kata `private` berarti hanya kode di dalam `Dog` yang boleh menyentuh *field* itu secara langsung, sehingga kelas dapat menjaga datanya tetap masuk akal, misalnya dengan menolak umur negatif. Menyembunyikan data di balik metode disebut **enkapsulasi**, gagasan besar pertama dalam pemrograman berorientasi objek; Bab 7 membahas kontrol akses secara lengkap.

Konstruktor adalah metode khusus yang membangun objek baru dan memberi *field*-nya nilai awal. Anda dapat mengenalinya karena namanya sama dengan nama kelas dan tidak memiliki tipe kembalian, bahkan `void` pun tidak: `public Dog(String name, String breed, int age, double weight)`. Empat nama di dalam kurung adalah **parameter**-nya: variabel yang menerima nilai yang diberikan oleh pemanggil. Perhatikan bahwa `Dog` tidak memiliki metode `main`. `Dog` bukan program; ia cetak biru yang dipakai program lain.

Metode adalah perilaku objek. `getAge()` mengembalikan umur anjing; `setAge(int age)` mengubahnya; `toString()` menggambarkan anjing itu secara utuh sebagai teks. Deklarasi metode menyatakan apa yang dikembalikan metode (`int`, `String`, atau `void` bila tidak ada), namanya, dan parameternya.

Tabel 3.2 menjajarkan bagian-bagian Java dengan padanannya dalam Python.

Tabel 3.2 Bagian-bagian kelas dalam Java dan dalam Python.

Bagian	Java	Python
Kepala kelas	`public class Dog {`	`class Dog:`
Field	`private int age;` dideklarasikan di badan kelas	`self._age = age` diisi di dalam `__init__`
Konstruktor	`public Dog(String name, ...)`	`def __init__(self, name, ...):`
Objek saat ini	`this` (biasanya boleh dihilangkan)	`self` (selalu ditulis, parameter pertama)
Getter	`public int getAge() { return age; }`	`def get_age(self): return self._age`
Setter	`public void setAge(int age) { this.age = age; }`	`def set_age(self, age): self._age = age`
Bentuk teks	`public String toString()`	`def __str__(self):`
Privat	kata kunci `private`, dipaksakan kompilator	garis bawah di depan `_age`, sebuah konvensi

Berikut kelas Python-nya, **Kode 3.8**.

Kode 3.8 `dog.py`: kelas `Dog` dalam Python; *field* dibuat di dalam `__init__`.

```
class Dog:
    """Seekor anjing dengan nama, ras, umur dalam tahun, dan berat dalam kg."""

    def __init__(self, name, breed, age, weight):
        # Garis bawah di depan berarti: "privat, silakan pakai metodenya"
        self._name = name
        self._age = age
        self._weight = weight
        self._breed = breed

    def get_age(self):
        return self._age

    def set_age(self, age):
        self._age = age

    def get_weight(self):
        return self._weight

    def set_weight(self, weight):
        self._weight = weight

    def get_name(self):
        return self._name

    def set_name(self, name):
        self._name = name

    def get_breed(self):
        return self._breed

    def set_breed(self, breed):
        self._breed = breed

    def __str__(self):
        return ("Dog name: " + self.get_name() + "; breed: " + self.get_breed()
                + "; age: " + str(self.get_age()) + "; weight: "
                + str(self.get_weight()) + " kg")
```

Bacalah dengan berpedoman pada **Tabel 3.2**, dan Anda akan melihat perbedaan yang kecil tetapi nyata.

- Python tidak mendeklarasikan *field* di badan kelas: `self._age = age` di dalam konstruktor menciptakan *field* `_age` pada objek ini.
- Konstruktor selalu bernama `__init__`: Python lebih dulu menciptakan objek kosong, lalu `__init__` *menginisialisasi*-nya.
- Python tidak mempunyai kata kunci `private`. Garis bawah di depan, `_age`, memberi tahu pemrogram lain "ini urusan internal; silakan pakai metodenya"; Python tidak memaksakannya.
- Python tidak menuliskan tipe dalam kelas ini. Variabel Python dapat menyimpan nilai bertipe apa saja; kita kembali ke soal ini di Subbab 3.6.2.

3.3.3 Kata `this`, dan kata `self`

Lihat lagi konstruktor dalam **Kode 3.7**:

```
public Dog(String name, String breed, int age, double weight) {
    this.name = name;
    ...
}
```

Sekarang ada *dua* hal bernama `name`: *field* `name` milik objek, dan parameter `name` yang menyimpan nilai yang diberikan pemanggil. Di dalam konstruktor, `name` saja berarti yang terdekat, yaitu

parameternya. Jadi, apa yang dilakukan `name = name;`? Pernyataan itu menyalin parameter ke dirinya sendiri dan membiarkan *field* tak tersentuh, sehingga anjing baru itu tidak bernama (*field* `name`-nya tetap `null`). Kompilator menerimanya tanpa keluhan; inilah *bug* yang senyap.

Kata kunci `this` memecahkan teka-teki itu. `this` berarti "objek yang sedang dikerjakan oleh kode ini", sehingga `this.name = name;` dibaca "isi *field* `name` *milik objek ini* dengan parameter `name`". Bila tidak ada bentrokan nama, seperti dalam `getAge()` yang berbunyi `return age;`, Anda boleh menghilangkan `this`.

Python membuat gagasan ini eksplisit dan wajib. Setiap metode menerima objek saat ini sebagai parameter pertamanya, yang menurut kebiasaan dinamai `self`, dan Anda harus selalu menulis `self._age` untuk menjangkau sebuah *field*. `_age` saja di dalam metode adalah variabel lokal, bukan *field*. Ketika Anda menulis `first_dog.get_age()`, diam-diam Python memanggil `Dog.get_age(first_dog)`: objek di depan titik menjadi `self`.

> **DEFINISI.** Di dalam metode atau konstruktor, **`this`** (Java) atau **`self`** (Python) merujuk ke objek tempat metode itu dipanggil, atau, di dalam konstruktor, ke objek yang sedang dibangun. `this.x` atau `self.x` adalah *field* `x` milik objek itu.

3.3.4 *Getter*, *setter*, dan `toString`

Karena *field*-nya privat, dunia luar memerlukan jalan masuk yang santun. Itulah gunanya ***getter*** dan ***setter***.

Getter mengembalikan nilai satu *field*: `public String getName() { return name; }`. Kata kunci `return` mengakhiri metode dan menyerahkan nilai kepada siapa pun yang memanggilnya. Metode yang tipe kembaliannya bukan `void` wajib berakhir dengan pernyataan `return` pada setiap jalur; kompilator memeriksanya.

Setter mengubah nilai satu *field*: `public void setName(String name) { this.name = name; }`. Tipe kembaliannya `void`, karena tidak ada yang perlu diserahkan kembali, sehingga ia tidak memerlukan `return`.

Mengapa *field*-nya tidak dijadikan publik saja? Karena *setter* adalah gerbang. Besok Anda dapat menambahkan satu baris pada `setAge`, `if (age < 0) return;`, dan setiap pemanggil langsung terlindungi; *field* publik tidak memiliki gerbang yang dapat ditambahkan. Sebuah kelas juga boleh sama sekali tidak menyediakan *setter*, seperti `Customer` di Subbab 3.12 untuk data yang tidak boleh berubah.

`toString` mengembalikan deskripsi teks objek. Setiap objek Java mewarisi satu `toString` dari kelas `Object` (Bab 7), tetapi hasil cetaknya seperti `Dog@1b6d3586`, yang tampak tak bermakna; dengan menulis versi kita sendiri, kita menentukan rupa seekor anjing ketika dicetak. `@Override` memberi tahu kompilator "saya berniat mengganti metode warisan", sehingga salah eja `tostring` dilaporkan sebagai galat. `toString` kita memanggil *getter* alih-alih membaca *field* secara langsung, sehingga pekerjaan apa pun yang kelak dilakukan *getter* (memformat, menghitung) otomatis ikut dinikmati `toString`.

Di Python, tugas yang sama dikerjakan oleh `__str__`. Setiap kali Python memerlukan sebuah objek dalam bentuk teks, dalam `print(dog)` atau `str(dog)`, Python memanggil `__str__`.

3.3.5 Membuat objek dengan `new`

Kelas saja tidak melakukan apa pun. Untuk memakainya, kita harus **membuat** (atau **menginstansiasi**) objek. Di Java caranya adalah kata kunci `new`:

```
Dog fido = new Dog("Fido", "Jack Russell", 5, 12.5);
```

Tiga hal terjadi dalam satu baris itu. Pertama, `new` meminta Java mencarikan tempat di memori untuk objek `Dog` baru, dengan ruang bagi keempat *field*-nya. Kedua, Java memanggil konstruktor sambil meneruskan empat **argumen** `"Fido"`, `"Jack Russell"`, `5`, dan `12.5`; konstruktor menyalinnya ke dalam *field*. Ketiga, `new` mengembalikan sebuah **referensi** ke objek baru itu, semacam panah yang menunjuk kepadanya, dan penugasan menyimpan panah itu dalam variabel `fido`.

Argumen harus cocok dengan parameter dalam jumlah, urutan, dan tipe: `new Dog("Fido", 5)` tidak dapat dikompilasi, begitu pula `new Dog("Fido", "Jack Russell", 12.5, 5)`, karena `12.5` tidak dapat masuk ke parameter `int` bernama `age`.

Di Python tidak ada kata kunci `new`; Anda memanggil kelas seolah-olah fungsi, `Dog("Fido", "Jack Russell", 5, 12.5)`, lalu Python menciptakan objek dan menjalankan `__init__` padanya.

Setelah objek ada, kita memanggil metodenya dengan titik: `fido.getAge()`, `fido.setWeight(13.0)`. **Kode 3.9** menguji dua ekor anjing.

Kode 3.9 `DogDemo.java`: membuat dua objek `Dog`, mencetaknya, dan mengubahnya melalui *setter*.

```java
public class DogDemo {

    public static void main(String[] args) {
        Dog firstDog = new Dog("Fred", "Yorkshire Terrier", 3, 7.6);
        Dog secondDog = new Dog("Bella", "Rhodesian Ridgeback", 4, 11.6);

        System.out.println(firstDog);
        System.out.println(secondDog.toString()); // efeknya sama dengan baris di atas

        firstDog.setAge(6);
        secondDog.setWeight(14.0);
        System.out.println(secondDog.getName() + " is now "
                + secondDog.getWeight() + " kg"); // seharusnya 14.0
        System.out.println(firstDog.getName() + " is now "
                + firstDog.getAge() + " years old"); // seharusnya 6
    }
}
```

Untuk mengompilasi dan menjalankan, kedua berkas harus dikompilasi bersama, karena `DogDemo` memakai `Dog`:

```
javac Dog.java DogDemo.java
java DogDemo
```

```
Dog name: Fred; breed: Yorkshire Terrier; age: 3; weight: 7.6 kg
Dog name: Bella; breed: Rhodesian Ridgeback; age: 4; weight: 11.6 kg
Bella is now 14.0 kg
Fred is now 6 years old
```

Bacalah keluaran itu sambil mencocokkannya dengan kode. `println(firstDog)` memerlukan teks, sehingga ia memanggil `firstDog.toString()` untuk kita; baris kedua memanggilnya secara eksplisit, dengan efek yang sama. Kemudian *setter* mengubah umur Fred dan berat Bella, dan *getter* mengonfirmasinya. Perubahan pada Fred tidak menyentuh Bella: setiap objek memiliki *field*-nya sendiri.

Program penggerak Python dalam **Kode 3.10** mengimpor kelas dari `dog.py` dengan `from dog import Dog`. Agar impor berhasil, kedua berkas harus berada dalam folder yang sama.

Kode 3.10 `dog_demo.py`: peragaan yang sama dalam Python.

```python
from dog import Dog


def main():
    first_dog = Dog("Fred", "Yorkshire Terrier", 3, 7.6)
    second_dog = Dog("Bella", "Rhodesian Ridgeback", 4, 11.6)

    print(first_dog)
    print(str(second_dog))  # efeknya sama dengan baris di atas

    first_dog.set_age(6)
    second_dog.set_weight(14.0)
    print(second_dog.get_name() + " is now "
          + str(second_dog.get_weight()) + " kg")  # seharusnya 14.0
    print(first_dog.get_name() + " is now "
          + str(first_dog.get_age()) + " years old")  # seharusnya 6


if __name__ == "__main__":
    main()
```

```
Dog name: Fred; breed: Yorkshire Terrier; age: 3; weight: 7.6 kg
Dog name: Bella; breed: Rhodesian Ridgeback; age: 4; weight: 11.6 kg
Bella is now 14.0 kg
Fred is now 6 years old
```

Satu perbedaan perlu dicatat. Operator `+` di Java mengubah bilangan menjadi teks bila sisi lainnya string; Python menuntut `str(...)` yang eksplisit. Subbab 3.6.2 menjelaskan alasannya.

3.3.6 Memeriksa bahwa kelas bekerja

Bagaimana kita tahu bahwa `Dog` benar? Dengan membuat objek, memanggil metode, dan memeriksa bahwa nilai yang kembali adalah nilai yang kita harapkan. Pemrogram menyebutnya **pengujian** (*testing*), dan itulah pembeda antara berharap dan mengetahui.

Pengujian paling sederhana adalah program biasa yang membandingkan nilai harapan dengan nilai sebenarnya lalu menyatakan PASS atau FAIL. **Kode 3.11** memeriksa bahwa `getAge()` mengembalikan umur yang diberikan kepada konstruktor, dan bahwa berat yang diisi dengan `setWeight` adalah berat yang terbaca kembali; **Kode 3.12** adalah kembarannya dalam Python.

Kode 3.11 `DogCheck.java`: pengujian buatan tangan untuk `getAge` dan `setWeight`.

```java
public class DogCheck {

    public static void main(String[] args) {
        Dog firstDog = new Dog("Fred", "Yorkshire Terrier", 3, 7.6);
        int expected = 3;                  // nilai yang seharusnya dikembalikan getAge()
        int actual = firstDog.getAge();    // nilai yang benar-benar dikembalikan

        if (expected == actual) {
            System.out.println("PASS: getAge() returned " + actual);
        } else {
            System.out.println("FAIL: expected " + expected + " but got " + actual);
        }

        firstDog.setWeight(8.1);
        double expectedWeight = 8.1;
        if (Math.abs(firstDog.getWeight() - expectedWeight) < 1e-9) {
            System.out.println("PASS: setWeight() then getWeight() gave " +
firstDog.getWeight());
        } else {
            System.out.println("FAIL: weight is " + firstDog.getWeight());
        }
    }
}
```

```
PASS: getAge() returned 3
PASS: setWeight() then getWeight() gave 8.1
```

Kode 3.12 `dog_check.py`: pemeriksaan yang sama dalam Python.

```python
from dog import Dog


def main():
    first_dog = Dog("Fred", "Yorkshire Terrier", 3, 7.6)
    expected = 3                   # nilai yang seharusnya dikembalikan get_age()
    actual = first_dog.get_age()   # nilai yang benar-benar dikembalikan

    if expected == actual:
        print("PASS: get_age() returned " + str(actual))
    else:
        print("FAIL: expected " + str(expected) + " but got " + str(actual))

    first_dog.set_weight(8.1)
    expected_weight = 8.1
    if abs(first_dog.get_weight() - expected_weight) < 1e-9:
        print("PASS: set_weight() then get_weight() gave "
              + str(first_dog.get_weight()))
    else:
        print("FAIL: weight is " + str(first_dog.get_weight()))


if __name__ == "__main__":
    main()
```

```
PASS: get_age() returned 3
PASS: set_weight() then get_weight() gave 8.1
```

Perhatikan bahwa berat tidak dibandingkan dengan `==`, tetapi dengan menanyakan apakah selisihnya sangat kecil. Subbab 3.7.3 menjelaskan mengapa membandingkan dua bilangan desimal dengan `==` adalah jebakan.

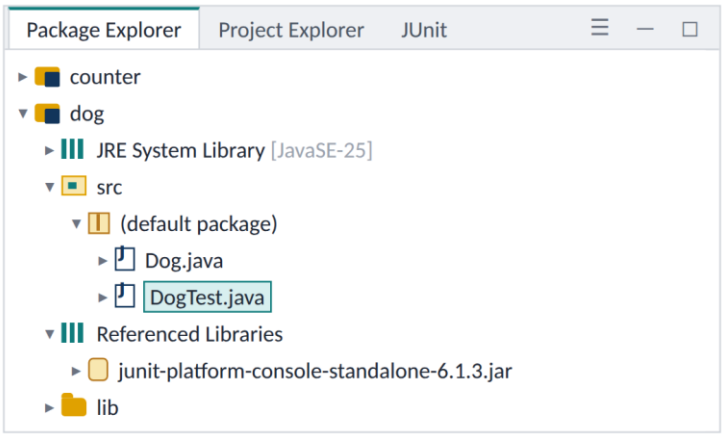
Menulis pemeriksaan semacam ini dengan tangan lekas melelahkan, sehingga pemrogram profesional memakai **kerangka kerja pengujian** (*testing framework*), yaitu pustaka yang menjalankan banyak pengujian kecil dan melaporkan hasilnya; di Java, yang baku adalah **JUnit**,

yang versi terkininya adalah JUnit 6 [3]. **Gambar 3.6** memperlihatkan kelas uji JUnit, `DogTest`, untuk pemeriksaan `getAge` yang sama, dan **Gambar 3.7** memperlihatkannya dalam proyek Eclipse bernama `dog`, bersebelahan dengan `Dog.java`, dengan berkas pustaka JUnit ditambahkan ke proyek.

Dog.java | DogTest.java

```
1  import static org.junit.jupiter.api.Assertions.assertEquals;
2
3  import org.junit.jupiter.api.Test;
4
5  class DogTest {
6
7      /**
8       * Testing the getAge() method.
9       */
10     @Test
11     void test1() {
12         Dog firstDog = new Dog("Fred", "Yorkshire Terrier", 3, 7.6);
13         // 3 is what I expect if getAge() is called
14         int expected = 3;
15         // The actual value
16         int firstDogAge = firstDog.getAge();
17         // By JUnit convention the expected value comes first;
18         // the message is shown only if the assertion fails
19         assertEquals(expected, firstDogAge, "test1 of getAge() in DogTest");
20     }
21 }
```

Gambar 3.6 Kelas uji JUnit `DogTest` di Eclipse; metode bertanda `@Test` membuat seekor anjing dan memakai `assertEquals` untuk membandingkan umur harapan dengan umur sebenarnya.

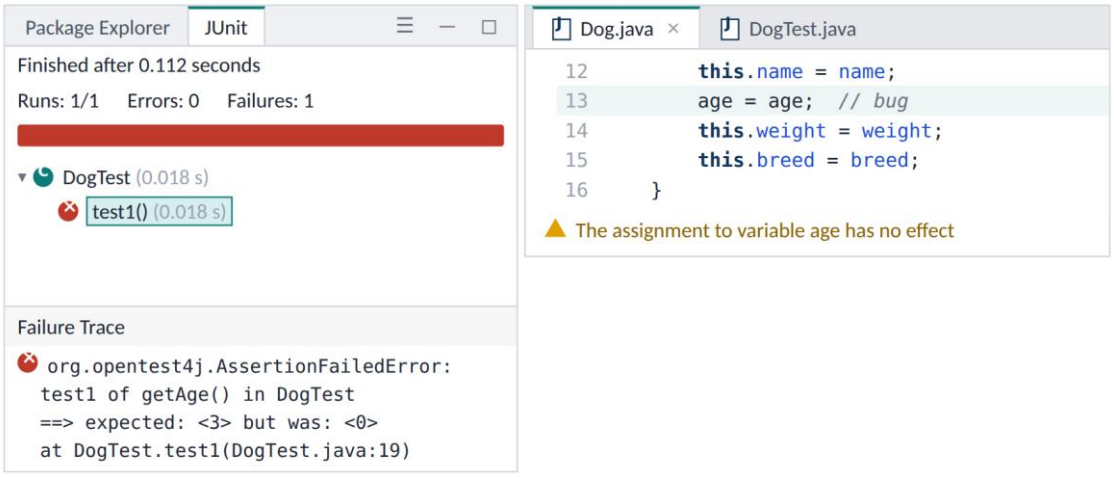


Gambar 3.7 Package Explorer di Eclipse: `Dog.java` dan `DogTest.java` di dalam proyek `dog`, dengan berkas pustaka JUnit di bawah *Referenced Libraries*.

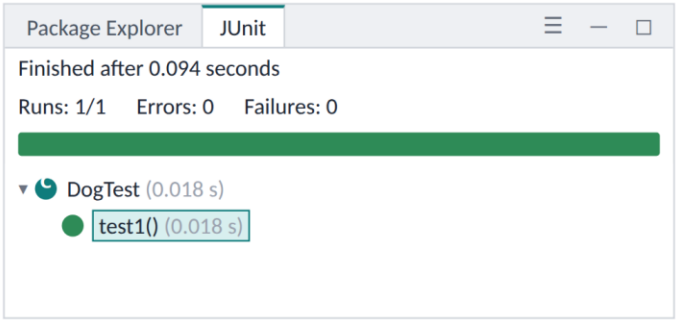
Pengujian itu membuat Fred lalu memanggil `assertEquals(expected, firstDogAge, message)`, dengan nilai harapan di depan sesuai konvensi JUnit; bila keduanya berbeda, pengujian gagal dan JUnit melaporkan pesan itu beserta kedua nilainya.

Eclipse melaporkan hasilnya dengan sebuah bilah berwarna. **Gambar 3.8** memperlihatkan bilah merah, dan kegagalannya sangat mendidik. Pada versi `Dog` ini konstruktornya menulis `age = age;` alih-alih `this.age = age;`, sehingga parameter `age` diberikan kepada dirinya sendiri dan *field*-nya tetap bernilai bawaan 0; Eclipse bahkan memperingatkan bahwa penugasan itu tidak berpengaruh apa-apa. Pengujian mengharapkan 3, objeknya menjawab 0, dan *Failure Trace*

mengatakan persis hal itu: `expected: <3> but was: <0>`, beserta pesannya dan baris penegasannya. Setelah `this.` dikembalikan, bilah berubah hijau, seperti pada **Gambar 3.9**: satu pengujian dijalankan, tanpa galat, tanpa kegagalan. Pengujian yang mengubah bug senyap menjadi bilah merah dengan kedua nilai berdampingan telah menjalankan tugasnya.



Gambar 3.8 Bilah merah: `test1` gagal dengan harapan 3 tetapi ternyata 0, karena konstruktor `Dog` versi ini menulis `age = age;` alih-alih `this.age = age;`.



Gambar 3.9 Bilah hijau: setelah diperbaiki, pengujian berjalan dan lulus.

Bab 6 menyiapkan JUnit 6 dan `unittest` milik Python dengan semestinya dan menjadikan pengujian kebiasaan sehari-hari. Untuk sekarang, ingatlah pola setiap pengujian: *siapkan* sebuah objek, *lakukan* sesuatu padanya dengan memanggil metode, lalu *tegaskan* (*assert*) bahwa hasilnya sesuai harapan.

3.4 Referensi dan Gambar Memori

3.4.1 Nilai dan panah

Berikut sebuah pertanyaan yang memisahkan pemrogram yang memahami objek dari yang sekadar memakainya. Setelah baris-baris ini, berapakah umur Fido?

```
Dog fido = new Dog("Fido", "Jack Russell", 5, 12.5);
Dog alias = fido;
alias.setAge(6);
```

Jika Anda menjawab “tetap 5, karena yang kita ubah `alias`, bukan `fido`”, Anda tidak sendirian, tetapi jawaban itu salah. Jawabannya 6. Untuk memahami sebabnya, kita memerlukan gambar memori.

Java memiliki dua keluarga tipe. **Tipe primitif**, seperti `int`, `double`, `char`, dan `boolean`, adalah nilai sederhana. Variabel bertipe primitif adalah kotak yang menyimpan nilai itu sendiri. **Tipe referensi** adalah semua yang lain: `String`, `Dog`, setiap kelas yang akan pernah Anda tulis. Variabel bertipe referensi *tidak* menyimpan objeknya. Ia menyimpan sebuah **referensi**, yaitu panah yang menunjuk ke tempat objek itu berada. Objek-objek itu sendiri tinggal di wilayah memori yang disebut ***heap***; variabel-variabel milik metode yang sedang berjalan tinggal di sebuah **bingkai tumpukan** (*stack frame*).

Perbedaan itu mengubah arti `=`. Untuk primitif, `int b = a;` menyalin *nilai*: kini dua kotak terpisah berisi bilangan yang sama, dan mengubah yang satu tidak menyentuh yang lain. Untuk referensi, `Dog alias = fido;` menyalin *panah*: kini dua panah menunjuk ke objek yang *sama*, dan *setter* yang dipanggil melalui salah satunya mengubah satu-satunya objek yang dilihat keduanya. Dua variabel yang merujuk ke objek yang sama disebut **alias**.

Kode 3.13 memperagakan semua ini, dan **Gambar 3.10** menggambar memori tepat sebelum dua pernyataan terakhir.

Kode 3.13 `ReferenceDemo.java`: menyalin primitif berarti menyalin nilai; menyalin referensi berarti menyalin panah.

```
public class ReferenceDemo {

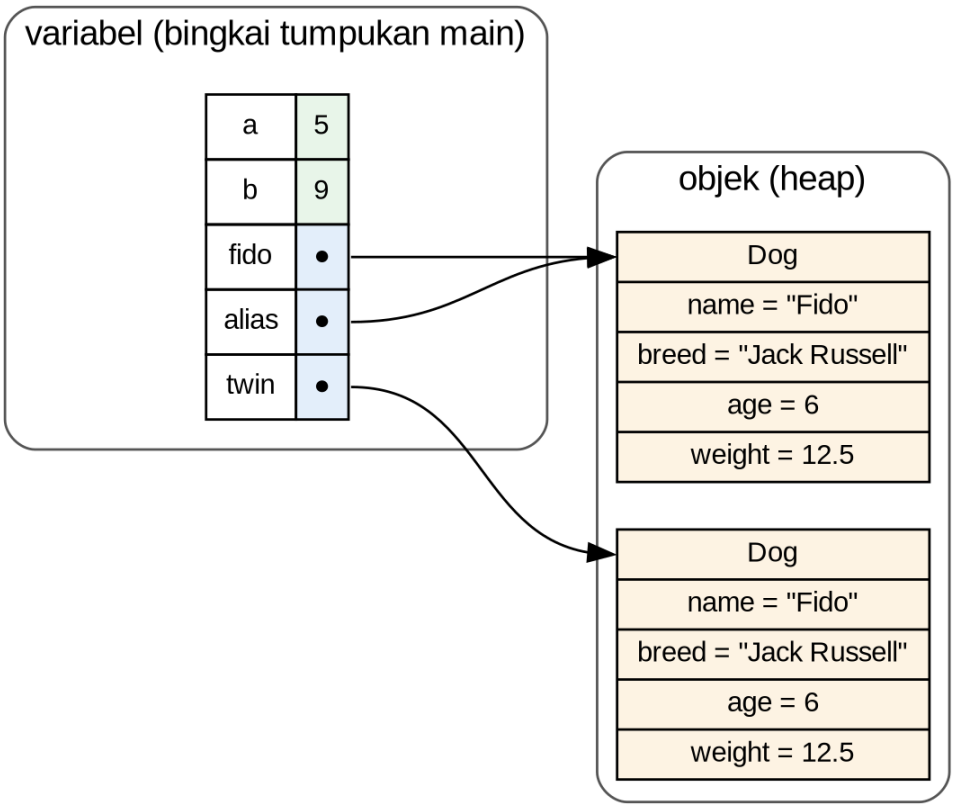
    public static void main(String[] args) {
        // Variabel primitif menyimpan nilainya sendiri.
        int a = 5;
        int b = a;          // b mendapat SALINAN nilai 5
        b = 9;
        System.out.println("a = " + a + ", b = " + b);

        // Variabel referensi menyimpan panah ke sebuah objek.
        Dog fido = new Dog("Fido", "Jack Russell", 5, 12.5);
        Dog alias = fido;   // alias mendapat salinan PANAH, bukan salinan anjingnya
        alias.setAge(6);
        System.out.println("fido's age seen through fido: " + fido.getAge());

        Dog twin = new Dog("Fido", "Jack Russell", 6, 12.5);
        System.out.println("fido == alias: " + (fido == alias));
        System.out.println("fido == twin:  " + (fido == twin));

        alias = null;       // alias kini tidak menunjuk ke mana pun; fido tidak tersentuh
        System.out.println("alias is null: " + (alias == null));
        System.out.println("fido still: " + fido);
    }
}
```

```
a = 5, b = 9
fido's age seen through fido: 6
fido == alias: true
fido == twin:  false
alias is null: true
fido still: Dog name: Fido; breed: Jack Russell; age: 6; weight: 12.5 kg
```



Gambar 3.10 Memori di dalam `main` pada **Kode 3.13**: `a` dan `b` menyimpan nilai, sedangkan `fido`, `alias`, dan `twin` menyimpan panah; `fido` dan `alias` menunjuk ke objek yang sama.

Telusuri keluaran itu dengan gambar di depan Anda.

1. `a = 5, b = 9`: `b` mendapat salinan 5, lalu ditimpa dengan 9; `a` tidak tersentuh.
2. `fido's age seen through fido: 6`: kita mengubah umur melalui `alias`, tetapi `alias` dan `fido` menunjuk ke anjing yang sama.
3. `fido == alias: true`: untuk referensi, `==` bertanya "apakah kedua panah ini menunjuk ke objek yang *sama*?" Ya.
4. `fido == twin: false`: `twin` adalah objek yang *berbeda* dengan nilai *field* yang sama. `==` membandingkan panah, bukan isi. Dua objek dengan isi yang sama disebut *setara* (*equal*); dua panah ke satu objek disebut *identik*.
5. `alias = null` membuat `alias` tidak menunjuk ke mana pun; `null` berarti "tidak ada objek". Anjingnya tidak terpengaruh, dan `fido` masih menjangkaunya.

Apa yang terjadi pada objek yang tidak lagi ditunjuk oleh panah mana pun? Anda tidak perlu berbuat apa-apa: dari waktu ke waktu, sistem waktu-jalan (*runtime*) Java menyadari bahwa objek itu tidak terjangkau dan mengambil kembali memorinya. Proses ini disebut **pengumpulan sampah** (*garbage collection*). Python melakukan hal yang sama.

> **PERINGATAN:** Memanggil metode melalui referensi `null`, misalnya `alias.getAge()` sesudah `alias = null`, menghentikan program dengan `NullPointerException`. Inilah galat waktu-jalan yang paling umum di Java. Sebelum memanggil metode pada referensi yang mungkin `null`, periksalah dulu: `if (alias != null) { ... }`.

3.4.2 Python: setiap variabel adalah nama

Python menyederhanakan gambaran itu dengan hanya memiliki satu keluarga: *setiap* nilai adalah objek, termasuk bilangan bulat 5, dan *setiap* variabel adalah nama yang terikat ke sebuah objek. Penugasan tidak pernah menyalin objek; ia mengikatkan nama ke objek. **Kode 3.14** adalah kembaran peragaan Java tadi dalam Python.

Kode 3.14 `reference_demo.py`: dalam Python, setiap variabel adalah nama yang terikat ke sebuah objek.

```
from dog import Dog


def main():
    # int adalah objek imutabel: "mengubah" b mengikat b ke objek lain.
    a = 5
    b = a
    b = 9
    print("a = " + str(a) + ", b = " + str(b))

    # Setiap variabel Python adalah nama yang terikat ke sebuah objek.
    fido = Dog("Fido", "Jack Russell", 5, 12.5)
    alias = fido  # alias adalah nama kedua untuk anjing yang SAMA
    alias.set_age(6)
    print("fido's age seen through fido: " + str(fido.get_age()))

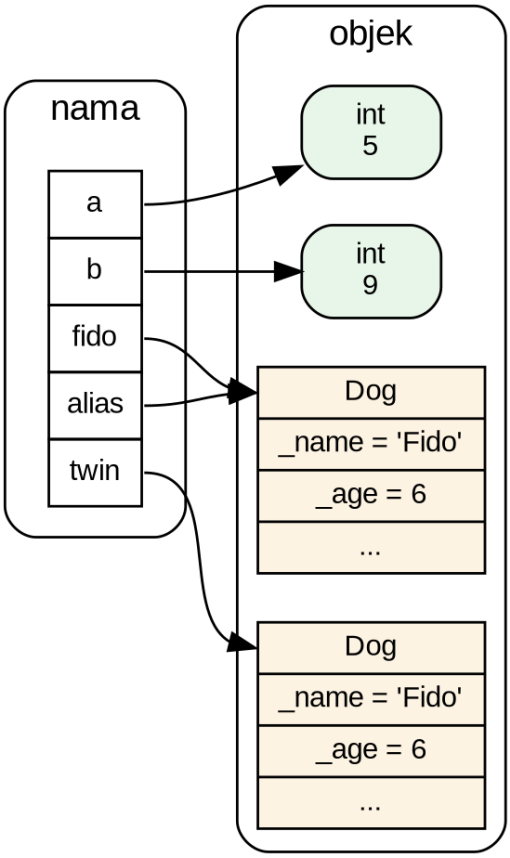
    twin = Dog("Fido", "Jack Russell", 6, 12.5)
    print("fido is alias: " + str(fido is alias))
    print("fido is twin:  " + str(fido is twin))

    alias = None  # alias kini tidak menamai apa pun; fido tidak tersentuh
    print("alias is None: " + str(alias is None))
    print("fido still: " + str(fido))


if __name__ == "__main__":
    main()
```

```
a = 5, b = 9
fido's age seen through fido: 6
fido is alias: True
fido is twin:  False
alias is None: True
fido still: Dog name: Fido; breed: Jack Russell; age: 6; weight: 12.5 kg
```

Hasilnya sama dengan Java, dan **Gambar 3.11** memperlihatkan sebabnya. `b = a` mengikat nama `b` ke objek bilangan bulat yang sama dengan `a`; kemudian `b = 9` *mengikat ulang* `b` ke objek lain. Objek 5 tidak berubah, karena bilangan bulat Python bersifat **imutabel** (tak dapat diubah): tidak ada operasi yang mengubah nilai di dalamnya; sebuah nama hanya dapat dipindahkan ke objek lain. Anjing itu **mutabel**: `set_age` mengubahnya di tempat, dan setiap nama yang terikat kepadanya melihat perubahan itu.



Gambar 3.11 Nama dan objek dalam **Kode 3.14**: setiap nama adalah panah; `fido` dan `alias` terikat ke objek `Dog` yang sama.

Python menuliskan identitas sebagai `is` dan "tidak ada objek" sebagai `None`. Operator `==` di Python menanyakan kesetaraan *isi* (untuk kelas yang mendefinisikan arti isi yang setara), yaitu pertanyaan yang di Java diajukan dengan metode `equals`. Kita akan melihat bahwa hal ini sangat penting untuk string.

Tabel 3.3 Identitas, kesetaraan, dan "tidak ada objek" dalam kedua bahasa.

Pertanyaan	Java	Python
Objek yang sama? (identitas)	`a == b` untuk referensi	`a is b`
Isi yang sama? (kesetaraan)	`a.equals(b)`	`a == b`
Tidak ada objek	`null`	`None`
Uji tidak ada objek	`x == null`	`x is None`

Tabel 3.3 layak disalin ke selembar kartu: kedua bahasa menuliskan kedua pertanyaan itu secara berkebalikan.

3.4.3 Cara menggambar diagram objek

Menggambar memori adalah cara paling ampuh untuk memahami program yang memakai objek, dan hanya memerlukan sebatang pensil. Berikut aturan-aturan yang mendasari **Gambar 3.10**.

1. Gambarlah satu kotak untuk setiap variabel dalam metode yang sedang Anda telusuri, beri label nama variabelnya. Susun kotak-kotak itu di sebelah kiri.
2. Untuk variabel primitif (`int`, `double`, `boolean`, `char`), tulis nilainya di dalam kotak.

3. Untuk variabel referensi, gambarlah sebuah titik di dalam kotak dan panah dari titik itu ke sebuah objek, atau tulis `null` bila ia tidak menunjuk ke mana pun.
4. Gambarlah setiap objek satu kali, di sebelah kanan, sebagai kotak dengan nama kelas di atasnya dan satu baris per *field*, *field = nilai*. *Field* yang juga berupa referensi (seperti `name`, sebuah `String`) boleh digambar dengan nilainya di antara tanda petik agar ringkas.
5. Setiap kali sebuah pernyataan dijalankan, perbarui gambarnya: `new` menambahkan objek; `x = y` untuk referensi memindahkan panah `x` ke mana pun panah `y` menunjuk; *setter* mengubah nilai di dalam kotak objek.
6. Objek yang tidak ditunjuk panah apa pun adalah sampah; coretlah.

Lakukan ini untuk setiap program dalam bab ini setidaknya sekali, dan pertanyaan "mengapa objek saya berubah?" tidak akan lagi mengejutkan Anda.

3.5 String

3.5.1 Teks adalah objek

String adalah deretan karakter: sebuah kata, kalimat, nama, atau nomor telepon. Di Java, string bertipe `String`, dan tipe itu adalah tipe *objek*; itulah sebabnya `String` diawali huruf kapital. Di balik layar, string dibangun dari tipe primitif yang lebih sederhana, `char`, yang menyimpan tepat satu karakter. Keduanya ditulis secara berbeda:

```
String myString = "Hello, my name is Yarik";
char myCharacter = 'y';
```

String ditulis di antara petik ganda; karakter tunggal di antara petik tunggal. `"y"` adalah `String` sepanjang satu; `'y'` adalah `char`. Keduanya tipe yang berbeda dan tidak dapat saling menggantikan begitu saja.

Python tidak mempunyai tipe karakter tersendiri. Tipe `str`-nya mencakup teks sepanjang apa pun, dan sebuah "karakter" hanyalah string sepanjang satu. Python menerima kedua gaya petik, `'y'` dan `"y"`, dan artinya persis sama.

Sebagian besar pemrograman sesungguhnya berurusan dengan teks: nama, kode, nota, data yang diketik manusia. Kelas `String` di Java memiliki puluhan metode, yang diuraikan dalam dokumentasi API Java [4], dan `str` milik Python sama kayanya [5]. Berikut metode-metode yang akan Anda pakai setiap pekan.

3.5.2 Posisi dan metode sehari-hari

Karakter-karakter sebuah string dinomori mulai dari 0. Dalam `"abcde"`, karakter pada posisi 0 adalah `a`, dan karakter pada posisi 3 adalah `d`. Banyaknya karakter adalah **panjang** string, di sini 5, sehingga posisi terakhir selalu `length - 1`.

Kode 3.15 melatih metode-metode sehari-hari, dimulai dengan contoh-contoh dari Subbab 3.2.

Kode 3.15 `StringBasics.java`: `startsWith`, `substring`, `charAt`, `length`, dan metode `String` sehari-hari lainnya.

```java
public class StringBasics {

    public static void main(String[] args) {
        String myString = "the quick brown fox jumps over the lazy dog";
        if (myString.startsWith("the")) {
            System.out.println("It does!");
        } else {
            System.out.println("It doesn't!");
        }

        String mySecondString = "HelloYarik";
        String rest = mySecondString.substring(5);         // dari indeks 5 sampai akhir
        String firstWord = mySecondString.substring(0, 5); // indeks 0,1,2,3,4
        System.out.println(rest + " / " + firstWord);

        String anotherString = "abcde";
        char characterThree = anotherString.charAt(3);
        System.out.println("charAt(3) = " + characterThree);
        System.out.println("length() = " + anotherString.length());

        String service = "  Express Wash  ";
        System.out.println("[" + service.trim() + "]");
        System.out.println(service.trim().toUpperCase());
        System.out.println(service.trim().toLowerCase());
        System.out.println("indexOf(\"Wash\") = " + service.trim().indexOf("Wash"));
        System.out.println("contains(\"press\") = " + service.contains("press"));
        System.out.println(service.trim().replace("Wash", "Iron"));
        System.out.println("endsWith(\"dog\") = " + myString.endsWith("dog"));

        String[] words = myString.split(" ");
        System.out.println("number of words = " + words.length);
        System.out.println("equalsIgnoreCase: " + "WASH".equalsIgnoreCase("wash"));
    }
}
```

```
It does!
Yarik / Hello
charAt(3) = d
length() = 5
[Express Wash]
EXPRESS WASH
express wash
indexOf("Wash") = 8
contains("press") = true
Express Iron
endsWith("dog") = true
number of words = 9
equalsIgnoreCase: true
```

Mari kita bahas satu per satu.

- **`startsWith(prefix)`** menjawab pertanyaan ya/tidak: apakah string diawali karakter-karakter ini? Metode ini mengembalikan `boolean`, sehingga dapat langsung diletakkan di dalam `if`. Di sini kalimatnya diawali `the`, sehingga kita melihat `It does!`. `endsWith` menanyakan hal yang sama tentang bagian akhir.
- **`substring(begin)`** dan **`substring(begin, end)`** memotong sebagian string. Bentuk pertama mengambil semuanya dari posisi `begin` sampai akhir: `"HelloYarik".substring(5)` adalah `"Yarik"`. Bentuk kedua mengambil karakter dari `begin` sampai *tetapi tidak termasuk* `end`: `substring(0, 5)` adalah posisi 0, 1, 2, 3, dan 4, yaitu `"Hello"`. Aturan "awal disertakan, akhir tidak" tampak ganjil pada mulanya, tetapi memiliki sifat yang elok: panjang potongan cukup `end - begin`, dan `substring(0, 5)` serta `substring(5)` bersambung tepat, tanpa tumpang tindih dan tanpa celah.

- `charAt(index)` mengembalikan satu `char` pada sebuah posisi: `"abcde".charAt(3)` adalah `'d'`.
- `length()` mengembalikan banyaknya karakter sebagai `int`.
- `trim()` membuang spasi di kedua ujung; `toUpperCase()` dan `toLowerCase()` mengubah huruf besar-kecil; `indexOf(text)` menemukan tempat sepotong teks pertama kali muncul (atau mengembalikan `-1` bila tidak muncul); `contains(text)` menanyakan apakah teks itu muncul; `replace(old, new)` menukar setiap kemunculan satu potongan dengan potongan lain; `split(" ")` memotong string di setiap spasi dan mengembalikan potongan-potongannya; `equalsIgnoreCase` membandingkan dua string tanpa memedulikan huruf kapital.

Python menuliskan sebagian besar operasi ini dengan operator, bukan metode, seperti diperlihatkan **Kode 3.16**.

Kode 3.16 `string_basics.py`: operasi yang sama dengan `str` milik Python.

```python
def main():
    my_string = "the quick brown fox jumps over the lazy dog"
    if my_string.startswith("the"):
        print("It does!")
    else:
        print("It doesn't!")

    my_second_string = "HelloYarik"
    rest = my_second_string[5:]         # dari indeks 5 sampai akhir
    first_word = my_second_string[0:5]  # indeks 0,1,2,3,4
    print(rest + " / " + first_word)

    another_string = "abcde"
    character_three = another_string[3]
    print("[3] = " + character_three)
    print("len() = " + str(len(another_string)))

    service = "  Express Wash  "
    print("[" + service.strip() + "]")
    print(service.strip().upper())
    print(service.strip().lower())
    print('find("Wash") = ' + str(service.strip().find("Wash")))
    print('"press" in service = ' + str("press" in service))
    print(service.strip().replace("Wash", "Iron"))
    print('endswith("dog") = ' + str(my_string.endswith("dog")))

    words = my_string.split(" ")
    print("number of words = " + str(len(words)))
    print("casefold compare: " + str("WASH".casefold() == "wash".casefold()))


if __name__ == "__main__":
    main()
```

```
It does!
Yarik / Hello
[3] = d
len() = 5
[Express Wash]
EXPRESS WASH
express wash
find("Wash") = 8
"press" in service = True
Express Iron
endswith("dog") = True
number of words = 9
casefold compare: True
```

Python memakai kurung siku untuk posisi, `s[3]`, dan untuk potongan, `s[5:]` dan `s[0:5]`. **Irisan** (*slice*) ini mengikuti aturan "awal disertakan, akhir tidak" yang sama dengan `substring` di Java. Panjang string diperoleh dengan fungsi bawaan `len(s)`, bukan metode. "Apakah memuat?" ditulis dengan operator `in`. Python juga membolehkan posisi *negatif*, dihitung dari akhir: `s[-1]` adalah karakter terakhir, dan `s[-3:]` adalah tiga karakter terakhir. Kelas Laundry di Subbab 3.12 memakai cara ini untuk menampilkan hanya digit-digit terakhir nomor telepon.

Tabel 3.4 adalah ringkasan untuk disimpan di dekat Anda.

Tabel 3.4 Operasi string sehari-hari dalam Java dan Python.

Tugas	Java (`String s`)	Python (`str s`)
Panjang	`s.length()`	`len(s)`
Karakter pada posisi `i`	`s.charAt(i)`	`s[i]`
Dari `i` sampai akhir	`s.substring(i)`	`s[i:]`
Dari `i` sampai, tidak termasuk, `j`	`s.substring(i, j)`	`s[i:j]`
Karakter terakhir	`s.charAt(s.length() - 1)`	`s[-1]`
Diawali / diakhiri	`s.startsWith(p)`, `s.endsWith(p)`	`s.startswith(p)`, `s.endswith(p)`
Memuat	`s.contains(t)`	`t in s`
Posisi pertama `t`	`s.indexOf(t)` (`-1` bila tidak ada)	`s.find(t)` (`-1` bila tidak ada)
Huruf besar / kecil	`s.toUpperCase()`, `s.toLowerCase()`	`s.upper()`, `s.lower()`
Buang spasi di tepi	`s.trim()` atau `s.strip()`	`s.strip()`
Ganti	`s.replace(a, b)`	`s.replace(a, b)`
Pecah menjadi kata	`s.split(" ")`	`s.split(" ")` atau `s.split()`
Ulangi	`s.repeat(3)`	`s * 3`
Isi sama	`s.equals(t)`	`s == t`

3.5.3 String tidak pernah berubah

Inilah fakta tentang string yang mengejutkan hampir semua orang: **string, sekali dibuat, tidak pernah dapat diubah**, baik di Java maupun di Python. String bersifat **imutabel**. Setiap metode yang tampaknya mengubah string, `toUpperCase`, `trim`, `replace`, sebenarnya membangun dan mengembalikan string *baru* dan membiarkan string asal persis seperti semula. **Kode 3.17** memperlihatkannya.

Kode 3.17 `StringImmutability.java`: metode string mengembalikan string baru; `==` membandingkan panah, sedangkan `equals` membandingkan karakter.

```java
public class StringImmutability {

    public static void main(String[] args) {
        String s = "laundry";
        String t = s.toUpperCase();   // membangun string BARU; s tidak berubah
        System.out.println("s = " + s + ", t = " + t);

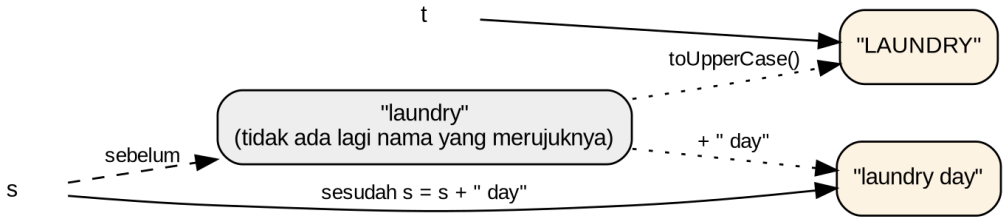
        s = s + " day";               // s kini menunjuk ke string baru yang lain
        System.out.println("s = " + s);

        String typed = new String("wash");  // objek terpisah dengan teks yang sama
        String code = "wash";
        System.out.println("code == typed:      " + (code == typed));
        System.out.println("code.equals(typed): " + code.equals(typed));

        StringBuilder receipt = new StringBuilder();
        for (int i = 1; i <= 3; i++) {
            receipt.append("item ").append(i).append("; ");
        }
        System.out.println(receipt.toString());
    }
}
```

```
s = laundry, t = LAUNDRY
s = laundry day
code == typed:      false
code.equals(typed): true
item 1; item 2; item 3;
```

Sesudah `String t = s.toUpperCase();`, variabel `s` masih merujuk ke `"laundry"` dalam huruf kecil: `toUpperCase` membangun string baru, `"LAUNDRY"`, dan `t` merujuk ke string itu. Lalu `s = s + " day";` tampak seperti mengubah `s`, padahal tidak ada string yang berubah. Pernyataan itu membangun string ketiga, `"laundry day"`, dan memindahkan panah `s` agar menunjuk kepadanya. `"laundry"` yang lama masih ada, tidak berubah, tanpa satu nama pun yang menunjuknya, dan pada waktunya pengumpul sampah menyingkirkannya. **Gambar 3.12** menggambar ketiga string dan panah-panahnya.

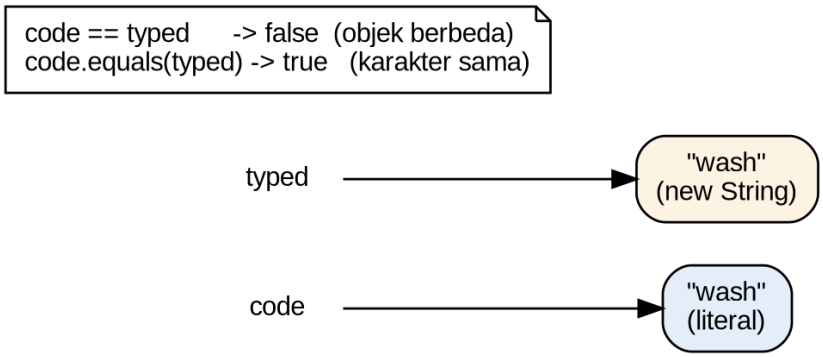


Gambar 3.12 Metode string tidak pernah mengubah string: `toUpperCase` dan `+` membangun string baru, dan penugasan hanya memindahkan panah.

Mengapa string dirancang demikian? Karena rancangan itu membuatnya aman untuk dibagi pakai: ketika `Customer` menyerahkan namanya melalui *getter*, tidak ada pemanggil yang dapat mengacak hurufnya. Bab 13 kembali membahas imutabilitas sebagai alat perancangan untuk kelas-kelas Anda sendiri.

Imutabilitas ada harganya: membangun teks panjang dengan `+` berulang-ulang di dalam perulangan menyalin semua karakter yang sudah ada pada setiap langkah. Untuk banyak potongan, Java menyediakan `StringBuilder` yang mutabel, sedangkan Python mengumpulkan potongan-potongan dalam sebuah *list* lalu menggabungkannya sekali dengan `"".join(parts)`; bagian akhir **Kode 3.17** dan **Kode 3.18** memperlihatkan keduanya.

Kini baris terpenting dalam **Kode 3.17**: `code == typed` bernilai `false`, padahal keduanya berisi `"wash"`. Sekarang Anda dapat menjelaskannya dengan **Tabel 3.3**: `==` pada referensi membandingkan panah, dan `typed` dibangun dengan `new String("wash")` sebagai objek terpisah. **Gambar 3.13** menggambarkannya. Metode `equals` membandingkan karakternya satu per satu dan menjawab `true`.



Gambar 3.13 Dua objek `String` berbeda dengan karakter yang sama: `==` menjawab `false`, `equals` menjawab `true`.

> **PERINGATAN:** Di Java, selalu bandingkan *isi* string dengan `equals`, jangan pernah dengan `==`. Jebakan ini kejam karena `==` kadang tampak berhasil: Java menyimpan satu salinan bersama untuk setiap literal string yang tertulis di kode sumber, sehingga `"wash" == "wash"` kebetulan bernilai `true`. Namun, teks yang datang saat program berjalan, diketik pengguna, dibaca dari berkas, atau dibangun dengan `substring` atau `+`, adalah objek baru, dan `==` akan menjawab `false` meskipun karakternya persis sama.

Python terhindar dari jebakan ini, karena `==` di Python membandingkan isi. **Kode 3.18** menceritakan hal yang sama; untuk membuat string yang pasti merupakan objek terpisah, program itu membangun `"wash"` dari huruf-hurufnya saat berjalan.

Kode 3.18 `string_immutability.py`: string Python juga imutabel; `==` membandingkan isi dan `is` membandingkan identitas.

```
def main():
    s = "laundry"
    t = s.upper()  # membangun string BARU; s tidak berubah
    print("s = " + s + ", t = " + t)

    s = s + " day"  # s kini menamai string baru yang lain
    print("s = " + s)

    typed = "".join(["w", "a", "s", "h"])  # dibangun saat program berjalan: objek terpisah
    code = "wash"
    print("code is typed: " + str(code is typed))
    print("code == typed: " + str(code == typed))

    parts = []
    for i in range(1, 4):
        parts.append("item " + str(i) + "; ")
    receipt = "".join(parts)
    print(receipt)


if __name__ == "__main__":
    main()
```

```
s = laundry, t = LAUNDRY
s = laundry day
code is typed: False
code == typed: True
item 1; item 2; item 3;
```

3.5.4 Memformat keluaran

Sejauh ini kita membangun keluaran dengan merekatkan string memakai `+`. Cara itu berhasil, tetapi tidak memberi kendali atas rupa bilangan. Berat `3.456` kg sebaiknya tampil di nota sebagai `3.46`; harga `24192` rupiah lebih mudah dibaca sebagai `24,192`; sekolom harga sebaiknya rata. Untuk itu kedua bahasa menyediakan **string format**: sebuah templat berisi tempat penampung (*placeholder*), beserta daftar nilai untuk mengisinya. **Kode 3.19** dan **Kode 3.20** memperlihatkan penampung yang paling berguna.

Kode 3.19 `StringFormatting.java`: `printf` dan `String.format` dengan lebar, desimal, dan pemisah ribuan.

```java
public class StringFormatting {

    public static void main(String[] args) {
        String name = "Siti";
        double weightKg = 3.456;
        long priceRupiah = 24192;

        System.out.println("Customer " + name + " brought " + weightKg + " kg");
        System.out.printf("Customer %s brought %.2f kg%n", name, weightKg);
        System.out.printf("Total: Rp%,d%n", priceRupiah);
        System.out.printf("|%-10s|%8.1f|%n", "Wash", 7000.0);
        System.out.printf("|%-10s|%8.1f|%n", "Iron", 5000.0);

        String line = String.format("%s owes Rp%,d", name, priceRupiah);
        System.out.println(line + " (" + line.length() + " characters)");
        System.out.println("Percent sign: " + String.format("%.1f%%", 2.3));
    }
}
```

```
Customer Siti brought 3.456 kg
Customer Siti brought 3.46 kg
Total: Rp24,192
|Wash      |  7000.0|
|Iron      |  5000.0|
Siti owes Rp24,192 (18 characters)
Percent sign: 2.3%
```

Kode 3.20 `string_formatting.py`: pemformatan yang sama dengan f-string dan `str.format` milik Python.

```
def main():
    name = "Siti"
    weight_kg = 3.456
    price_rupiah = 24192

    print("Customer " + name + " brought " + str(weight_kg) + " kg")
    print(f"Customer {name} brought {weight_kg:.2f} kg")
    print(f"Total: Rp{price_rupiah:,d}")
    print(f"|{'Wash':<10}|{7000.0:8.1f}|")
    print(f"|{'Iron':<10}|{5000.0:8.1f}|")

    line = "{} owes Rp{:,d}".format(name, price_rupiah)
    print(line + " (" + str(len(line)) + " characters)")
    print("Percent sign: " + f"{2.3:.1f}%")


if __name__ == "__main__":
    main()
```

```
Customer Siti brought 3.456 kg
Customer Siti brought 3.46 kg
Total: Rp24,192
|Wash      |  7000.0|
|Iron      |  5000.0|
Siti owes Rp24,192 (18 characters)
Percent sign: 2.3%
```

Di Java, `System.out.printf(template, values...)` mencetak, sedangkan `String.format(template, values...)` mengembalikan teksnya. Setiap penampung diawali `%`. Di Python, sebuah **f-string**, yaitu literal string yang diawali huruf `f`, memuat nilai langsung di antara kurung kurawal, diikuti titik dua dan spesifikasi format: `f"{weight_kg:.2f}"`. **Tabel 3.5** mendaftar penampung yang kita pakai dalam buku ini.

Tabel 3.5 Penampung format yang dipakai dalam buku ini.

Makna	Penampung Java	f-string Python	Contoh keluaran
Teks	`%s`	`{name}`	`Siti`
Bilangan bulat	`%d`	`{n:d}` atau `{n}`	`24192`
Dengan pemisah ribuan	`%,d`	`{n:,d}`	`24,192`
Desimal, 2 angka	`%.2f`	`{x:.2f}`	`3.46`
Rata kanan dalam 8 kolom	`%8.1f`	`{x:8.1f}`	`7000.0`
Rata kiri dalam 10 kolom	`%-10s`	`{s:<10}`	`Wash`
Baris baru	`%n`	(ditambahkan `print`)	
Tanda persen harfiah	`%%`	`%`	`2.3%`

Dua hal patut diwaspadai. Pertama, `%,d` di Java memakai pemisah ribuan sesuai **lokal** (*locale*, pengaturan kewilayahan) komputer Anda: pada komputer berpengaturan Indonesia, program yang sama mencetak `24.192` dengan titik, sebagaimana diharapkan pembaca Indonesia. Keluaran dalam buku ini berasal dari komputer berpengaturan Inggris, dan `{:,d}` di Python selalu memakai koma. Kedua, di dalam string format Java tanda persen bersifat khusus, sehingga `%` harfiah ditulis `%%`. Program Investment di Subbab 3.11.3 tidak pernah menjumpai persoalan ini, karena pesannya dibangun dengan konkatenasi, dan di sana `%` hanyalah karakter biasa.

3.5.5 Dari teks ke bilangan, dan kembali

Semua yang diketik pengguna datang sebagai teks. Kata `"4"` di baris perintah adalah string, bukan bilangan 4, dan Anda tidak dapat berhitung dengannya. Sebelum berhitung, Anda harus **mengurai** (*parse*) teks itu: membaca karakternya dan membangun bilangan yang digambarkannya. Java menyediakan `Integer.parseInt` untuk bilangan bulat dan `Double.parseDouble` untuk desimal; Python memakai `int(...)` dan `float(...)`. Ke arah sebaliknya, `String.valueOf(n)` di Java dan `str(n)` di Python mengubah bilangan menjadi teks. **Kode 3.21** dan **Kode 3.22** memperlihatkan kedua arah.

Kode 3.21 `ParseNumbers.java`: mengubah teks baris perintah menjadi `int` dan `double`.

```java
public class ParseNumbers {

    public static void main(String[] args) {
        String kgText = args.length > 0 ? args[0] : "4";
        String rateText = args.length > 1 ? args[1] : "7500.5";

        int kg = Integer.parseInt(kgText);           // teks -> int
        double rate = Double.parseDouble(rateText);  // teks -> double
        System.out.println(kg + " kg at " + rate + " per kg = " + (kg * rate));

        String back = String.valueOf(kg) + "!";      // int -> teks
        System.out.println(back);
        System.out.println("Text \"12\" + \"3\" = " + ("12" + "3"));
        System.out.println("Number 12 + 3 = " + (Integer.parseInt("12") + 3));
    }
}
```

```
4 kg at 7500.5 per kg = 30002.0
4!
Text "12" + "3" = 123
Number 12 + 3 = 15
```

Dua baris pertama `main` memakai bentuk ringkas `if` yang disebut **operator kondisional**: `kondisi ? a : b` bernilai `a` bila kondisinya benar dan `b` bila salah. Jadi `args.length > 0 ? args[0] : "4"` berarti "kata pertama di baris perintah bila ada, dan teks `"4"` bila tidak". Python menuliskan gagasan yang sama dengan kata-kata, `a if kondisi else b`, seperti diperlihatkan **Kode 3.22**. `Customer` di Subbab 3.12 memakainya sekali lagi untuk memilih antara kata *member* dan *guest*.

Dengan dua kata di baris perintah, `java ParseNumbers 6 7000`, program memakai keduanya sebagai ganti nilai bawaan:

```
6 kg at 7000.0 per kg = 42000.0
6!
Text "12" + "3" = 123
Number 12 + 3 = 15
```

Kode 3.22 `parse_numbers.py`: `int(...)` dan `float(...)` mengurai teks di Python.

```python
import sys


def main():
    kg_text = sys.argv[1] if len(sys.argv) > 1 else "4"
    rate_text = sys.argv[2] if len(sys.argv) > 2 else "7500.5"

    kg = int(kg_text)          # teks -> int
    rate = float(rate_text)    # teks -> float
    print(str(kg) + " kg at " + str(rate) + " per kg = " + str(kg * rate))

    back = str(kg) + "!"       # int -> teks
    print(back)
    print('Text "12" + "3" = ' + ("12" + "3"))
    print("Number 12 + 3 = " + str(int("12") + 3))


if __name__ == "__main__":
    main()
```

```
4 kg at 7500.5 per kg = 30002.0
4!
Text "12" + "3" = 123
Number 12 + 3 = 15
```

Dua baris terakhir setiap keluaran menegaskan pokok persoalannya. `"12" + "3"` menyambung dua string dan menghasilkan `"123"`; `Integer.parseInt("12") + 3` menjumlahkan dua bilangan dan menghasilkan `15`. Lambang sama, makna berbeda, ditentukan oleh tipe di kedua sisinya.

Bagaimana jika teksnya bukan bilangan? Penguraian gagal dengan lantang. Menjalankan `java ParseNumbers 3 abc` berhenti dengan

```
Exception in thread "main" java.lang.NumberFormatException: For input string: "abc"
    at
java.base/jdk.internal.math.FloatingDecimal.readJavaFormatString(FloatingDecimal.java:2054)
    at java.base/jdk.internal.math.FloatingDecimal.parseDouble(FloatingDecimal.java:110)
    at java.base/java.lang.Double.parseDouble(Double.java:792)
    at ParseNumbers.main(ParseNumbers.java:8)
```

dan versi Python mengakhiri laporan galatnya dengan

```
ValueError: could not convert string to float: 'abc'
```

Bacalah pesan Java dari bawah ke atas: galat terjadi di `main`, pada baris 8 berkas `ParseNumbers.java`, di dalam pemanggilan `Double.parseDouble`, karena string masukannya `"abc"`. Bab 5 memperlihatkan cara menangkap galat semacam itu dan meminta pengguna mengulang, alih-alih berhenti.

3.6 Tipe

3.6.1 Delapan tipe primitif Java

Setiap nilai dalam program Java memiliki **tipe**, dan tipe itu menentukan dua hal: nilai apa saja yang diperbolehkan, dan operasi apa saja yang masuk akal. Anda tidak dapat mengalikan dua ekor anjing, dan Anda tidak dapat memanggil `toUpperCase` pada bilangan 7.

Kita sudah bertemu tipe referensi, yang jumlahnya sebanyak kelas yang ada. Tipe primitif adalah keluarga kecil yang tetap, beranggotakan delapan tipe, yang tercantum dalam **Tabel 3.6** beserta memori yang diperlukan masing-masing dan rentang nilai yang dapat disimpannya [6].

Tabel 3.6 Delapan tipe primitif Java, dan tipe Python yang memainkan peran masing-masing.

Tipe	Menyimpan	Ukuran	Rentang	Padanan Python
`byte`	bilangan bulat	8 bit	−128 sampai 127	`int`
`short`	bilangan bulat	16 bit	−32 768 sampai 32 767	`int`
`int`	bilangan bulat	32 bit	−2 147 483 648 sampai 2 147 483 647	`int`
`long`	bilangan bulat	64 bit	sekitar $\pm 9.22 \times 10^{18}$	`int`
`float`	bilangan desimal	32 bit	sekitar 7 digit signifikan	(tidak ada)
`double`	bilangan desimal	64 bit	sekitar 15 sampai 16 digit signifikan	`float`
`char`	satu karakter	16 bit	`'\u0000'` sampai `''`	`str` sepanjang 1
`boolean`	nilai kebenaran	(tidak ditetapkan)	`true` atau `false`	`bool`

Ada dua jenis bilangan: **bilangan bulat** (*integer*) dan **bilangan desimal**.

Untuk bilangan bulat ada empat tipe, `byte`, `short`, `int`, dan `long`, yang hanya berbeda dalam banyaknya bit yang dipakai, sehingga berbeda pula besarnya bilangan yang dapat ditampung. Tipe dengan b bit dapat mewakili 2^b pola yang berbeda; Java memakai separuhnya untuk bilangan negatif dan separuh lainnya untuk nol dan bilangan positif. Jadi rentang tipe bilangan bulat b-bit adalah

$$-2^{b-1} \le v \le 2^{b-1} - 1 \tag{3.1}$$

yang, menurut Persamaan (3.1), memberikan $-2^7 = -128$ sampai $2^7 - 1 = 127$ untuk `byte` 8-bit, dan -2^{31} sampai $2^{31} - 1$, sedikit di atas dua miliar ke kedua arah, untuk `int` 32-bit. Tipe yang lebih kecil menghemat memori, yang penting ketika Anda menyimpan jutaan bilangan pada perangkat kecil, tetapi sering kali perbedaannya tidak terasa. Sebagian besar metode Java mengembalikan `int` atau `long`, sehingga kedua tipe itulah yang akan Anda pakai. Program dalam **Kode 3.27** mencetak rentang-rentang itu langsung dari konstanta milik Java sendiri.

Untuk bilangan desimal ada `float`, 32 bit dengan sekitar 7 digit signifikan, dan `double`, 64 bit dengan sekitar dua kali lipatnya (karena itulah disebut "presisi ganda"). Sebagian besar metode Java, seperti `Math.sqrt` dan `Math.pow`, mengembalikan `double`, sehingga pilihlah `double` kecuali Anda punya alasan kuat.

Dua tipe sisanya adalah `char`, satu karakter, dan `boolean`, yang menyimpan `true` atau `false` dan tidak ada yang lain. `boolean` adalah hasil setiap kondisi dalam `if` atau perulangan.

Python jauh lebih sederhana. Python hanya memiliki satu tipe bilangan bulat, `int`, **tanpa batas atas**: ia tumbuh sampai sebanyak digit yang diperlukan. Python memiliki satu tipe desimal, `float`, yang persis sama dengan `double` di Java. Python memiliki `bool` untuk nilai kebenaran, dan sama sekali tidak memiliki tipe karakter.

3.6.2 Pengetikan statis dan pengetikan dinamis

Perbedaan terbesar antara Java dan Python bukanlah sebuah kata kunci atau tanda kurung, melainkan *kapan* tipe diperiksa.

Java **bertipe statis** (*statically typed*). Setiap variabel dideklarasikan dengan sebuah tipe, dan sepanjang hidupnya variabel itu hanya boleh menyimpan nilai bertipe tersebut. Kompilator memeriksa setiap penugasan dan setiap pemanggilan metode *sebelum program berjalan*. **Kode 3.23** memperlihatkan aturan-aturan itu bekerja.

Kode 3.23 `StaticTyping.java`: di Java setiap variabel memiliki satu tipe seumur hidupnya; `var` membiarkan kompilator menyimpulkan tipenya.

```java
public class StaticTyping {

    public static void main(String[] args) {
        int count = 3;              // count hanya boleh menyimpan nilai int
        double price = 7000;        // int 7000 dilebarkan menjadi 7000.0
        var service = "Wash";       // var: kompilator menyimpulkan String
        long total = count * 7000L;

        System.out.println(count + " " + price + " " + service + " " + total);
        System.out.println(service.getClass().getSimpleName());
        // count = "three";  // tidak akan terkompilasi: String tidak bisa menjadi int
    }
}
```

```
3 7000.0 Wash 21000
String
```

Beberapa rincian dalam **Kode 3.23** patut diperhatikan. `double price = 7000;` diperbolehkan: `int` dapat **dilebarkan** (*widened*) menjadi `double` tanpa kehilangan apa pun. Kebalikannya, `int n = 7000.5;`, ditolak, karena pecahannya akan hilang. `var service = "Wash";` tidak membuat `service` tak bertipe: `var` sekadar meminta kompilator *menyimpulkan* tipe dari ruas kanan, dan sejak itu `service` selamanya bertipe `String`. Adapun `7000L`, dengan huruf `L`, adalah literal `long`, sehingga `count * 7000L` merupakan perkalian `long`.

Sekarang hapus `//` di depan baris `count = "three";` lalu kompilasi ulang. Kompilator menolak, dan program tidak pernah berjalan:

```
StaticTyping.java:11: error: incompatible types: String cannot be converted to int
        count = "three";
                ^
1 error
```

Python **bertipe dinamis** (*dynamically typed*). Variabel tidak memiliki tipe yang dideklarasikan; sebuah nama dapat diikat ke nilai bertipe apa pun, dan kemudian diikat ulang ke nilai bertipe lain. *Nilai*-nya tetap memiliki tipe (setiap objek tahu apakah ia `int` atau `str`), dan tipe itu tetap diperiksa, tetapi baru pada saat sebuah operasi dijalankan. **Kode 3.24** memperlihatkan akibat-akibatnya.

Kode 3.24 `dynamic_typing.py`: di Python sebuah nama dapat merujuk ke nilai-nilai bertipe berbeda; tipe diperiksa ketika operasi dijalankan.

```
def main():
    count = 3
    print(count, type(count).__name__)
    count = "three"          # boleh: NAMA count kini merujuk ke sebuah str
    print(count, type(count).__name__)

    price: float = 7000      # petunjuk tipe mendokumentasikan maksud; Python tidak
memaksakannya
    print(price, type(price).__name__)
    total = 3 * 7000
    print(total, type(total).__name__, isinstance(total, int))

    try:
        print("Total: " + total)
    except TypeError as error:
        print("TypeError:", error)


if __name__ == "__main__":
    main()
```

```
3 int
three str
7000 int
21000 int True
TypeError: can only concatenate str (not "int") to str
```

Nama `count` mula-mula merujuk ke sebuah `int`, lalu ke sebuah `str`, dan Python tidak berkeberatan. Anotasi `price: float = 7000` adalah **petunjuk tipe** (*type hint*): editor dan pemeriksa tipe membacanya, tetapi Python sendiri tidak memaksakannya, itulah sebabnya `type(price)` tetap `int`. Terakhir, `"Total: " + total` gagal dengan `TypeError` *saat program berjalan*: Python tidak mau menebak apakah Anda bermaksud menjumlahkan bilangan atau menyambung teks, dan itulah sebabnya `str(...)` diperlukan di Subbab 3.3.5. Python bertipe *dinamis* tetapi *kuat* (*strongly typed*): tipe diperiksa belakangan, tetapi tetap diperiksa.

Tidak ada gaya yang begitu saja lebih baik: pengetikan statis menangkap kesalahan sebelum program berjalan dan membantu dalam program besar; pengetikan dinamis membuat program kecil lebih singkat ditulis.

3.6.3 Karakter dan nilai kebenaran

Kode 3.25 mengamati dua tipe primitif yang tersisa.

Kode 3.25 `CharBoolean.java`: di balik layar, `char` adalah bilangan kecil; `&&`, `||`, dan `!` menggabungkan nilai kebenaran.

```java
public class CharBoolean {

    public static void main(String[] args) {
        char grade = 'B';
        char next = (char) (grade + 1);      // di balik layar, karakter adalah bilangan
kecil
        System.out.println(grade + " is code " + (int) grade + "; next is " + next);

        boolean isMember = true;
        boolean isExpress = false;
        double weightKg = 2.5;
        boolean discount = isMember && weightKg >= 2.0;
        boolean surcharge = isExpress || weightKg > 10.0;
        System.out.println("discount = " + discount + ", surcharge = " + surcharge);
        System.out.println("!isMember = " + !isMember);

        String note = null;
        // && berhenti lebih awal: note.length() tidak pernah dipanggil bila note bernilai
null
        if (note != null && note.length() > 0) {
            System.out.println("Note: " + note);
        } else {
            System.out.println("No note on this order");
        }
    }
}
```

```
B is code 66; next is C
discount = true, surcharge = false
!isMember = false
No note on this order
```

Sebuah `char` disimpan sebagai bilangan, yaitu **kode karakter**-nya dalam tabel Unicode: `'B'` adalah 66. Menambahkan 1 menghasilkan 67, dan mengubahnya kembali dengan `(char)` menghasilkan `'C'`. Dengan cara inilah program melangkah menyusuri abjad.

Nilai kebenaran digabungkan dengan tiga **operator logika**. `a && b` (**dan**) bernilai `true` hanya bila keduanya benar. `a || b` (**atau**) bernilai `true` bila setidaknya salah satunya benar. `!a` (**tidak**) mengubah `true` menjadi `false` dan sebaliknya. Dalam **Kode 3.25**, seorang anggota dengan cucian 2,5 kg berhak atas potongan harga (`true && true`), dan pesanan itu tidak dikenai biaya tambahan (`false || false`).

Operator `&&` dan `||` adalah operator **hubung-singkat** (*short-circuit*): keduanya berhenti begitu jawabannya sudah pasti. Jika sisi kiri `&&` bernilai `false`, seluruh ungkapan pasti `false`, sehingga sisi kanan tidak pernah dievaluasi. Pengujian `note != null && note.length() > 0` bergantung pada sifat ini: ketika `note` bernilai `null`, Java tidak pernah memanggil `note.length()`, sehingga tidak pernah berjumpa `NullPointerException`.

Kode 3.26 `char_boolean.py`: `ord` dan `chr` mengonversi antara karakter dan kodenya; `and`, `or`, dan `not` menggabungkan nilai kebenaran.

```
def main():
    grade = "B"  # Python tidak punya tipe char: cukup string sepanjang satu karakter
    next_grade = chr(ord(grade) + 1)
    print(grade + " is code " + str(ord(grade)) + "; next is " + next_grade)

    is_member = True
    is_express = False
    weight_kg = 2.5
    discount = is_member and weight_kg >= 2.0
    surcharge = is_express or weight_kg > 10.0
    print("discount = " + str(discount) + ", surcharge = " + str(surcharge))
    print("not is_member = " + str(not is_member))

    note = None
    # "and" berhenti lebih awal: len(note) tidak pernah dipanggil bila note bernilai None
    if note is not None and len(note) > 0:
        print("Note: " + note)
    else:
        print("No note on this order")

    print("bool(0) = " + str(bool(0)) + ", bool('') = " + str(bool(""))
          + ", bool('x') = " + str(bool("x")))


if __name__ == "__main__":
    main()
```

```
B is code 66; next is C
discount = True, surcharge = False
not is_member = False
No note on this order
bool(0) = False, bool('') = False, bool('x') = True
```

Kode 3.26 memperlihatkan bahwa Python menuliskan operator-operator itu sebagai kata, `and`, `or`, `not`, dan semuanya juga bersifat hubung-singkat. Fungsi `ord` di Python memberikan kode sebuah string sepanjang satu, dan `chr` melakukan kebalikannya. Satu fitur Python tidak memiliki padanan di Java: **truthiness** (nilai kebenaran tersirat). Di Python nilai apa pun dapat berdiri sebagai kondisi: `0`, string kosong `""`, `None`, dan koleksi kosong dianggap salah, selebihnya dianggap benar. Java hanya menerima nilai `boolean` sungguhan dalam kondisi, dan `if (count)` tidak dapat dikompilasi.

3.7 Aritmetika dan Jebakannya

3.7.1 Operator dan prioritas

Aritmetika dalam program bekerja seperti pada kalkulator, dengan beberapa tambahan. Misalkan

```
int i = 0;
int j = 3;
int result = 2 + i * j;
```

Java mengalikan `i` dengan `j` lebih dulu, lalu menambahkan 2, persis seperti dalam matematika, sehingga `result` bernilai 2. Perkalian dan pembagian mengikat lebih erat daripada penjumlahan dan pengurangan; urutan ini disebut **prioritas** (*precedence*). Tanda kurung mengesampingkannya: `(2 + i) * j` bernilai 6. **Tabel 3.7** mendaftar operator-operator itu, dari yang paling erat sampai yang paling longgar.

Tabel 3.7 Operator yang umum, dari prioritas tertinggi sampai terendah. Di Python, `**` mengikat lebih erat daripada minus uner, sehingga `-2 ** 2` bernilai `-4`.

Operator	Makna	Contoh Java	Contoh Python
`( )`	pengelompokan	`(2 + i) * j`	`(2 + i) * j`
`-x`, `!x` / `not x`	negasi	`-x`, `!ok`	`-x`, `not ok`
`*`, `/`, `%`	kali, bagi, sisa bagi	`7 / 2` adalah `3`	`7 / 2` adalah `3.5`, `7 // 2` adalah `3`
`**`	pangkat	(pakai `Math.pow`)	`2 ** 3` adalah `8`
`+`, `-`	tambah, kurang (dan sambung string)	`"a" + 1` adalah `"a1"`	`"a" + str(1)`
`<`, `<=`, `>`, `>=`	perbandingan	`i > 0`	`i > 0`
`==`, `!=`	sama dengan, tidak sama dengan	`i == 0`	`i == 0`
`&&` / `and`	dan logika	`a && b`	`a and b`
`||` / `or`	atau logika	`a || b`	`a or b`
`=`, `+=`, `-=`, …	penugasan	`x += 3`	`x += 3`

Java tidak memiliki operator pangkat. Untuk 2^3 Anda memanggil `Math.pow(2, 3)`, yang menerima bilangan pokok dan pangkatnya lalu mengembalikan `double` `8.0`. Python menulis `2 ** 3` dan, karena kedua operannya bilangan bulat, menghasilkan bilangan bulat 8.

Java memiliki dua operator mungil yang akan Anda jumpai di setiap perulangan: `i++` menambahkan satu pada `i`, dan `i--` mengurangkan satu. Python tidak memilikinya; Python menulis `i += 1` dan `i -= 1`. Kedua bahasa memiliki penugasan majemuk `+=`, `-=`, `*=`, `/=`: `x += 3` berarti `x = x + 3`.

Aritmetika komputer *berbeda* dari aritmetika sekolah di tiga tempat, dan masing-masing pernah menimbulkan kerugian uang sungguhan dalam perangkat lunak sungguhan.

3.7.2 Jebakan pertama: luapan bilangan bulat

Batas-batas dalam **Tabel 3.6** berlaku pada *setiap langkah* perhitungan, bukan hanya pada jawaban akhirnya. Perhatikan:

```
int x = 100000;   // 100,000
int y = 200000;   // 200,000
int z = 5000;     // 5,000
System.out.println(x * y / z);
```

Dengan tangan: $100\,000 \times 200\,000 = 20\,000\,000\,000$, dan dibagi 5 000 hasilnya 4 000 000. Tidak satu pun dari ketiga variabel itu mendekati batas `int`, begitu pula jawabannya. Namun, **Kode 3.27** mencetak sesuatu yang sama sekali lain.

Kode 3.27 `IntegerTypes.java`: rentang tipe-tipe bilangan bulat, dan luapan di tengah perhitungan.

```java
public class IntegerTypes {

    public static void main(String[] args) {
        System.out.println("byte  : " + Byte.MIN_VALUE + " .. " + Byte.MAX_VALUE);
        System.out.println("short : " + Short.MIN_VALUE + " .. " + Short.MAX_VALUE);
        System.out.println("int   : " + Integer.MIN_VALUE + " .. " + Integer.MAX_VALUE);
        System.out.println("long  : " + Long.MIN_VALUE + " .. " + Long.MAX_VALUE);

        int x = 100000;   // 100,000
        int y = 200000;   // 200,000
        int z = 5000;     // 5,000
        System.out.println("x * y / z as int : " + (x * y / z));
        System.out.println("x * y as int     : " + (x * y));
        System.out.println("x * y / z as long: " + ((long) x * y / z));

        int biggest = Integer.MAX_VALUE;
        System.out.println("MAX_VALUE + 1    : " + (biggest + 1));
    }
}
```

```
byte  : -128 .. 127
short : -32768 .. 32767
int   : -2147483648 .. 2147483647
long  : -9223372036854775808 .. 9223372036854775807
x * y / z as int : -294967
x * y as int     : -1474836480
x * y / z as long: 4000000
MAX_VALUE + 1    : -2147483648
```

Jawabannya $-294\,967$: negatif, dan tidak masuk akal. Biang keladinya adalah hasil kali antara $x \cdot y = 2 \times 10^{10}$, yang jauh di atas batas `int` sebesar $2^{31} - 1 = 2\,147\,483\,647$. Java tidak berhenti dan tidak memberi peringatan. Java hanya menyimpan 32 bit terendah dari hasil yang sebenarnya, yang sama saja dengan mengambil nilai sebenarnya modulo 2^{32} lalu membaca hasilnya sebagai bilangan bertanda:

$$\text{tersimpan}(v) = ((v + 2^{31}) \bmod 2^{32}) - 2^{31} \tag{3.2}$$

Mari kita periksa Persamaan (3.2) dengan tangan untuk $v = 2 \times 10^{10}$. Pertama, $2^{32} = 4\,294\,967\,296$. Lalu $v + 2^{31} = 20\,000\,000\,000 + 2\,147\,483\,648 = 22\,147\,483\,648$. Pembagian dengan 2^{32} menghasilkan 5 kali utuh, karena $5 \times 4\,294\,967\,296 = 21\,474\,836\,480$, dengan sisa $22\,147\,483\,648 - 21\,474\,836\,480 = 672\,647\,168$. Terakhir, $672\,647\,168 - 2\,147\,483\,648 = -1\,474\,836\,480$. Itulah persis baris "x * y as int" pada keluaran. Membaginya dengan 5 000, sambil membuang pecahannya sebagaimana dilakukan pembagian bilangan bulat (Subbab 3.7.4), menghasilkan $-294\,967$. Misteri terpecahkan: hasil kali itu **meluap** (*overflow*).

Putaran balik yang sama menjelaskan baris terakhir: `Integer.MAX_VALUE + 1` tidak menghasilkan $2\,147\,483\,648$, tetapi melompat ke `int` yang paling negatif, $-2\,147\,483\,648$, seperti odometer mobil yang berputar dari 999999 ke 000000.

Obatnya adalah melakukan aritmetika yang berisiko dalam tipe yang lebih lebar. Menulis `(long) x * y / z` mengubah `x` menjadi `long` *sebelum* perkalian, sehingga hasil kalinya dihitung dalam 64 bit, tempat 2×10^{10} muat dengan leluasa, dan jawabannya menjadi $4\,000\,000$ yang benar. Perhatikan letaknya: `(long) (x * y)` sudah terlambat, karena kerusakan terjadi di dalam kurung, dalam aritmetika `int`, sebelum konversi.

Python terhindar dari masalah ini karena `int`-nya tidak berbatas, seperti diperlihatkan **Kode 3.28**. Untuk melihat apa yang akan dihitung Java, program itu meniru `int` 32-bit dengan sebuah fungsi kecil yang menerapkan Persamaan (3.2) memakai operasi bit.

Kode 3.28 `integer_types.py`: bilangan bulat Python tidak pernah meluap; `wrap_int32` meniru `int` 32-bit milik Java.

```
def wrap_int32(value):
    """Memampatkan int Python ke 32 bit seperti yang dilakukan int Java."""
    value = value & 0xFFFFFFFF          # simpan hanya 32 bit terendah
    if value >= 2**31:                  # bit teratas berarti "negatif"
        value -= 2**32
    return value


def main():
    x = 100000   # 100,000
    y = 200000   # 200,000
    z = 5000     # 5,000
    print("x * y // z          : " + str(x * y // z))
    print("x * y               : " + str(x * y))
    print("x * y wrapped to 32 : " + str(wrap_int32(x * y)))
    print("wrapped, then // z  : " + str(int(wrap_int32(x * y) / z)))

    biggest = 2**31 - 1
    print("2**31 - 1 + 1       : " + str(biggest + 1))
    print("wrapped             : " + str(wrap_int32(biggest + 1)))
    print("2**100              : " + str(2**100))


if __name__ == "__main__":
    main()
```

```
x * y // z          : 4000000
x * y               : 20000000000
x * y wrapped to 32 : -1474836480
wrapped, then // z  : -294967
2**31 - 1 + 1       : 2147483648
wrapped             : -2147483648
2**100              : 1267650600228229401496703205376
```

> **PERINGATAN:** Luapan terjadi diam-diam. Setiap kali perhitungan mengalikan bilangan bulat yang besar, misalnya uang dalam rupiah (harga beberapa juta dikali jumlah beberapa ribu saja sudah melampaui dua miliar), lakukanlah dalam `long`. Di Java, `Math.multiplyExact(a, b)` adalah pilihan yang lebih aman: ia menghentikan program dengan galat alih-alih mengembalikan jawaban yang salah.

3.7.3 Jebakan kedua: komputer tidak bisa menghitung sampai satu dalam persepuluhan

Komputer bekerja dengan bit dalam jumlah bulat, sehingga bilangan **titik mengambang** (*floating-point*) menyimpan desimal sebagai *signifikan* berupa bilangan bulat dan sebuah pangkat dua, mirip notasi ilmiah yang menyimpan 6.02×10^{23} [7]. Bilangan yang tampak polos seperti 0.1 tidak memiliki representasi tepat dalam basis dua, sebagaimana $\frac{1}{3}$ tidak memilikinya dalam basis sepuluh (0.3333 ... harus berhenti di suatu tempat). Komputer menyimpan bilangan terdekat yang dapat diwakilinya, dan galat-galat kecil pun menyusup masuk [8].

Kode 3.29 mencoba menjumlahkan 0.1 sepuluh kali. Jawabannya seharusnya 1.

Kode 3.29 `FloatingPoint.java`: menjumlahkan 0,1 sepuluh kali, dan mengapa bilangan desimal tidak boleh dibandingkan dengan `==`.

```
public class FloatingPoint {

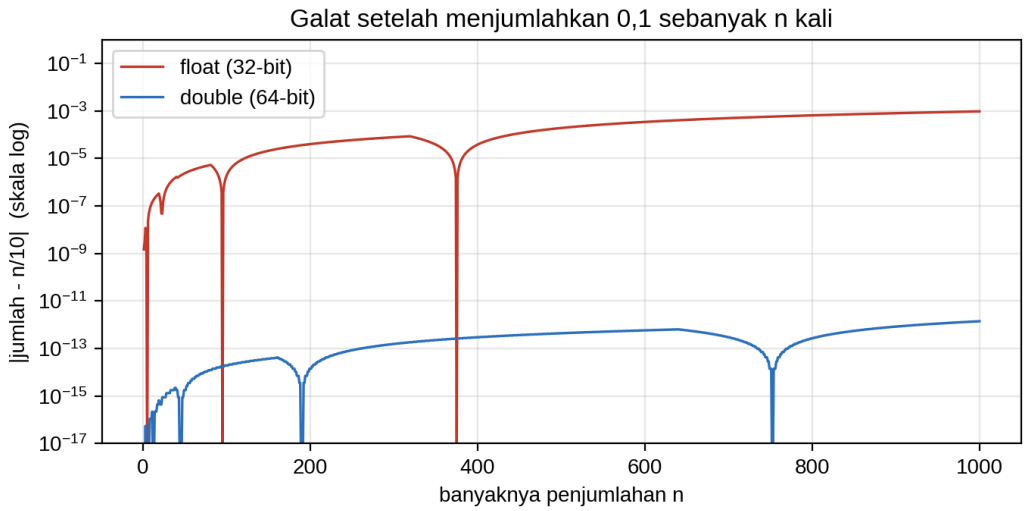
    public static void main(String[] args) {
        System.out.println(0.1f + 0.1f + 0.1f + 0.1f + 0.1f + 0.1f + 0.1f + 0.1f + 0.1f +
0.1f); // float
        System.out.println(0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1);
// double

        System.out.println("0.1 + 0.2 + 0.3 = " + (0.1 + 0.2 + 0.3));
        System.out.println("0.2 + 0.3 + 0.1 = " + (0.2 + 0.3 + 0.1));

        double a = 0.1 + 0.2;
        double b = 0.3;
        System.out.println("a == b            : " + (a == b));
        System.out.println("|a - b| < 1e-9    : " + (Math.abs(a - b) < 1e-9));
        System.out.println("2^3 with Math.pow : " + Math.pow(2, 3));
    }
}
```

```
1.0000001
0.9999999999999999
0.1 + 0.2 + 0.3 = 0.6000000000000001
0.2 + 0.3 + 0.1 = 0.6
a == b            : false
|a - b| < 1e-9    : true
2^3 with Math.pow : 8.0
```

Tidak satu pun baris mencetak `1.0`. Jumlah `float` melewati sasaran menjadi `1.0000001`; jumlah `double` sedikit kurang, `0.9999999999999999`. Galat `double` jauh lebih kecil, karena `double` memiliki lebih banyak bit, tetapi tidak nol. **Gambar 3.14** mengikuti galat itu seiring makin banyak persepuluhan yang dijumlahkan: galatnya tumbuh, naik-turun, dan untuk `float` kira-kira semiliar kali lebih besar daripada untuk `double`.



Gambar 3.14 Galat setelah 0,1 dijumlahkan berulang-ulang, untuk `float` dan `double`; lekukan sampai ke dasar grafik adalah saat-saat ketika galat pembulatan kebetulan saling meniadakan.

Dua pelajaran dapat dipetik. Pelajaran pertama: hukum-hukum aritmetika yang biasa hanya berlaku secara hampiran. Penjumlahan tidak lagi tepat asosiatif: `0.1 + 0.2 + 0.3` menghasilkan `0.6000000000000001`, sedangkan `0.2 + 0.3 + 0.1` menghasilkan `0.6`, karena pembulatannya terjadi pada langkah yang berbeda.

Pelajaran kedua adalah aturan yang paling penting dalam praktik: **jangan pernah membandingkan dua bilangan titik mengambang dengan `==`**. Sekalipun menurut matematika

keduanya sama, nilai hasil hitungannya bisa berbeda pada bit terakhir, dan `0.1 + 0.2 == 0.3` bernilai `false`. Bandingkanlah dengan toleransi: dua desimal a dan b dianggap sama bila

$$|a - b| < \varepsilon \tag{3.3}$$

untuk ε (huruf Yunani epsilon) kecil yang dipilih sesuai besarnya bilangan yang terlibat, misalnya 10^{-9}. Di Java, bentuknya `Math.abs(a - b) < 1e-9`, dengan `1e-9` sebagai cara program menulis 10^{-9}. Persamaan (3.3) persis yang dipakai `DogCheck` untuk berat di Subbab 3.3.6.

Kode 3.30 `floating_point.py`: efek yang sama di Python, dengan `math.isclose` dan modul `decimal`.

```
import math
from decimal import Decimal


def main():
    print(0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1)

    print("0.1 + 0.2 + 0.3 = " + str(0.1 + 0.2 + 0.3))
    print("0.2 + 0.3 + 0.1 = " + str(0.2 + 0.3 + 0.1))

    a = 0.1 + 0.2
    b = 0.3
    print("a == b             : " + str(a == b))
    print("abs(a - b) < 1e-9  : " + str(abs(a - b) < 1e-9))
    print("math.isclose(a, b) : " + str(math.isclose(a, b)))
    print("2 ** 3, math.pow   : " + str(2 ** 3) + ", " + str(math.pow(2, 3)))
    print("Decimal sum        : " + str(Decimal("0.1") * 10))


if __name__ == "__main__":
    main()
```

```
0.9999999999999999
0.1 + 0.2 + 0.3 = 0.6000000000000001
0.2 + 0.3 + 0.1 = 0.6
a == b             : False
abs(a - b) < 1e-9  : True
math.isclose(a, b) : True
2 ** 3, math.pow   : 8, 8.0
Decimal sum        : 1.0
```

Kode 3.30 memperlihatkan bahwa Python berperilaku persis seperti `double` di Java, karena `float` Python *memang* double 64-bit. Python juga menyediakan `math.isclose(a, b)`, versi siap pakai dari Persamaan (3.3), dan modul `decimal`, yang menghitung dalam basis sepuluh sehingga mewakili 0.1 secara tepat: `Decimal("0.1") * 10` tepat bernilai `1.0`.

> **WAWASAN KUNCI:** Uang tidak boleh disimpan sebagai `double`. Daftar harga yang totalnya meleset sepecahan rupiah cepat atau lambat akan gagal seimbang. Simpanlah uang sebagai bilangan bulat dalam satuan terkecilnya, yaitu rupiah dalam `long` (Java) atau `int` (Python), seperti yang dilakukan kelas-kelas Profitina Laundry di Subbab 3.12, atau pakailah `BigDecimal` di Java dan `Decimal` di Python.

3.7.4 Jebakan ketiga: pembagian bilangan bulat

Cepat atau lambat setiap pemula menulis `1 / 16` dan mendapat `0`. Bila kedua operan `/` adalah bilangan bulat, Java melakukan **pembagian bilangan bulat**: membagi lalu membuang pecahannya. Jadi `1 / 16` bernilai `0`, bukan `0.0625`; `17 / 5` bernilai `3`; `5 / 6` bernilai `0`.

Pasangannya adalah operator **sisa bagi** `%`: `17 % 5` bernilai 2, karena $17 = 3 \times 5 + 2$. Untuk sembarang bilangan bulat a dan $b \neq 0$, kedua operator itu terikat oleh

$$a = (a/b) \cdot b + (a \bmod b) \tag{3.4}$$

dengan a/b berarti hasil bagi bulat. Persamaan (3.4) adalah aturan "hasil bagi dan sisa" yang Anda pelajari di sekolah dasar. Sisa bagi jauh lebih berguna daripada kesan pertamanya: `n % 2 == 0` menguji apakah `n` genap, `n % 10` adalah digit desimal terakhir `n`, dan `year % 100 == 0` menyaring setiap tahun keseratus. Subbab 3.10 membangun sebuah algoritma utuh dari `% 10` dan `/ 10`.

Untuk mendapat jawaban pecahan, setidaknya satu operan harus desimal. Dengan konstanta, cukup tulis salah satunya sebagai desimal: `1 / 16.0` bernilai `0.0625`. Dengan variabel Anda tidak dapat menambahkan `.0`, sehingga Anda melakukan **konversi tipe** (*cast*): tuliskan tipe sasaran dalam kurung di depan nilai, dan Java memperlakukan nilai itu sebagai tipe tersebut untuk sementara. `(double) x / y` mengubah `x` menjadi `double`, lalu pembagiannya menjadi pembagian desimal. Seperti pada luapan, letaknya penting: `(double) (x / y)` membagi lebih dulu, secara bilangan bulat, lalu mengonversi hasil yang sudah terpotong, sehingga menghasilkan `0.0`. Mengonversi desimal menjadi bilangan bulat, `(int) 3.99`, sekadar memotong pecahannya dan menghasilkan `3`; untuk membulatkan ke bilangan bulat terdekat, pakailah `Math.round`. **Kode 3.31** dan **Kode 3.32** mencoba semua ini.

Kode 3.31 `IntegerDivision.java`: pembagian bilangan bulat, operator sisa bagi, dan konversi tipe.

```java
public class IntegerDivision {

    public static void main(String[] args) {
        System.out.println("1 / 16    = " + (1 / 16));
        System.out.println("1 / 16.0  = " + (1 / 16.0));

        int x = 5;
        int y = 6;
        System.out.println("x / y          = " + (x / y));
        System.out.println("(double) x / y = " + ((double) x / y));
        System.out.println("(double) (x / y) = " + ((double) (x / y)));

        System.out.println("17 / 5 = " + (17 / 5) + ", 17 % 5 = " + (17 % 5));
        System.out.println("-7 / 2 = " + (-7 / 2) + ", -7 % 2 = " + (-7 % 2));
        System.out.println("(int) 3.99 = " + (int) 3.99);
        System.out.println("Math.round(3.5) = " + Math.round(3.5));
        int result = 2 + 0 * 3;          // perkalian sebelum penjumlahan
        System.out.println("2 + 0 * 3 = " + result + ", (2 + 0) * 3 = " + ((2 + 0) * 3));
    }
}
```

```
1 / 16    = 0
1 / 16.0  = 0.0625
x / y          = 0
(double) x / y = 0.8333333333333334
(double) (x / y) = 0.0
17 / 5 = 3, 17 % 5 = 2
-7 / 2 = -3, -7 % 2 = -1
(int) 3.99 = 3
Math.round(3.5) = 4
2 + 0 * 3 = 2, (2 + 0) * 3 = 6
```

Kode 3.32 `integer_division.py`: Python memisahkan pembagian sejati `/` dari pembagian lantai `//`.

```
def main():
    print("1 // 16  = " + str(1 // 16))
    print("1 / 16   = " + str(1 / 16))

    x = 5
    y = 6
    print("x // y   = " + str(x // y))
    print("x / y    = " + str(x / y))
    print("float(x // y) = " + str(float(x // y)))

    print("17 // 5 = " + str(17 // 5) + ", 17 % 5 = " + str(17 % 5))
    print("-7 // 2 = " + str(-7 // 2) + ", -7 % 2 = " + str(-7 % 2))
    print("int(-7 / 2) = " + str(int(-7 / 2)))
    print("int(3.99) = " + str(int(3.99)))
    print("round(3.5) = " + str(round(3.5)) + ", round(2.5) = " + str(round(2.5)))
    result = 2 + 0 * 3  # perkalian sebelum penjumlahan
    print("2 + 0 * 3 = " + str(result) + ", (2 + 0) * 3 = " + str((2 + 0) * 3))


if __name__ == "__main__":
    main()
```

```
1 // 16  = 0
1 / 16   = 0.0625
x // y   = 0
x / y    = 0.8333333333333334
float(x // y) = 0.0
17 // 5 = 3, 17 % 5 = 2
-7 // 2 = -4, -7 % 2 = 1
int(-7 / 2) = -3
int(3.99) = 3
round(3.5) = 4, round(2.5) = 2
2 + 0 * 3 = 2, (2 + 0) * 3 = 6
```

Python menempuh jalan yang lebih bersih: `/` miliknya *selalu* menghasilkan desimal, `1 / 16` bernilai `0.0625`, dan operator tersendiri, `//`, melakukan pembagian bilangan bulat (pembagian lantai). Jebakannya hilang, tetapi muncul perbedaan yang harus diingat ketika menerjemahkan antarbahasa: **untuk bilangan negatif, Java dan Python tidak sepakat**. Java memotong *ke arah nol*, sehingga `-7 / 2` bernilai `-3` dan `-7 % 2` bernilai `-1`; Python membulatkan *ke bawah*, sehingga `-7 // 2` bernilai `-4` dan `-7 % 2` bernilai `1`. Keduanya mematuhi Persamaan (3.4): $(-3) \cdot 2 + (-1) = -7$ dan $(-4) \cdot 2 + 1 = -7$. Kejutan kedua menanti pada `round` di Python, yang membulatkan nilai tepat setengah ke tetangga yang *genap*: `round(2.5)` bernilai `2` dan `round(3.5)` bernilai `4`, sedangkan `Math.round` di Java selalu membulatkan setengah ke atas.

Tabel 3.8 Pembagian dan pembulatan dalam Java dan Python, berdampingan.

Ungkapan	Java	Python
`1 / 16`	`0`	`0.0625`
`1 // 16`	(tidak sah)	`0`
`17 / 5` dan `17 % 5`	`3` dan `2`	`3.4` dan `2`
`-7 / 2` (hasil bagi bulat)	`-3`	`-7 // 2` adalah `-4`
`-7 % 2`	`-1`	`1`
`(double) 5 / 6` / `5 / 6`	`0.8333333333333334`	`0.8333333333333334`
pembulatan 2,5	`Math.round(2.5)` adalah `3`	`round(2.5)` adalah `2`

Tabel 3.8 menghimpun perbedaan-perbedaan itu di satu tempat.

3.8 Percabangan

3.8.1 Melakukan sesuatu hanya bila sebuah kondisi terpenuhi

Sering kali program hanya boleh melakukan sesuatu bila kondisi tertentu benar: "*jika* pelanggan adalah anggota, *maka* beri potongan 10%, *jika tidak*, kenakan harga penuh". Pernyataan untuk keperluan ini adalah `if`. Bentuk umumnya

```
if (condition) {
    pernyataan yang dijalankan bila kondisinya benar
} else {
    pernyataan yang dijalankan bila kondisinya salah
}
```

Kondisinya harus berupa ungkapan `boolean`: perbandingan seperti `i > 0`, variabel `boolean`, metode yang mengembalikan `boolean` seperti `s.startsWith("the")`, atau gabungan semua itu dengan `&&`, `||`, dan `!`. Bagian `else` boleh tidak ada. Kurung kurawal mengelompokkan beberapa pernyataan menjadi satu **blok**; Java mengizinkan Anda menghilangkannya bila sebuah cabang hanya berisi satu pernyataan, tetapi selalu menuliskannya adalah kebiasaan yang baik, sebab pernyataan kedua yang ditambahkan belakangan tanpa kurung kurawal diam-diam jatuh di luar `if`.

Kode 3.33 dibuka dengan sebuah teka-teki kecil. Sebelum melihat keluarannya, putuskan apa yang dicetaknya, lalu putuskan apa yang akan dicetaknya seandainya `i` bernilai 1.

Kode 3.33 `IfDemo.java`: sebuah `if`/`else`, dan rantai `else if` yang dicoba untuk beberapa nilai.

```
public class IfDemo {

    public static void main(String[] args) {
        int i = 0;
        int j = 3;
        if (i > 0) {
            System.out.println("i is greater than 0!");
            System.out.println("j is " + (j - 1));
        } else {
            System.out.println("i is less than or equal to 0");
            System.out.println("j is " + j);
        }

        for (int k = 2; k <= 6; k++) {
            if (k > 5) {
                System.out.println(k + ": more than 5");
            } else if (k > 4) {
                System.out.println(k + ": more than 4");
            } else if (k > 3) {
                System.out.println(k + ": more than 3");
            } else {
                System.out.println(k + ": 3 or less");
            }
        }
    }
}
```

```
i is less than or equal to 0
j is 3
2: 3 or less
3: 3 or less
4: more than 3
5: more than 4
6: more than 5
```

Dengan `i = 0`, kondisi `i > 0` bernilai salah, sehingga hanya blok `else` yang dijalankan: ia mencetak `i is less than or equal to 0` lalu `j is 3`. Seandainya `i` bernilai 1, kondisinya benar, blok pertama dijalankan, dan program mencetak `i is greater than 0!` lalu `j is 2`, karena blok itu mencetak `j - 1`. Perhatikan bahwa tepat satu dari kedua blok yang dijalankan, tidak pernah keduanya dan tidak pernah tidak satu pun.

Versi Python dalam **Kode 3.34** memiliki logika yang sama dengan tanda baca yang lebih sedikit: tanpa kurung di sekitar kondisi, titik dua di akhir baris `if`, dan indentasi untuk menandai blok. Python menuliskan "else if" sebagai satu kata, `elif`.

Kode 3.34 `if_demo.py`: `if`, `elif`, dan `else` dalam Python.

```python
def main():
    i = 0
    j = 3
    if i > 0:
        print("i is greater than 0!")
        print("j is " + str(j - 1))
    else:
        print("i is less than or equal to 0")
        print("j is " + str(j))

    for k in range(2, 7):
        if k > 5:
            print(str(k) + ": more than 5")
        elif k > 4:
            print(str(k) + ": more than 4")
        elif k > 3:
            print(str(k) + ": more than 3")
        else:
            print(str(k) + ": 3 or less")


if __name__ == "__main__":
    main()
```

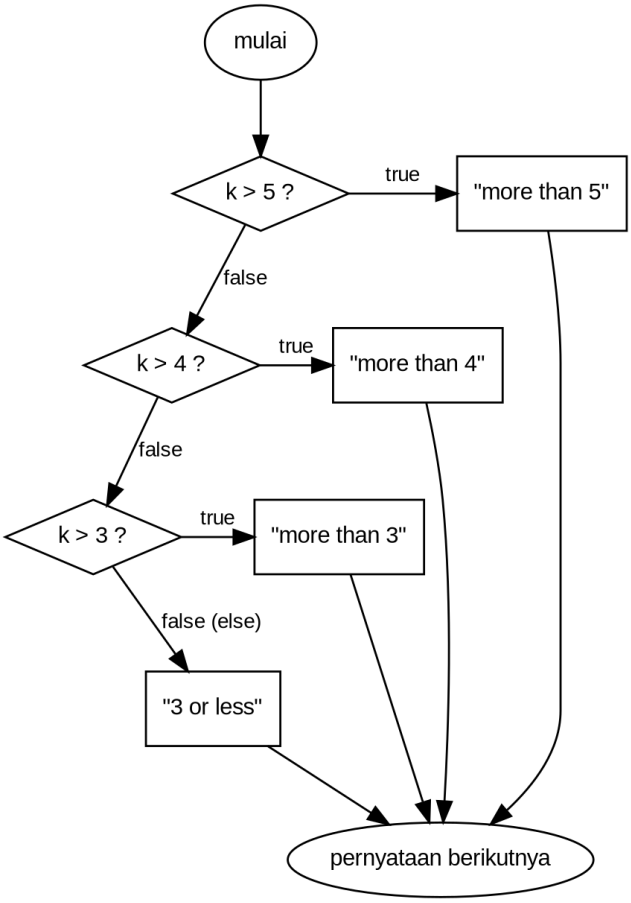
```
i is less than or equal to 0
j is 3
2: 3 or less
3: 3 or less
4: more than 3
5: more than 4
6: more than 5
```

3.8.2 Rantai keputusan

Anda boleh terus menambahkan cabang `else if` sebanyak yang diperlukan. Paruh kedua **Kode 3.33** mencoba rantai semacam itu untuk `k` dari 2 sampai 6:

```
if (k > 5) { ... } else if (k > 4) { ... } else if (k > 3) { ... } else { ... }
```

Kondisi-kondisinya diuji *berurutan, dari atas*, dan kondisi benar yang pertama menang; sisanya dilewati. Urutan itu penting. Ketika `k` bernilai 6, ketiga kondisi benar, tetapi hanya cabang pertama yang dijalankan dan mencetak `more than 5`. Ketika `k` bernilai 2 atau 3, tidak satu pun benar, dan `else` penutup yang menangkapnya. **Gambar 3.15** menggambarkan rantai itu sebagai diagram alir, dan **Tabel 3.9** mencatat pengujian mana yang memutuskan setiap nilai.



Gambar 3.15 Rantai `else if` sebagai diagram alir: kondisi dicoba dari atas, dan kondisi benar yang pertama yang memutuskan.

Tabel 3.9 Menelusuri rantai `else if` pada **Kode 3.33** untuk $k = 2, \ldots, 6$.

k	k > 5?	k > 4?	k > 3?	Tercetak
2	false	false	false	`2: 3 or less`
3	false	false	false	`3: 3 or less`
4	false	false	true	`4: more than 3`
5	false	true	(tidak diuji)	`5: more than 4`
6	true	(tidak diuji)	(tidak diuji)	`6: more than 5`

Seandainya pengujian ditulis dalam urutan terbalik, dimulai dengan `k > 3`, maka 4, 5, dan 6 akan berhenti pada pengujian pertama dan semuanya mencetak `more than 3`. Bila kondisi-kondisinya bertumpang tindih, letakkan kondisi yang paling khusus di depan.

3.8.3 `switch`: memilih di antara banyak nilai

Rantai `if` cocok untuk sedikit kasus, tetapi ketika satu variabel dibandingkan dengan banyak nilai tertentu, rantai itu menjadi panjang dan berulang-ulang. Pernyataan `switch` dibuat untuk keadaan ini. Ia menerima variabel bertipe tertentu (`byte`, `short`, `int`, `char`, juga `String` dan enumerasi) lalu melompat ke `case` yang cocok dengan nilainya.

Bentuk klasik `switch` memiliki jebakan, dan **Kode 3.35** sengaja langsung terperosok ke dalamnya.

Kode 3.35 `SwitchDemo.java`: *fall-through* tanpa `break`, perbaikannya dengan `break` dan `default`, serta bentuk panah.

```
public class SwitchDemo {

    public static void main(String[] args) {
        char input = 'b';

        System.out.println("-- without break --");
        switch (input) {
            case 'a': {
                System.out.println("Input was a");
            }
            case 'b': {
                System.out.println("Input was b");
            }
            case 'c': {
                System.out.println("Input was c");
            }
        }

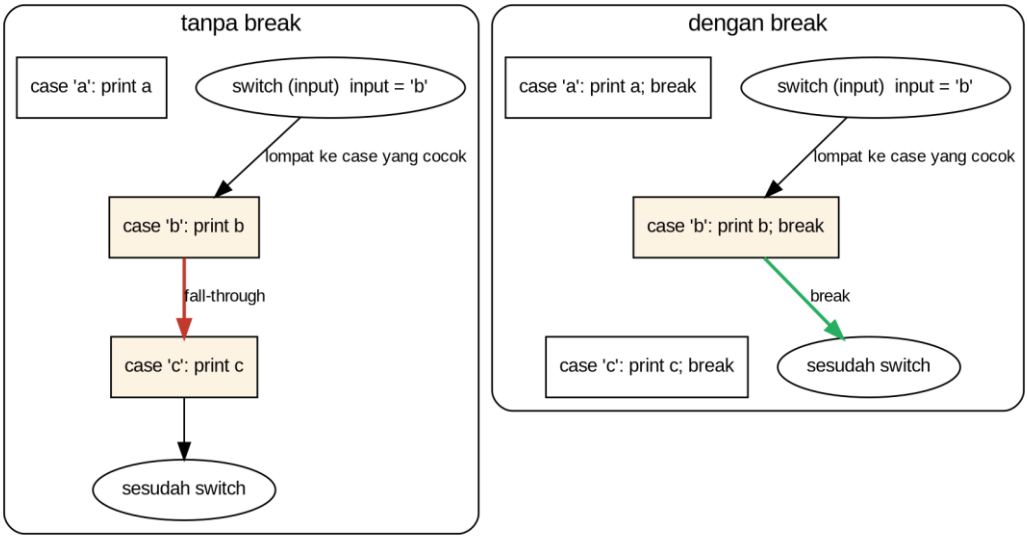
        System.out.println("-- with break and default --");
        char[] tries = {'b', 'z'};
        for (char c : tries) {
            switch (c) {
                case 'a': {
                    System.out.println("Input was a");
                    break;
                }
                case 'b': {
                    System.out.println("Input was b");
                    break;
                }
                case 'c': {
                    System.out.println("Input was c");
                    break;
                }
                default: {
                    System.out.println("Unexpected letter");
                }
            }
        }
```

Kode 3.35 (Lanjutan) *SwitchDemo.java: fall-through tanpa break, perbaikannya dengan break dan default, serta bentuk panah.*

```
        System.out.println("-- arrow form (Java 14+) --");
        String code = "EXP";
        String label = switch (code) {
            case "REG" -> "Regular, 2 days";
            case "EXP" -> "Express, same day";
            default -> "Unknown service";
        };
        System.out.println(label);
    }
}
```

```
-- without break --
Input was b
Input was c
-- with break and default --
Input was b
Unexpected letter
-- arrow form (Java 14+) --
Express, same day
```

`switch` pertama dimaksudkan untuk melaporkan huruf apa yang disimpan `input`. Dengan `input = 'b'` seharusnya tercetak satu baris, tetapi yang tercetak dua: `Input was b` *dan* `Input was c`. Sebabnya, label `case` hanyalah *pintu masuk*. Begitu eksekusi melompat masuk di `case 'b':`, ia terus mengalir ke bawah melalui setiap pernyataan berikutnya, termasuk yang berada di bawah `case 'c':`, sampai mencapai akhir `switch`. Gejala ini disebut ***fall-through*** (jatuh terus), dan **Gambar 3.16** menggambarkannya.



Gambar 3.16 Tanpa `break`, eksekusi jatuh terus dari *case* yang cocok ke *case* berikutnya; dengan `break`, eksekusi keluar dari `switch`.

Obatnya adalah kata kunci `break` di akhir setiap *case*, yang langsung melompat keluar dari `switch`. `switch` kedua dalam **Kode 3.35** menambahkan `break` pada setiap *case*, dan menambahkan *case* `default`, yang dijalankan bila tidak ada *case* yang cocok. Perulangannya mencoba dua huruf: `'b'` hanya mencetak `Input was b`, dan `'z'`, yang tidak cocok dengan apa pun, mencetak `Unexpected letter`. *Case* `default` tidak memerlukan `break`, karena ia yang terakhir dan tidak ada lagi tempat untuk jatuh.

Sejak Java 14, bentuk kedua `switch` menghapus jebakan itu sepenuhnya [9]. Dalam **bentuk panah**, `case "EXP" -> "Express, same day";`, setiap *case* berupa satu ungkapan atau blok, tidak ada *fall-through* dan tidak ada `break` yang bisa terlupa, dan seluruh `switch` menghasilkan nilai yang dapat ditugaskan. Bagian ketiga **Kode 3.35** memakainya untuk mengubah kode layanan menjadi label. Utamakan bentuk panah dalam kode baru; pelajari bentuk klasik karena Anda akan menjumpainya dalam program-program lama.

Python sama sekali tidak mengenal *fall-through*. Sejak versi 3.10, Python memiliki pernyataan `match` [10], yang `case _:`-nya berperan sebagai `default`, dan untuk pencarian sederhana pemrogram Python sering memakai *dictionary* (kamus), seperti diperlihatkan **Kode 3.36**. *Dictionary* dibahas kembali di Bab 9.

Kode 3.36 `switch_demo.py`: `match` (Python 3.10 ke atas), rantai `elif`, dan pencarian dalam *dictionary*.

```
def describe(letter):
    match letter:  # Python 3.10+; match tidak pernah fall-through
        case "a":
            return "Input was a"
        case "b":
            return "Input was b"
        case "c":
            return "Input was c"
        case _:
            return "Unexpected letter"


def main():
    for c in ["b", "z"]:
        print(describe(c))

    code = "EXP"
    if code == "REG":
        label = "Regular, 2 days"
    elif code == "EXP":
        label = "Express, same day"
    else:
        label = "Unknown service"
    print(label)

    labels = {"REG": "Regular, 2 days", "EXP": "Express, same day"}
    print(labels.get("XYZ", "Unknown service"))


if __name__ == "__main__":
    main()
```

```
Input was b
Unexpected letter
Express, same day
Unknown service
```

Kapan sebaiknya memakai `switch` alih-alih `if`? Pakailah `switch` (atau `match`) bila setiap cabang membandingkan *satu* variabel dengan *nilai-nilai konstan tertentu*. Pakailah `if`/`else if` bila cabang-cabangnya menguji rentang (`k > 4`), beberapa variabel yang berbeda, atau kondisi sembarang.

3.9 Perulangan

3.9.1 `while` dan `for`

Kadang kita ingin melakukan sesuatu beberapa kali. Kita bisa saja menulis

```
System.out.println("Hello");
System.out.println("Hello");
System.out.println("Hello");
System.out.println("Hello");
```

tetapi cara itu menjemukan, dan mustahil bila jumlahnya baru diketahui saat program berjalan. Sebuah **perulangan** (*loop*) berkata “lakukan ini, empat kali”; dua jenis utamanya adalah `while` dan `for`.

Berikut percobaan pertama dengan keduanya. Keduanya dimaksudkan mencetak `Hello` empat kali, tetapi salah satunya tidak benar. Yang mana, dan mengapa?

```
int i = 0;
while (i < 4) {
    System.out.println("Hello");
}

for (int j = 0; j < 4; j++) {
    System.out.println("Hello");
}
```

Perulangan `for` sudah benar. Perulangan `while` tidak pernah berhenti. Kondisinya, `i < 4`, diperiksa sebelum setiap pengulangan, dan tidak ada apa pun di badannya yang mengubah `i`, sehingga `i` tetap 0, kondisinya tetap benar, dan program mencetak `Hello` selama-lamanya (sampai Anda menghentikannya dengan Ctrl+C, atau dengan tombol berhenti di editor Anda). Perulangan semacam ini disebut **perulangan tak berujung** (*infinite loop*). Perbaikannya satu baris, `i++;`, di akhir badan perulangan. **Kode 3.37** memuat perulangan-perulangan yang sudah diperbaiki, masing-masing mencetak pencacahnya agar Anda dapat melihatnya bertambah.

Kode 3.37 `LoopBasics.java`: perulangan `while` yang sudah diperbaiki, perulangan `for` yang setara, dan perulangan `do-while`.

```java
public class LoopBasics {

    public static void main(String[] args) {
        int i = 0;
        while (i < 4) {
            System.out.println("Hello (while, i = " + i + ")");
            i++;                        // tanpa baris ini perulangan tidak pernah berhenti
        }

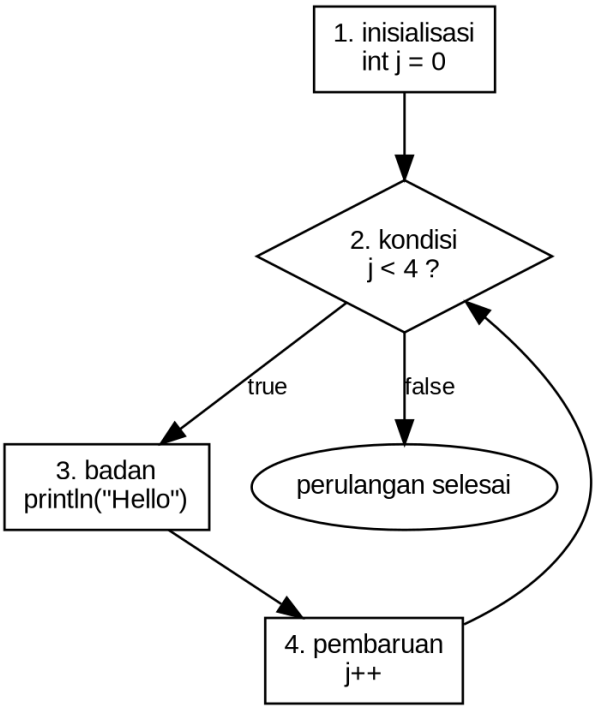
        for (int j = 0; j < 4; j++) {
            System.out.println("Hello (for, j = " + j + ")");
        }

        int k = 10;
        do {
            System.out.println("do-while runs once even though k = " + k);
        } while (k < 4);
    }
}
```

```
Hello (while, i = 0)
Hello (while, i = 1)
Hello (while, i = 2)
Hello (while, i = 3)
Hello (for, j = 0)
Hello (for, j = 1)
Hello (for, j = 2)
Hello (for, j = 3)
do-while runs once even though k = 10
```

Perulangan `while` hanya memiliki kondisi. Sebelum setiap pengulangan, yang disebut **iterasi**, ia memeriksa kondisinya; bila benar, ia menjalankan badannya lalu memeriksa lagi; begitu kondisinya salah, perulangan berakhir dan program berlanjut sesudahnya. Bila kondisinya sudah salah sejak pertama kali, badannya dijalankan nol kali. Tanggung jawab pentingnya ada pada Anda: sesuatu di dalam badan perulangan pada akhirnya harus membuat kondisinya salah.

Perulangan `for` menaruh seluruh pembukuannya dalam satu baris, dalam tiga bagian yang dipisahkan titik koma: `for (initialization; condition; update)`. **Gambar 3.17** memperlihatkan urutan bagian-bagian itu dijalankan.



Gambar 3.17 Empat bagian `for (int j = 0; j < 4; j++)`: inisialisasi sekali, lalu kondisi, badan, dan pembaruan dalam satu siklus sampai kondisinya salah.

1. **Inisialisasi**, `int j = 0`, dijalankan sekali, sebelum yang lain.
2. **Kondisi**, `j < 4`, diperiksa sebelum setiap iterasi; bila salah, perulangan berakhir.
3. **Badan** dijalankan.
4. **Pembaruan**, `j++`, dijalankan sesudah badan, lalu kendali kembali ke langkah 2.

Jadi `j` bernilai 0, 1, 2, dan 3 di dalam badan. Dengan `for`, satu baris sudah memperlihatkan persis berapa kali badan dijalankan; dengan `while`, pencacahan tersebar di tiga tempat, dan mudah melupakan salah satunya. Pakailah `for` bila Anda tahu banyaknya iterasi, dan `while` bila Anda mengulang *sampai sesuatu terjadi*, misalnya "sampai saldonya berlipat dua" pada Soal Latihan 3.8.

Perulangan ketiga dalam **Kode 3.37** adalah **perulangan `do-while`**. Ia memeriksa kondisinya *sesudah* badan, sehingga badan selalu dijalankan setidaknya sekali, bahkan di sini, ketika kondisi `k < 4` sudah salah sejak awal. Perulangan ini cocok untuk keadaan "tanya dulu, baru periksa", misalnya meminta masukan pengguna sampai masukannya sah.

Perulangan Python, dalam **Kode 3.38**, tampak berbeda tetapi bekerja sama.

Kode 3.38 `loop_basics.py`: perulangan `while` Python, perulangan `for` atas `range`, dan pola `while True` yang menggantikan `do-while`.

```
def main():
    i = 0
    while i < 4:
        print("Hello (while, i = " + str(i) + ")")
        i += 1  # tanpa baris ini perulangan tidak pernah berhenti

    for j in range(4):
        print("Hello (for, j = " + str(j) + ")")

    k = 10
    while True:  # Python tidak punya do-while; inilah pola yang lazim
        print("do-while style runs once even though k = " + str(k))
        if not k < 4:
            break


if __name__ == "__main__":
    main()
```

```
Hello (while, i = 0)
Hello (while, i = 1)
Hello (while, i = 2)
Hello (while, i = 3)
Hello (for, j = 0)
Hello (for, j = 1)
Hello (for, j = 2)
Hello (for, j = 3)
do-while style runs once even though k = 10
```

`while` di Python sama dengan di Java. `for` di Python selalu menyusuri butir-butir sebuah deretan; untuk mencacah, Anda meminta deretan bilangan dengan **`range`**: `range(4)` menghasilkan 0, 1, 2, 3; `range(1, 11)` menghasilkan 1 sampai 10 (ujungnya tidak disertakan); `range(10, 0, -1)` mencacah mundur dari 10 ke 1; dan `range(3, 16, 3)` menghasilkan 3, 6, 9, 12, 15. Python tidak memiliki `do-while`; penggantinya yang lazim adalah `while True:` dengan sebuah `break`. Di kedua bahasa, `break` langsung meninggalkan perulangan, dan `continue` melewati sisa iterasi yang sedang berjalan.

3.9.2 Empat deret

Berikut empat soal perulangan kecil. Cobalah masing-masing di atas kertas sebelum melihat penyelesaiannya. Bagaimana Anda menulis perulangan `for` yang mencetak

1. bilangan 1 sampai 10 dalam urutan naik?
2. bilangan 1 sampai 10 dalam urutan turun?
3. bilangan 3, 6, 9, 12, 15?
4. bilangan kuadrat 1, 4, 9, 16, 25, ..., 100, yaitu kuadrat sampai 10^2?

Setiap jawabannya adalah perulangan `for` yang ketiga bagiannya dipilih dengan cermat, dalam **Kode 3.39** dan **Kode 3.40**.

Kode 3.39 `LoopSequences.java`: empat perulangan `for`, satu untuk setiap deret.

```java
public class LoopSequences {

    public static void main(String[] args) {
        // 1-10, menaik
        for (int i = 1; i <= 10; i++) {
            System.out.print(i + " ");
        }
        System.out.println();

        // 1-10, menurun
        for (int i = 10; i >= 1; i--) {
            System.out.print(i + " ");
        }
        System.out.println();

        // 3, 6, 9, 12, 15
        for (int i = 3; i <= 15; i += 3) {
            System.out.print(i + " ");
        }
        System.out.println();

        // 1, 4, 9, ..., 100: kuadrat dari 1..10
        for (int i = 1; i <= 10; i++) {
            System.out.print(i * i + " ");
        }
        System.out.println();
    }
}
```

```
1 2 3 4 5 6 7 8 9 10
10 9 8 7 6 5 4 3 2 1
3 6 9 12 15
1 4 9 16 25 36 49 64 81 100
```

Kode 3.40 `loop_sequences.py`: deret-deret yang sama dengan `range`.

```python
def main():
    # 1-10, menaik
    for i in range(1, 11):
        print(i, end=" ")
    print()

    # 1-10, menurun
    for i in range(10, 0, -1):
        print(i, end=" ")
    print()

    # 3, 6, 9, 12, 15
    for i in range(3, 16, 3):
        print(i, end=" ")
    print()

    # 1, 4, 9, ..., 100: kuadrat dari 1..10
    for i in range(1, 11):
        print(i * i, end=" ")
    print()


if __name__ == "__main__":
    main()
```

```
1 2 3 4 5 6 7 8 9 10
10 9 8 7 6 5 4 3 2 1
3 6 9 12 15
1 4 9 16 25 36 49 64 81 100
```

Mari kita nalar satu per satu.

1. **Naik.** Mulai dari 1, lanjutkan selama `i <= 10`, tambahkan 1 setiap kali. Menulis `i < 10` akan berhenti di 9; kelebihan atau kekurangan satu iterasi begitu sering terjadi sampai memiliki nama sendiri, **galat selisih satu** (*off-by-one error*). Di Python, `range(1, 11)` memerlukan 11 karena ujungnya tidak disertakan.
2. **Turun.** Mulai dari 10, lanjutkan selama `i >= 1`, dan *kurangkan* 1 dengan `i--`. Di Python langkahnya `-1`: `range(10, 0, -1)` berhenti sebelum 0.
3. **Kelipatan tiga.** Mulai dari 3 dan tambahkan 3 setiap kali dengan `i += 3`; berhenti sesudah 15 dengan `i <= 15`. Alternatif yang sama baiknya adalah mencacah `i` dari 1 sampai 5 lalu mencetak `3 * i`.
4. **Kuadrat.** Alih-alih mencari aturan pembaruan yang cerdik, cacahlah `i` dari 1 sampai 10 dan cetak `i * i`. Pencacah perulangan tidak harus berupa bilangan yang dicetak; ia hanya perlu *menentukannya*.

`System.out.print`, tanpa `ln`, mencetak tanpa mengakhiri baris, sehingga bilangan-bilangannya tampil berdampingan; `System.out.println()` kosong sesudah setiap perulangan mengakhiri barisnya. `print(i, end=" ")` di Python melakukan hal yang sama dengan mengganti akhir baris yang biasa dengan sebuah spasi.

3.9.3 Perulangan for-each

Sering kali sebuah perulangan hanya perlu mengunjungi setiap elemen sebuah koleksi, satu demi satu, tanpa memedulikan posisinya. Kedua bahasa memiliki perulangan khusus untuk itu, perulangan **for-each** (untuk setiap). Di Java ditulis `for (String word : args)`, dibaca "untuk setiap `word` dalam `args`". Di Python bentuknya cukup `for word in args:` yang biasa. **Kode 3.41** mencetak kata-kata baris perintah dua kali, pertama dengan perulangan yang mencacah indeks, lalu dengan perulangan for-each, dan **Kode 3.42** melakukan hal yang sama dalam Python.

Kode 3.41 `ArgsEcho.java`: mengunjungi setiap kata baris perintah melalui indeks dan dengan perulangan for-each.

```java
public class ArgsEcho {

    public static void main(String[] args) {
        System.out.println("You gave " + args.length + " word(s).");
        for (int i = 0; i < args.length; i++) {
            System.out.println("args[" + i + "] = " + args[i]);
        }
        for (String word : args) {            // for-each: tidak perlu indeks
            System.out.print(word.toUpperCase() + " ");
        }
        System.out.println();
    }
}
```

Dijalankan dengan tiga kata, `java ArgsEcho wash dry iron`:

```
You gave 3 word(s).
args[0] = wash
args[1] = dry
args[2] = iron
WASH DRY IRON
```

Kode 3.42 `args_echo.py`: hal yang sama dalam Python, dengan `enumerate` untuk indeks dan nilai sekaligus.

```python
import sys


def main():
    args = sys.argv[1:]  # sys.argv[0] adalah nama skrip itu sendiri
    print("You gave " + str(len(args)) + " word(s).")
    for i in range(len(args)):
        print("args[" + str(i) + "] = " + args[i])
    for i, word in enumerate(args):  # indeks dan nilai sekaligus
        print(str(i) + ":" + word, end=" ")
    print()
    for word in args:  # for-each: tidak perlu indeks
        print(word.upper(), end=" ")
    print()


if __name__ == "__main__":
    main()
```

Dijalankan sebagai `python3 args_echo.py wash dry iron`:

```
You gave 3 word(s).
args[0] = wash
args[1] = dry
args[2] = iron
0:wash 1:dry 2:iron
WASH DRY IRON
```

Perulangan for-each lebih pendek dan tidak memiliki indeks yang bisa keliru; pakailah setiap kali Anda tidak memerlukan posisinya. Bila Anda memerlukan keduanya, pakailah perulangan berindeks di Java atau `enumerate` di Python. Larik dan *list* adalah pokok bahasan Bab 5.

3.9.4 Memilih perulangan yang tepat

Tabel 3.10 Bentuk-bentuk perulangan Java dan Python, dan kapan memakai masing-masing.

Perulangan	Memeriksa kondisi	Dijalankan setidaknya sekali?	Paling cocok bila
`for (init; cond; update)` / `for i in range(...)`	sebelum setiap iterasi	tidak	banyaknya iterasi diketahui
`while (cond)` / `while cond:`	sebelum setiap iterasi	tidak	mengulang sampai sesuatu terjadi
`do { } while (cond);` / `while True: ... break`	sesudah setiap iterasi	ya	harus bertindak sebelum dapat memeriksa
`for (T x : items)` / `for x in items:`	(mengunjungi setiap butir)	tidak (bila kosong)	mengunjungi setiap elemen, tanpa indeks

Tabel 3.10 merangkum pilihannya. Perulangan apa pun dapat ditulis ulang sebagai perulangan jenis lain, sehingga pilihannya bukan soal kemampuan melainkan kejelasan: pilihlah perulangan yang bentuknya, sekali pandang, memberi tahu pembaca jenis pengulangan apa yang sedang terjadi.

3.10 Tabel Penelusuran

Bagaimana Anda *tahu* apa yang dilakukan sebuah perulangan? Menjalankannya hanya memberi tahu apa yang terjadi untuk satu masukan, dan tidak memberi tahu *mengapa* keluaran yang salah itu salah. Teknik profesionalnya adalah **menelusuri** (*trace*) perulangan dengan tangan memakai **tabel penelusuran** (*tracing table*): satu kolom untuk setiap variabel, satu baris untuk setiap saat

yang ingin Anda catat, biasanya akhir setiap iterasi. Anda mengisinya dengan pensil, berperan sebagai komputer, satu pernyataan demi satu pernyataan [11]. Tabel ini adalah *debugger* dari Bab 2 yang Anda bawa di dalam kepala.

Mari kita telusuri **Kode 3.43** (dalam Python, **Kode 3.44**). Program ini menerima bilangan 4071 dan menghitung dua hal sekaligus: jumlah digit-digitnya dan bilangan dengan digit-digit yang dibalik.

Kode 3.43 `TraceDemo.java`: perulangan `while` yang membongkar sebuah bilangan digit demi digit.

```java
public class TraceDemo {

    public static void main(String[] args) {
        int n = 4071;
        int digitSum = 0;
        int reversed = 0;
        while (n > 0) {
            int digit = n % 10;              // digit terakhir
            digitSum = digitSum + digit;
            reversed = reversed * 10 + digit;
            n = n / 10;                      // buang digit terakhir
        }
        System.out.println("digit sum = " + digitSum);
        System.out.println("reversed  = " + reversed);
    }
}
```

```
digit sum = 12
reversed  = 1704
```

Kode 3.44 `trace_demo.py`: perulangan yang sama dalam Python, dengan `//` untuk pembagian bilangan bulat.

```python
def main():
    n = 4071
    digit_sum = 0
    reversed_number = 0
    while n > 0:
        digit = n % 10  # digit terakhir
        digit_sum = digit_sum + digit
        reversed_number = reversed_number * 10 + digit
        n = n // 10  # buang digit terakhir
    print("digit sum = " + str(digit_sum))
    print("reversed  = " + str(reversed_number))


if __name__ == "__main__":
    main()
```

```
digit sum = 12
reversed  = 1704
```

Perulangan ini memakai dua alat dari Subbab 3.7.4. `n % 10` adalah digit terakhir `n`, dan `n / 10` (di Python `n // 10`) adalah `n` tanpa digit terakhirnya. Sebelum perulangan, `n = 4071`, `digitSum = 0`, `reversed = 0`. **Tabel 3.11** mencatat keadaan sesudah setiap iterasi.

Tabel 3.11 Menelusuri **Kode 3.43** untuk $n = 4071$; perulangan berakhir ketika n mencapai 0.

Iterasi	n di awal	digit = n % 10	digitSum	reversed	n = n / 10	n > 0?
(sebelum)	4071		0	0		true
1	4071	1	0 + 1 = 1	0 · 10 + 1 = 1	407	true
2	407	7	1 + 7 = 8	1 · 10 + 7 = 17	40	true
3	40	0	8 + 0 = 8	17 · 10 + 0 = 170	4	true
4	4	4	8 + 4 = 12	170 · 10 + 4 = 1704	0	false

Tabel itu memperlihatkan mengapa keluarannya `12` dan `1704`, dan lebih dari itu: perulangan berjalan sekali untuk setiap digit, karena setiap iterasi membuang satu digit, dan nol di ujung 4071 menjadi nol *di depan* hasil pembalikan, yang lenyap, sehingga kebalikannya 1704, bukan 17040.

Tabel penelusuran memiliki beberapa aturan yang menjaganya tetap jujur.

1. Buat satu kolom untuk setiap variabel yang berubah, ditambah satu untuk kondisi perulangan.
2. Tulis nilai-nilai awal pada baris pertama, sebelum perulangan.
3. Jalankan badan perulangan satu pernyataan demi satu pernyataan, *berurutan*, memakai nilai-nilai sebagaimana adanya pada saat itu, bukan sebagaimana nanti di akhir iterasi.
4. Catat keadaan pada titik yang sama di setiap iterasi, dan periksa kondisi pada titik tempat perulangan memeriksanya.
5. Berhenti ketika kondisinya salah, lalu baca hasilnya dari baris terakhir.

Aturan 3 yang paling sering dilanggar. Pada iterasi 2, `reversed = reversed * 10 + digit` memakai `reversed` dari iterasi 1 (yaitu 1) dan `digit` yang baru saja dihitung pada iterasi yang sama (7). Salah urutan, dan tabelnya berbohong.

Subbab 3.11 dan soal-soal latihan memakai tabel penelusuran lagi, dan Bab 4 membahas soal-soal utuh dengan jenis yang persis seperti ini.

3.11 Latihan Terbimbing, Langkah demi Langkah

Tiga soal dalam subbab ini memakai semua yang sudah kita pelajari. Untuk setiap soal kita memahami persoalannya, memilih tipe-tipenya, menulis programnya dalam kedua bahasa, dan memeriksa keluarannya dengan tangan, langkah yang mengubah "programnya mencetak sesuatu" menjadi "programnya benar".

3.11.1 Luas lingkaran

Soal. Luas A sebuah lingkaran berjari-jari r adalah

$$A = \pi r^2 \tag{3.5}$$

Tulislah program yang menghitungnya, dengan variabel `r` diberi nilai awal 5, lalu cetak hasilnya. Tipe apa yang sebaiknya dimiliki variabel `A` dan `r`?

Memilih tipe. Jari-jari bisa saja berupa desimal, misalnya 2,5 cm, dan luasnya pasti desimal, karena $\pi = 3.14159\ldots$ bukan bilangan bulat. Jadi kedua variabel bertipe `double`. Luas bertipe `int` akan keliru: Java menolak menyimpan hasil kali bertipe `double` ke dalamnya tanpa konversi tipe, dan konversi tipe akan membuang pecahannya. `Math.PI` di Java dan `math.pi` di Python menyediakan π dengan presisi `double` penuh. Persamaan (3.5) kemudian menjadi satu baris

kode, `Math.PI * r * r`; mengalikan jari-jari dengan dirinya sendiri lebih sederhana dan lebih akurat daripada `Math.pow(r, 2)`.

Kode 3.45 dan **Kode 3.46** memberikan penyelesaiannya. Komentar dokumentasi di awalnya menjelaskan pilihan-pilihan itu, sebagaimana layaknya penyelesaian yang baik. Nama variabel Java diawali huruf kecil, sehingga luasnya dinamai `a`.

Kode 3.45 `CircleArea.java`: luas lingkaran berjari-jari 5.

```java
public class CircleArea {
    /** Variabel a dan r dideklarasikan sebagai double; a dihitung
     *  dari r sebagai Math.PI * r * r lalu dicetak dengan rapi.
     *  Program diuji dengan r = 5.
     */
    public static void main(String[] args) {
        double a;
        double r;
        r = 5;
        a = Math.PI * r * r;
        System.out.println("The area of a circle with radius " + r
                + " is " + a + ".");
        System.out.printf("Rounded to two decimals: %.2f%n", a);
    }
}
```

```
The area of a circle with radius 5.0 is 78.53981633974483.
Rounded to two decimals: 78.54
```

Kode 3.46 `circle_area.py`: perhitungan yang sama dalam Python.

```python
import math


def main():
    """a dan r bertipe float; a dihitung dari r sebagai math.pi * r * r
    lalu dicetak dengan rapi. Program diuji dengan r = 5.
    """
    r = 5.0
    a = math.pi * r * r
    print("The area of a circle with radius " + str(r)
          + " is " + str(a) + ".")
    print(f"Rounded to two decimals: {a:.2f}")


if __name__ == "__main__":
    main()
```

```
The area of a circle with radius 5.0 is 78.53981633974483.
Rounded to two decimals: 78.54
```

Memeriksa dengan tangan. $r^2 = 25$ dan $25 \times 3.14159265\ldots = 78.5398\ldots$, cocok. Keluaran mentahnya menampilkan enam belas digit signifikan, jauh lebih banyak daripada yang diperlukan siapa pun, dan karena itulah baris kedua memformat luasnya menjadi dua angka desimal dengan `%.2f`.

3.11.2 Dari satuan imperial ke kilogram

Soal. Tulislah program yang mengubah massa dalam sistem imperial menjadi kilogram, dengan memakai konversi dalam **Tabel 3.12**. Berat seseorang adalah 11 *stone* dan 6 pon; simpanlah sebagai `stones = 11` dan `pounds = 6`, dengan semua satuan lain bernilai 0. Berapa kilogramkah itu?

Tabel 3.12 Satuan massa imperial dan nilainya dalam pon.

Satuan	Dalam pon
1 *ton*	2240 pon
1 *hundredweight*	112 pon
1 *quarter*	28 pon
1 *stone*	14 pon
1 *ounce*	$\frac{1}{16}$ pon
1 *drachm*	$\frac{1}{256}$ pon
1 *grain*	$\frac{1}{7000}$ pon
1 pon	0,45359237 kilogram

Merencanakan. Rencananya terdiri atas dua tahap. Pertama, ubah setiap satuan menjadi pon lalu jumlahkan; kemudian ubah pon menjadi kilogram. Dengan t, h, q, s, p, o, d, g sebagai banyaknya *ton*, *hundredweight*, *quarter*, *stone*, pon, *ounce*, *drachm*, dan *grain*, berat dalam pon adalah

$$W_{\text{lb}} = 2240t + 112h + 28q + 14s + p + \frac{o}{16} + \frac{d}{256} + \frac{g}{7000} \tag{3.6}$$

dan berat dalam kilogram adalah

$$W_{\text{kg}} = 0.45359237 \cdot W_{\text{lb}} \tag{3.7}$$

Alih-alih menaburkan bilangan 2240, 112, 28, dan seterusnya di sepanjang rumus, program lebih dulu menyimpan setiap faktor konversi dalam variabel bernama jelas seperti `poundsPerStone`. Pembaca lalu melihat `stones * poundsPerStone` dan langsung memahaminya, dan faktor yang salah ketik hanya punya satu tempat untuk bersembunyi. Bilangan yang muncul dalam kode tanpa nama disebut **bilangan ajaib** (*magic number*), dan memberinya nama adalah salah satu perbaikan termurah yang dapat Anda lakukan pada program apa pun.

Jebakan dalam faktor. Perhatikan cara pecahan-pecahannya ditulis: `1 / 16.0`, bukan `1 / 16`. Menurut Subbab 3.7.4, `1 / 16` adalah pembagian bilangan bulat yang menghasilkan 0, yang diam-diam akan membuat setiap *ounce* tak berbobot. Menulis `16.0` menjadikannya pembagian desimal. (Di Python, `1 / 16` sudah merupakan pembagian desimal.) Penyelesaiannya adalah **Kode 3.47** dalam Java dan **Kode 3.48** dalam Python.

Kode 3.47 `ImperialToKg.java`: mengubah 11 *stone* dan 6 pon menjadi kilogram.

```
public class ImperialToKg {
    /**
     * Faktor konversi disimpan dulu dalam variabel. Berat imperial
     * lalu diubah menjadi pon, pon menjadi kilogram, dan
     * program dijalankan untuk 11 stone dan 6 pon.
     */
    public static void main(String[] args) {
        double poundsPerTon           = 2240;
        double poundsPerHundredweight = 112;
        double poundsPerQuarter       = 28;
        double poundsPerStone         = 14;
        double poundsPerOunce         = 1 / 16.0;
        double poundsPerDrachm        = 1 / 256.0;
        double poundsPerGrain         = 1 / 7000.0;
        double kilogramsPerPound      = 0.45359237;

        double tons = 0;
        double hundredweights = 0;
        double quarters = 0;
        double stones = 0;
        double pounds = 0;
        double ounces = 0;
        double drachms = 0;
        double grains = 0;

        /* Ubah semuanya ke pon dulu, lalu kalikan
         * dengan kilogramsPerPound.
         */
        stones = 11;
        pounds = 6;
        double weightInPounds =
                tons * poundsPerTon
                + hundredweights * poundsPerHundredweight
                + quarters * poundsPerQuarter
                + stones * poundsPerStone
                + pounds
                + ounces * poundsPerOunce
                + drachms * poundsPerDrachm
                + grains * poundsPerGrain;
        double weightInKilograms = weightInPounds * kilogramsPerPound;
        System.out.println("A person with " + stones + " stones and " + pounds
                + " pounds has a weight corresponding to "
                + weightInKilograms + " kilograms.");
        System.out.printf("That is %.1f pounds, or about %.2f kg.%n",
                weightInPounds, weightInKilograms);
    }
```

Kode 3.47 (Lanjutan) *ImperialToKg.java: mengubah 11 stone dan 6 pon menjadi kilogram.*

```
}
```

```
A person with 11.0 stones and 6.0 pounds has a weight corresponding to 72.57477920000001
kilograms.
That is 160.0 pounds, or about 72.57 kg.
```

Kode 3.48 `imperial_to_kg.py`: konversi yang sama dalam Python.

```
def main():
    """Faktor konversi disimpan dulu dalam variabel. Berat imperial
    lalu diubah menjadi pon, pon menjadi kilogram, dan
    program dijalankan untuk 11 stone dan 6 pon.
    """
    pounds_per_ton = 2240
    pounds_per_hundredweight = 112
    pounds_per_quarter = 28
    pounds_per_stone = 14
    pounds_per_ounce = 1 / 16
    pounds_per_drachm = 1 / 256
    pounds_per_grain = 1 / 7000
    kilograms_per_pound = 0.45359237

    tons = 0.0
    hundredweights = 0.0
    quarters = 0.0
    stones = 0.0
    pounds = 0.0
    ounces = 0.0
    drachms = 0.0
    grains = 0.0

    # Ubah semuanya ke pon dulu, lalu kalikan dengan kilograms_per_pound.
    stones = 11.0
    pounds = 6.0
    weight_in_pounds = (tons * pounds_per_ton
                        + hundredweights * pounds_per_hundredweight
                        + quarters * pounds_per_quarter
                        + stones * pounds_per_stone
                        + pounds
                        + ounces * pounds_per_ounce
                        + drachms * pounds_per_drachm
                        + grains * pounds_per_grain)
    weight_in_kilograms = weight_in_pounds * kilograms_per_pound
    print("A person with " + str(stones) + " stones and " + str(pounds)
          + " pounds has a weight corresponding to "
          + str(weight_in_kilograms) + " kilograms.")
    print(f"That is {weight_in_pounds:.1f} pounds, or about {weight_in_kilograms:.2f} kg.")


if __name__ == "__main__":
    main()
```

```
A person with 11.0 stones and 6.0 pounds has a weight corresponding to 72.57477920000001
kilograms.
That is 160.0 pounds, or about 72.57 kg.
```

Memeriksa dengan tangan. Hanya dua suku Persamaan (3.6) yang tidak nol: $14 \times 11 = 154$ pon dari *stone*, ditambah 6 pon, menghasilkan $W_{\text{lb}} = 160$. Menurut Persamaan (3.7), $W_{\text{kg}} = 160 \times 0.45359237 = 72.5747792$ kg. Program mencetak `72.57477920000001`: bilangan yang sama dengan galat titik mengambang yang mungil pada digit keenam belas, persis efek dari Subbab 3.7.3. Orang dengan berat 11 *stone* 6 pon beratnya sekitar 72,57 kg, sebagaimana dinyatakan baris yang sudah diformat.

3.11.3 Investasi selama lima ratus tahun

Soal. Modal sebesar GBP 100 (seratus pound sterling) diinvestasikan dengan suku bunga tetap 2,3% per tahun. Bunganya ditambahkan ke modal pada akhir setiap tahun, sehingga bunga tahun berikutnya diperoleh dari jumlah yang lebih besar; inilah **bunga majemuk**. Pakailah rumus

$$\text{total} = \text{modal} \cdot (1 + 0.01 \cdot \text{suku bunga})^{\text{tahun}} \tag{3.8}$$

untuk mencetak saldo sesudah masing-masing dari 5 tahun pertama dan sesudah 500 tahun.

Mengapa rumusnya benar. Sesudah satu tahun, modal dikalikan dengan $1 + 0.023 = 1.023$: 100% semula ditambah bunga 2,3%. Tahun kedua mengalikan hasil *itu* dengan 1.023 lagi, dan seterusnya. Sesudah n tahun, modal telah dikalikan dengan 1.023 tepat n kali, yaitu 1.023^n, dan itulah Persamaan (3.8). Faktor 0.01 mengubah persentase 2,3 menjadi pecahan 0,023.

Penyelesaian pertama. Pada titik ini kita sudah mengenal variabel, aritmetika, dan pencetakan, tetapi kita berpura-pura belum mengenal perulangan. Hasilnya, **Kode 3.49** (dan **Kode 3.50** dalam Python), jujur tetapi berulang-ulang: dua pernyataan yang sama muncul enam kali, hanya banyaknya tahun yang berubah.

Kode 3.49 `Investment.java`: Persamaan (3.8) dihitung untuk enam lama investasi, satu demi satu.

```
public class Investment {
    /** Rumus total = capital * (1 + 0.01 * interestRate)^years
     *  ditulis sekali untuk setiap tahun 1, 2, 3, 4, 5, dan 500.
     *  Kode ini sengaja berulang-ulang: InvestmentLoop.java menghapus
     *  pengulangan itu dengan sebuah perulangan.
     */
    public static void main(String[] args) {
        double capital = 100;       // modal awal dalam pound sterling
        double interestRate = 2.3;  // suku bunga per tahun, dalam persen
        int years = 1;              // lamanya modal diinvestasikan
        double total;               // modal setelah bunga ditambahkan

        total = capital * Math.pow(1 + 0.01 * interestRate, years);
        System.out.println("After " + years + " years a capital of GBP " + capital
                + " grows with an interest rate of " + interestRate + "\u0025 to GBP "
                + Math.round(total));
        years = 2;
        total = capital * Math.pow(1 + 0.01 * interestRate, years);
        System.out.println("After " + years + " years a capital of GBP " + capital
                + " grows with an interest rate of " + interestRate + "\u0025 to GBP "
                + Math.round(total));
        years = 3;
        total = capital * Math.pow(1 + 0.01 * interestRate, years);
        System.out.println("After " + years + " years a capital of GBP " + capital
                + " grows with an interest rate of " + interestRate + "\u0025 to GBP "
                + Math.round(total));
        years = 4;
        total = capital * Math.pow(1 + 0.01 * interestRate, years);
        System.out.println("After " + years + " years a capital of GBP " + capital
                + " grows with an interest rate of " + interestRate + "\u0025 to GBP "
                + Math.round(total));
        years = 5;
        total = capital * Math.pow(1 + 0.01 * interestRate, years);
        System.out.println("After " + years + " years a capital of GBP " + capital
                + " grows with an interest rate of " + interestRate + "\u0025 to GBP "
                + Math.round(total));
        years = 500;
        total = capital * Math.pow(1 + 0.01 * interestRate, years);
        System.out.println("After " + years + " years a capital of GBP " + capital
                + " grows with an interest rate of " + interestRate + "\u0025 to GBP "
                + Math.round(total));
    }
}
```

```
After 1 years a capital of GBP 100.0 grows with an interest rate of 2.3% to GBP 102
After 2 years a capital of GBP 100.0 grows with an interest rate of 2.3% to GBP 105
After 3 years a capital of GBP 100.0 grows with an interest rate of 2.3% to GBP 107
After 4 years a capital of GBP 100.0 grows with an interest rate of 2.3% to GBP 110
After 5 years a capital of GBP 100.0 grows with an interest rate of 2.3% to GBP 112
After 500 years a capital of GBP 100.0 grows with an interest rate of 2.3% to GBP 8665964
```

Kode 3.50 `investment.py`: penyelesaian berulang yang sama dalam Python.

```
def main():
    """Rumus total = capital * (1 + 0.01 * interest_rate) ** years
    ditulis sekali untuk setiap tahun 1, 2, 3, 4, 5, dan 500.
    Kode ini sengaja berulang-ulang: investment_loop.py menghapus
    pengulangan itu dengan sebuah perulangan.
    """
    capital = 100.0      # modal awal dalam pound sterling
    interest_rate = 2.3  # suku bunga per tahun, dalam persen
    years = 1            # lamanya modal diinvestasikan

    total = capital * (1 + 0.01 * interest_rate) ** years
    print("After " + str(years) + " years a capital of GBP " + str(capital)
          + " grows with an interest rate of " + str(interest_rate) + "% to GBP "
          + str(round(total)))
    years = 2
    total = capital * (1 + 0.01 * interest_rate) ** years
    print("After " + str(years) + " years a capital of GBP " + str(capital)
          + " grows with an interest rate of " + str(interest_rate) + "% to GBP "
          + str(round(total)))
    years = 3
    total = capital * (1 + 0.01 * interest_rate) ** years
    print("After " + str(years) + " years a capital of GBP " + str(capital)
          + " grows with an interest rate of " + str(interest_rate) + "% to GBP "
          + str(round(total)))
    years = 4
    total = capital * (1 + 0.01 * interest_rate) ** years
    print("After " + str(years) + " years a capital of GBP " + str(capital)
          + " grows with an interest rate of " + str(interest_rate) + "% to GBP "
          + str(round(total)))
    years = 5
    total = capital * (1 + 0.01 * interest_rate) ** years
    print("After " + str(years) + " years a capital of GBP " + str(capital)
          + " grows with an interest rate of " + str(interest_rate) + "% to GBP "
          + str(round(total)))
    years = 500
    total = capital * (1 + 0.01 * interest_rate) ** years
    print("After " + str(years) + " years a capital of GBP " + str(capital)
          + " grows with an interest rate of " + str(interest_rate) + "% to GBP "
          + str(round(total)))


if __name__ == "__main__":
    main()
```

```
After 1 years a capital of GBP 100.0 grows with an interest rate of 2.3% to GBP 102
After 2 years a capital of GBP 100.0 grows with an interest rate of 2.3% to GBP 105
After 3 years a capital of GBP 100.0 grows with an interest rate of 2.3% to GBP 107
After 4 years a capital of GBP 100.0 grows with an interest rate of 2.3% to GBP 110
After 5 years a capital of GBP 100.0 grows with an interest rate of 2.3% to GBP 112
After 500 years a capital of GBP 100.0 grows with an interest rate of 2.3% to GBP 8665964
```

`Math.pow` menghitung pangkat (Python menulis `**`), dan `Math.round` membulatkan ke pound terdekat. Tanda persen ditulis `%`, yaitu *escape* Unicode-nya; kompilator mengubah *escape* itu menjadi `%` biasa sebelum membaca apa pun, sehingga `%` polos pun sama baiknya di sini. (Dalam string format, kedua bentuk itu harus digandakan menjadi `%%`, Subbab 3.5.4.) Pesan “After 1 years”

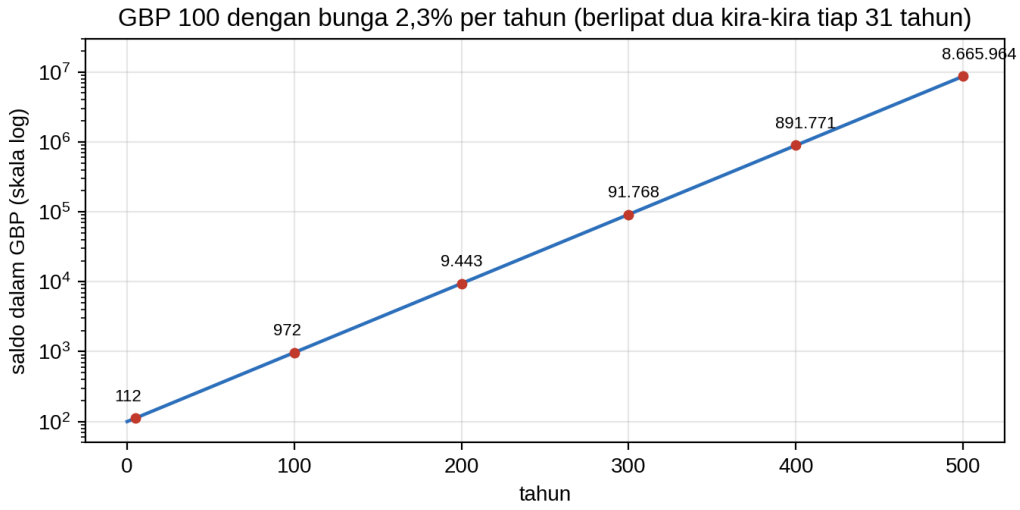
tidak gramatikal dalam bahasa Inggris; versi dengan perulangan di bawah menghindarinya dengan tata letak tabel.

Memeriksa dengan tangan. **Tabel 3.13** menghitung lima tahun pertama.

Tabel 3.13 Menghitung Persamaan (3.8) dengan tangan untuk lima tahun pertama.

Tahun n	1.023^n	$100 \cdot 1.023^n$	Dibulatkan
1	1.023000	102.30	102
2	1.046529	104.65	105
3	1.070599	107.06	107
4	1.095223	109.52	110
5	1.120413	112.04	112

Kolom yang dibulatkan cocok dengan program. Baris terakhirnyalah yang mengejutkan: sesudah 500 tahun, GBP 100 tumbuh menjadi lebih dari GBP 8,6 juta. Itulah kekuatan pemajemukan, dan **Gambar 3.18** memperlihatkannya. Pada sumbu tegak logaritmik, tempat setiap langkah naik berarti dikali sepuluh, pertumbuhan itu berupa garis lurus, karena saldonya dikalikan dengan faktor yang sama setiap tahun.



Gambar 3.18 Saldo GBP 100 dengan bunga 2,3% per tahun selama 500 tahun, pada skala logaritmik; garis lurus adalah ciri khas pertumbuhan majemuk.

Menghapus pengulangan. Kode yang berulang adalah tanda bahaya: jika pesannya salah ketik, salah ketiknya ada enam, dan memperbaiki lima di antaranya adalah *bug* yang menunggu waktu. Perulangan menghapus pengulangan itu. **Kode 3.51** menghitung saldo tahun demi tahun, mengalikan dengan faktornya sekali per iterasi, dan hanya mencetak tahun-tahun yang kita inginkan: lima tahun pertama dan setiap tahun keseratus. Di akhir, program itu mencocokkan hasil perulangan dengan rumusnya. **Kode 3.52** adalah versi Python-nya.

Kode 3.51 `InvestmentLoop.java`: saldo dihitung tahun demi tahun dalam sebuah perulangan.

```java
public class InvestmentLoop {

    public static void main(String[] args) {
        double capital = 100;       // GBP (pound sterling)
        double interestRate = 2.3;  // persen per tahun
        double factor = 1 + 0.01 * interestRate;

        System.out.println("Year      Balance (GBP)");
        double balance = capital;
        for (int year = 1; year <= 500; year++) {
            balance = balance * factor;          // tambahkan bunga satu tahun
            if (year <= 5 || year % 100 == 0) {
                System.out.printf("%4d %18.2f%n", year, balance);
            }
        }
        double direct = capital * Math.pow(factor, 500);
        System.out.printf("Formula check for year 500: %.2f%n", direct);
    }
}
```

```
Year      Balance (GBP)
   1             102.30
   2             104.65
   3             107.06
   4             109.52
   5             112.04
 100             971.77
 200            9443.36
 300           91767.76
 400          891771.29
 500         8665963.51
Formula check for year 500: 8665963.51
```

Kode 3.52 `investment_loop.py`: versi perulangan dalam Python.

```python
def main():
    capital = 100.0      # GBP (pound sterling)
    interest_rate = 2.3  # persen per tahun
    factor = 1 + 0.01 * interest_rate

    print("Year      Balance (GBP)")
    balance = capital
    for year in range(1, 501):
        balance = balance * factor  # tambahkan bunga satu tahun
        if year <= 5 or year % 100 == 0:
            print(f"{year:4d} {balance:18.2f}")
    direct = capital * factor ** 500
    print(f"Formula check for year 500: {direct:.2f}")


if __name__ == "__main__":
    main()
```

```
Year      Balance (GBP)
   1             102.30
   2             104.65
   3             107.06
   4             109.52
   5             112.04
 100             971.77
 200            9443.36
 300           91767.76
 400          891771.29
 500         8665963.51
Formula check for year 500: 8665963.51
```

Kondisi `year <= 5 || year % 100 == 0` memadukan Subbab 3.6.3 dan Subbab 3.7.4: cetak bila tahunnya salah satu dari lima tahun pertama, atau bila tahunnya habis dibagi 100. Dan kedua cara menghitung tahun ke-500, yaitu 500 perkalian dalam perulangan dan satu `Math.pow`, sepakat sampai ke sen, meskipun setiap perkalian membawa galat pembulatan yang mungil; dengan `double`, galat-galat itu tetap jauh di bawah satu sen.

3.12 Profitina Laundry: Customer dan Service

Profitina Laundry adalah usaha laundry kecil di sebuah lingkungan permukiman. Pelanggan membawa sekantong pakaian ke meja layanan; petugas menimbangnya, pelanggan memilih layanan, dan petugas menulis nota. Ada tiga layanan: *Wash & Fold* (cuci lipat) seharga Rp7.000 per kilogram, *Iron Only* (setrika saja) seharga Rp5.000 per kilogram, dan *Wash, Dry & Iron* (cuci, kering, setrika) seharga Rp10.000 per kilogram. Setiap layanan memiliki berat minimum: kantong yang lebih ringan daripada batas minimum ditagih seolah-olah beratnya sama dengan batas itu. Berat ditagih dalam kelipatan setengah kilogram, selalu dibulatkan ke atas. Layanan ekspres, selesai pada hari yang sama, 50% lebih mahal. Anggota terdaftar mendapat potongan 10%.

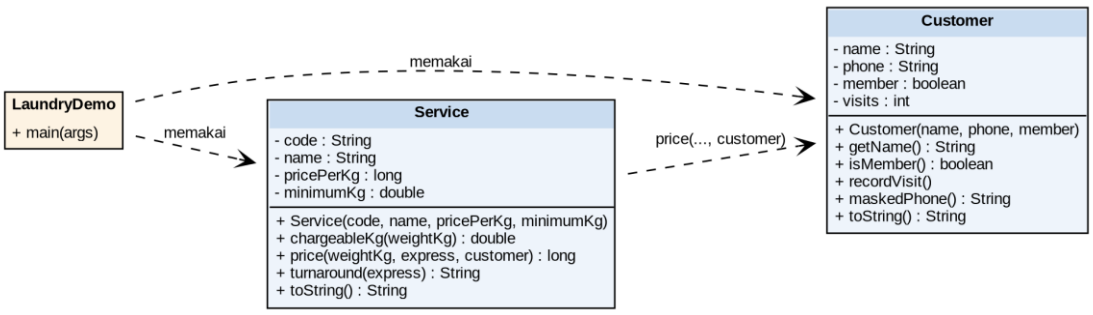
Aturan-aturan ini khas logika bisnis: masing-masing sederhana, tetapi rumit bila digabungkan, dan mudah keliru secara halus.

3.12.1 Menemukan kelas-kelasnya

Kata-kata benda dalam uraian itu mengisyaratkan kelas-kelasnya: *pelanggan*, *layanan*, *kantong*, *nota*. Dalam bab ini kita membangun dua yang pertama; pesanan dan nota mendapat kelasnya sendiri setelah kita mengenal koleksi (Bab 5).

`Customer` (pelanggan) memiliki nama, nomor telepon, penanda keanggotaan, dan hitungan kunjungan. Nama dan nomor telepon tidak pernah berubah setelah pendaftaran, sehingga keduanya mendapat *getter* tetapi tidak mendapat *setter*, dan di Java keduanya ditandai `final`, yang membuat kompilator menolak penugasan kedua. Keanggotaan dapat berubah, sehingga ia memiliki *setter*. Hitungan kunjungan hanya berubah satu demi satu, sehingga alih-alih *setter* tersedia metode `recordVisit()`. Ada juga `maskedPhone()`, yang hanya menampilkan tiga digit terakhir; Subbab 3.13 menjelaskan alasannya.

`Service` (layanan) memiliki kode, nama, harga per kilogram, dan berat minimum. Ia tahu cara menghitung berat yang ditagihkan dan harga sebuah kantong. **Gambar 3.19** memperlihatkan kedua kelas itu dan program peragaan yang memakainya.



Gambar 3.19 Dua kelas pertama Profitina Laundry dan program yang memakainya.

3.12.2 Aturan harga sebagai rumus

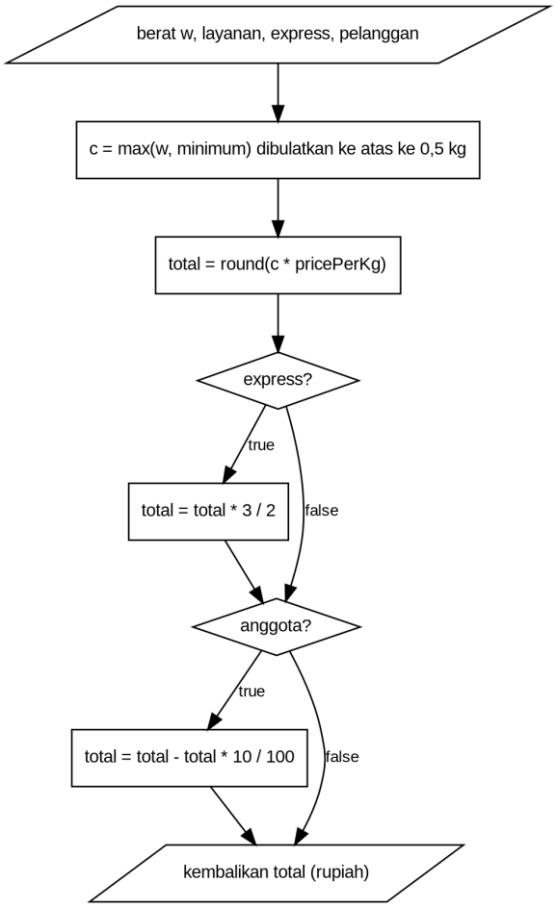
Sebelum menulis kode, tetapkan aturannya dengan tepat. Untuk kantong seberat w kilogram dan layanan dengan batas minimum m kilogram serta harga r rupiah per kilogram, **berat yang ditagihkan** adalah yang lebih besar di antara w dan m, dibulatkan ke atas ke setengah kilogram berikutnya:

$$c = \frac{\lceil 2 \cdot \max(w, m) \rceil}{2} \tag{3.9}$$

Kurung $\lceil \cdot \rceil$ berarti "bulatkan ke atas menjadi bilangan bulat" (fungsi **atap**, *ceiling*). Menggandakan dulu lalu membagi dua sesudahnya mengubah "bulatkan ke atas ke setengah" menjadi "bulatkan ke atas ke bilangan bulat": untuk $w = 4.2$, $2w = 8.4$, dibulatkan ke atas menjadi 9, dibagi dua menjadi 4,5. Harga dasarnya lalu $c \cdot r$. Layanan ekspres mengalikannya dengan $\frac{3}{2}$, dan keanggotaan mengurangkan sepersepuluhnya, dengan urutan itu:

$$p = \begin{cases} c \cdot r & \text{reguler} \\ \frac{3}{2}\, c \cdot r & \text{ekspres} \end{cases}, \qquad \text{harga} = \begin{cases} p & \text{tamu} \\ p - \lfloor p/10 \rfloor & \text{anggota} \end{cases} \tag{3.10}$$

Kurung lantai $\lfloor \cdot \rfloor$ dalam Persamaan (3.10) berarti "bulatkan ke bawah", persis yang dilakukan pembagian bilangan bulat. Karena semua harga berupa rupiah bulat yang disimpan dalam tipe bilangan bulat, sebagaimana dianjurkan di Subbab 3.7.3, setiap langkah sesudah Persamaan (3.9) adalah aritmetika bilangan bulat yang eksak. **Gambar 3.20** memperlihatkan perhitungan itu sebagai diagram alir.



Gambar 3.20 Perhitungan harga dalam `Service.price`: berat yang ditagihkan, harga dasar, lalu dua penyesuaian yang bersifat pilihan.

3.12.3 Kodenya

Kode 3.53 dan **Kode 3.54** adalah kelas-kelas Java-nya.

Kode 3.53 `Customer.java`: pelanggan laundry dengan nomor telepon tersamar dan penghitung kunjungan.

```
public class Customer {
    private final String name;
    private final String phone;
    private boolean member;
    private int visits;

    public Customer(String name, String phone, boolean member) {
        this.name = name;
        this.phone = phone;
        this.member = member;
        this.visits = 0;
    }

    public String getName() {
        return name;
    }

    public String getPhone() {
        return phone;
    }

    public boolean isMember() {
        return member;
    }

    public void setMember(boolean member) {
        this.member = member;
    }

    public int getVisits() {
        return visits;
    }

    public void recordVisit() {
        visits++;
    }

    /** Hanya menampilkan tiga digit terakhir nomor telepon. */
    public String maskedPhone() {
        int keep = 3;
        if (phone.length() <= keep) {
            return phone;
        }
        return "*".repeat(phone.length() - keep) + phone.substring(phone.length() - keep);
    }
```

Kode 3.53 (Lanjutan) *Customer.java: pelanggan laundry dengan nomor telepon tersamar dan penghitung kunjungan.*

```
    @Override
    public String toString() {
        String kind = member ? "member" : "guest";
        return name + " (" + kind + ", " + maskedPhone() + ", " + visits + " visit(s))";
    }
}
```

Kode 3.54 `Service.java`: layanan laundry yang menghitung berat yang ditagihkan, harga, dan lama pengerjaan.

```
public class Service {
    private final String code;
    private final String name;
    private final long pricePerKg;     // rupiah bulat
    private final double minimumKg;

    public Service(String code, String name, long pricePerKg, double minimumKg) {
        this.code = code;
        this.name = name;
        this.pricePerKg = pricePerKg;
        this.minimumKg = minimumKg;
    }

    public String getCode() {
        return code;
    }

    public String getName() {
        return name;
    }

    public long getPricePerKg() {
        return pricePerKg;
    }

    public double getMinimumKg() {
        return minimumKg;
    }

    /** Berat yang ditagihkan: minimal sebesar batas minimum, dibulatkan ke atas ke 0,5 kg.
*/
    public double chargeableKg(double weightKg) {
        double kg = Math.max(weightKg, minimumKg);
        return Math.ceil(kg * 2) / 2;
    }

    /** Harga dalam rupiah untuk satu kantong laundry. */
    public long price(double weightKg, boolean express, Customer customer) {
        long total = Math.round(chargeableKg(weightKg) * pricePerKg);
        if (express) {
            total = total * 3 / 2;            // layanan express 50% lebih mahal
        }
        if (customer.isMember()) {
            total = total - total * 10 / 100; // anggota mendapat potongan 10%
        }
        return total;
    }
```

Kode 3.54 (Lanjutan) *Service.java: layanan laundry yang menghitung berat yang ditagihkan, harga, dan lama pengerjaan.*

```
    public String turnaround(boolean express) {
        if (express) {
            return "same day";
        }
        return switch (code) {
            case "WF" -> "2 days";
            case "IR" -> "1 day";
            case "WDI" -> "3 days";
            default -> "ask at the counter";
        };
    }

    @Override
    public String toString() {
        return String.format("%-4s %-18s Rp%,6d/kg (min %.1f kg)", code, name, pricePerKg,
minimumKg);
    }
}
```

Setiap gagasan dalam bab ini muncul dalam kedua kelas itu: *field* privat, konstruktor yang memakai `this`, *getter*, dan *setter* hanya untuk *field* yang boleh berubah. `maskedPhone` memakai `length`, `substring`, dan `repeat` di balik sebuah `if`. `chargeableKg` menerapkan Persamaan (3.9) dengan `Math.max` dan `Math.ceil`. `price` menerima sebuah `Customer` dan bertanya kepadanya `customer.isMember()`, begitulah objek-objek bekerja sama, lalu menerapkan Persamaan (3.10) dalam aritmetika `long` yang eksak, mengalikan sebelum membagi dalam `total * 3 / 2` agar tidak ada pecahan yang hilang. `turnaround` memakai `return` dini dan `switch` bentuk panah.

Program peragaan, **Kode 3.55**, mendaftarkan dua pelanggan, memproses empat kantong dalam sebuah perulangan, dan mencetak laporan kecil. Larik, yaitu daftar nilai berbentuk `{...}`, dibahas dengan semestinya di Bab 5; di sini kita memakainya hanya sebagai empat kolom sejajar, dengan posisi `i` pada setiap larik menggambarkan kantong nomor `i`.

Kode 3.55 `LaundryDemo.java`: menghitung harga empat kantong cucian untuk dua pelanggan.

```java
public class LaundryDemo {

    public static void main(String[] args) {
        Service washFold = new Service("WF", "Wash & Fold", 7000, 3.0);
        Service ironOnly = new Service("IR", "Iron Only", 5000, 2.0);
        Service full = new Service("WDI", "Wash, Dry & Iron", 10000, 3.0);

        System.out.println("PROFITINA LAUNDRY - price list");
        System.out.println(washFold);
        System.out.println(ironOnly);
        System.out.println(full);
        System.out.println();

        Customer siti = new Customer("Siti Aminah", "081234567890", true);
        Customer budi = new Customer("Budi Santoso", "085712340000", false);

        Customer[] who = {siti, budi, budi, siti};
        Service[] what = {washFold, full, ironOnly, washFold};
        double[] kg = {4.2, 2.0, 5.75, 12.0};
        boolean[] express = {false, true, false, false};

        long grandTotal = 0;
        for (int i = 0; i < who.length; i++) {
            who[i].recordVisit();
            long price = what[i].price(kg[i], express[i], who[i]);
            grandTotal += price;
            System.out.printf("%-13s %-18s %5.2f kg -> %4.1f kg  Rp%,8d  %s%n",
                    who[i].getName(), what[i].getName(), kg[i],
                    what[i].chargeableKg(kg[i]), price, what[i].turnaround(express[i]));
        }
        System.out.printf("Total for the day: Rp%,d%n", grandTotal);
        System.out.println(siti);
        System.out.println(budi);
    }
}
```

```
javac Customer.java Service.java LaundryDemo.java
java LaundryDemo
```

```
PROFITINA LAUNDRY - price list
WF   Wash & Fold        Rp 7,000/kg (min 3.0 kg)
IR   Iron Only          Rp 5,000/kg (min 2.0 kg)
WDI  Wash, Dry & Iron   Rp10,000/kg (min 3.0 kg)

Siti Aminah   Wash & Fold         4.20 kg ->  4.5 kg  Rp  28,350  2 days
Budi Santoso  Wash, Dry & Iron    2.00 kg ->  3.0 kg  Rp  45,000  same day
Budi Santoso  Iron Only           5.75 kg ->  6.0 kg  Rp  30,000  1 day
Siti Aminah   Wash & Fold        12.00 kg -> 12.0 kg  Rp  75,600  2 days
Total for the day: Rp178,950
Siti Aminah (member, *********890, 2 visit(s))
Budi Santoso (guest, *********000, 2 visit(s))
```

Kelas-kelas Python mengikuti rancangan yang sama baris demi baris, dalam **Kode 3.56**, **Kode 3.57**, dan **Kode 3.58**.

Kode 3.56 `customer.py`: kelas `Customer` dalam Python.

```python
class Customer:
    """Pelanggan laundry: nama, nomor telepon, keanggotaan, dan jumlah kunjungan."""

    def __init__(self, name, phone, member):
        self._name = name
        self._phone = phone
        self._member = member
        self._visits = 0

    def get_name(self):
        return self._name

    def get_phone(self):
        return self._phone

    def is_member(self):
        return self._member

    def set_member(self, member):
        self._member = member

    def get_visits(self):
        return self._visits

    def record_visit(self):
        self._visits += 1

    def masked_phone(self):
        """Hanya menampilkan tiga digit terakhir nomor telepon."""
        keep = 3
        if len(self._phone) <= keep:
            return self._phone
        return "*" * (len(self._phone) - keep) + self._phone[-keep:]

    def __str__(self):
        kind = "member" if self._member else "guest"
        return (self._name + " (" + kind + ", " + self.masked_phone() + ", "
                + str(self._visits) + " visit(s))")
```

Kode 3.57 `service.py`: kelas `Service` dalam Python.

```
import math


class Service:
    """Layanan laundry dengan harga per kilogram dan berat minimum."""

    def __init__(self, code, name, price_per_kg, minimum_kg):
        self._code = code
        self._name = name
        self._price_per_kg = price_per_kg  # rupiah bulat
        self._minimum_kg = minimum_kg

    def get_code(self):
        return self._code

    def get_name(self):
        return self._name

    def get_price_per_kg(self):
        return self._price_per_kg

    def get_minimum_kg(self):
        return self._minimum_kg

    def chargeable_kg(self, weight_kg):
        """Berat yang ditagihkan: minimal sebesar batas minimum, dibulatkan ke atas ke 0,5
kg."""
        kg = max(weight_kg, self._minimum_kg)
        return math.ceil(kg * 2) / 2

    def price(self, weight_kg, express, customer):
        """Harga dalam rupiah untuk satu kantong laundry."""
        total = int(self.chargeable_kg(weight_kg) * self._price_per_kg)
        if express:
            total = total * 3 // 2  # layanan express 50% lebih mahal
        if customer.is_member():
            total = total - total * 10 // 100  # anggota mendapat potongan 10%
        return total
```

Kode 3.57 (Lanjutan) *service.py: kelas Service dalam Python.*

```
    def turnaround(self, express):
        if express:
            return "same day"
        match self._code:
            case "WF":
                return "2 days"
            case "IR":
                return "1 day"
            case "WDI":
                return "3 days"
            case _:
                return "ask at the counter"

    def __str__(self):
        return (f"{self._code:<4} {self._name:<18} Rp{self._price_per_kg:6,d}/kg "
                f"(min {self._minimum_kg:.1f} kg)")
```

Kode 3.58 `laundry_demo.py`: program peragaan dalam Python.

```
from customer import Customer
from service import Service


def main():
    wash_fold = Service("WF", "Wash & Fold", 7000, 3.0)
    iron_only = Service("IR", "Iron Only", 5000, 2.0)
    full = Service("WDI", "Wash, Dry & Iron", 10000, 3.0)

    print("PROFITINA LAUNDRY - price list")
    print(wash_fold)
    print(iron_only)
    print(full)
    print()

    siti = Customer("Siti Aminah", "081234567890", True)
    budi = Customer("Budi Santoso", "085712340000", False)

    who = [siti, budi, budi, siti]
    what = [wash_fold, full, iron_only, wash_fold]
    kg = [4.2, 2.0, 5.75, 12.0]
    express = [False, True, False, False]

    grand_total = 0
    for i in range(len(who)):
        who[i].record_visit()
        price = what[i].price(kg[i], express[i], who[i])
        grand_total += price
        print(f"{who[i].get_name():<13} {what[i].get_name():<18} {kg[i]:5.2f} kg -> "
              f"{what[i].chargeable_kg(kg[i]):4.1f} kg  Rp{price:8,d}  "
              f"{what[i].turnaround(express[i])}")
    print(f"Total for the day: Rp{grand_total:,d}")
    print(siti)
    print(budi)


if __name__ == "__main__":
    main()
```

```
PROFITINA LAUNDRY - price list
WF   Wash & Fold        Rp 7,000/kg (min 3.0 kg)
IR   Iron Only          Rp 5,000/kg (min 2.0 kg)
WDI  Wash, Dry & Iron   Rp10,000/kg (min 3.0 kg)

Siti Aminah   Wash & Fold         4.20 kg ->  4.5 kg  Rp  28,350  2 days
Budi Santoso  Wash, Dry & Iron    2.00 kg ->  3.0 kg  Rp  45,000  same day
Budi Santoso  Iron Only           5.75 kg ->  6.0 kg  Rp  30,000  1 day
Siti Aminah   Wash & Fold        12.00 kg -> 12.0 kg  Rp  75,600  2 days
Total for the day: Rp178,950
Siti Aminah (member, *********890, 2 visit(s))
Budi Santoso (guest, *********000, 2 visit(s))
```

Kedua keluaran itu identik, karakter demi karakter. Beberapa pilihan kecil dalam penerjemahan memungkinkan hal itu: `math.ceil` untuk `Math.ceil`, `//` di setiap tempat `/` Java membagi bilangan bulat, dan `int(...)` alih-alih `round` milik Python, yang membulatkan setengah ke bilangan genap. Karena c adalah kelipatan 0.5 dan setiap harga per kilogram adalah kelipatan 1000, hasil kali $c \cdot r$ selalu bilangan bulat, sehingga `int` tidak menghilangkan apa pun di sini.

3.12.4 Memeriksa harga dengan tangan

Perhitungan harga yang meleset beberapa ribu rupiah pada setiap kantong adalah bencana bagi usaha kecil, sehingga kita memeriksa setiap baris keluaran dengan sebuah tabel penelusuran, **Tabel 3.14**.

Tabel 3.14 Memeriksa keempat harga pada **Kode 3.55** dengan Persamaan (3.9) dan Persamaan (3.10).

Kantong	Layanan, r, m	w	c Menurut Persamaan (3.9)	$c \cdot r$	Ekspres	Anggota	Harga
1 Siti	WF, 7000, 3.0	4.20	$\lceil 8.4 \rceil / 2 = 4.5$	31 500	tidak	−3 150	28 350
2 Budi	WDI, 10000, 3.0	2.00	$\lceil 6 \rceil / 2 = 3.0$	30 000	$\times \frac{3}{2} = 45\,000$	tidak	45 000
3 Budi	IR, 5000, 2.0	5.75	$\lceil 11.5 \rceil / 2 = 6.0$	30 000	tidak	tidak	30 000
4 Siti	WF, 7000, 3.0	12.00	$\lceil 24 \rceil / 2 = 12.0$	84 000	tidak	−8 400	75 600

Kantong 2 memperlihatkan batas minimum yang bekerja: 2 kg milik Budi ditagih sebagai 3 kg. Kantong 3 memperlihatkan pembulatannya: 5,75 kg ditagih sebagai 6 kg. Keempat harga berjumlah $28\,350 + 45\,000 + 30\,000 + 75\,600 = 178\,950$, total hari itu pada keluaran. Dua baris terakhir memperlihatkan keadaan objek-objek sesudah perulangan: setiap pelanggan berkunjung dua kali, dan nomor-nomor teleponnya tersamar.

3.13 Profitina dalam Praktik

Profitina, platform Visibilitas AI di app.profitina.com, mengukur seberapa siap situs web sebuah merek untuk dikutip oleh mesin penjawab AI, lalu membantu memperbaiki apa yang ditemukannya. Servernya adalah aplikasi web Python, dan gagasan-gagasan bab ini tampak dalam strukturnya.

Kelas untuk benda-benda dalam domainnya. Sebagaimana laundry memiliki `Customer` dan `Service`, pemindai Profitina memodelkan setiap jenis sinyal yang diperiksanya (seberapa mudah isi sebuah halaman diekstraksi, apakah data terstrukturnya sah, seberapa sehat situs itu secara teknis) sebagai kelas tersendiri yang memberi skor dan menjelaskan dirinya sendiri. Bab 6 memperlihatkan antarmuka bersama yang diterapkan kelas-kelas itu.

String di mana-mana. Halaman web adalah teks. Memutuskan apakah sebuah halaman menyebut suatu merek, memeriksa berkas `robots.txt`, membaca blok schema.org: semuanya pekerjaan string dengan metode-metode dari Subbab 3.5, dan semuanya membandingkan *isi*, tidak pernah identitas, sesuai pelajaran dari **Tabel 3.3**.

Bilangan dengan hati-hati. Skor berupa desimal, dibandingkan dengan ambang batas memakai toleransi Persamaan (3.3), tidak pernah dengan `==`. Jumlah koin utilitas PROFIT, yang memiliki enam angka desimal, ditangani sebagai bilangan bulat dalam satuan terkecilnya, seperti rupiah di laundry kita.

Data pribadi, ditangani dengan sengaja. Metode `maskedPhone` di laundry bukan hiasan. Undang-Undang Pelindungan Data Pribadi, UU PDP No. 27 Tahun 2022 [12], mensyaratkan agar data pribadi diproses hanya sejauh diperlukan untuk tujuannya. Nota atau layar yang menampilkan `*********890` tetap memungkinkan petugas mengenali pelanggannya, tanpa membeberkan nomor lengkapnya kepada semua orang di depan meja layanan. Profitina mengikuti prinsip yang sama dalam fitur-fitur kepatuhannya sendiri, dengan persetujuan eksplisit, ekspor data, dan penghapusan akun secara menyeluruh.

Pengujian sebagai gerbang. Proses rilis Profitina menjalankan jauh lebih dari seribu pengujian otomatis yang bentuknya sama dengan `DogCheck`, nilai harapan melawan nilai sebenarnya, dan sebuah rilis baru keluar bila semuanya lulus. Platform memberi saran; penggunanya yang memutuskan.

3.14 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya

Lupa `this` di dalam konstruktor. `name = name;` menugaskan parameter ke dirinya sendiri. Tulislah `this.name = name;`; di Python, `self.name = name`.

Membandingkan string dengan `==` di Java. Operator itu membandingkan panah, bukan karakter. Pakailah `equals` (Subbab 3.5.3); di Python, `==` yang benar dan `is` yang salah.

Mengira metode string mengubah string-nya. `name.trim();` sendirian membuang hasilnya. Tulislah `name = name.trim();`.

Mengira penugasan menyalin objek. `Dog b = a;` membuat alias. Gambarlah diagram objeknya (Subbab 3.4.3).

Memanggil metode melalui `null` atau `None`. Periksalah dulu, atau pastikan variabel itu selalu diberi objek.

Luapan bilangan bulat di tengah perhitungan. Konversikan ke `long` *sebelum* perkalian yang berisiko, `(long) x * y`, bukan sesudahnya.

Membandingkan desimal dengan `==`. Pakailah toleransi, Persamaan (3.3), atau `math.isclose` di Python.

Pembagian bilangan bulat yang tidak disengaja. `1 / 16` bernilai 0 di Java: tulislah `1 / 16.0` atau `(double) x / y`, dan ingatlah bahwa bilangan negatif dibagi secara berbeda dalam kedua bahasa (Subbab 3.7.4).

Menyimpan uang dalam `double`. Pakailah satuan bulat dalam `long`, atau `BigDecimal`/`Decimal`.

`break` yang terlupa dalam `switch` klasik. Eksekusi jatuh terus ke *case* berikutnya. Utamakan bentuk panah di Java, atau `match` di Python.

Rantai `else if` dengan urutan yang salah. Kondisi benar yang pertama menang; letakkan kondisi yang paling khusus di depan.

Perulangan `while` yang tak pernah berakhir. Sesuatu di dalam badannya harus menggerakkan kondisi menuju salah. Periksalah sebelum menjalankan program.

Galat selisih satu. `<` lawan `<=`, mulai dari 0 atau 1, ujung `range` Python yang tidak disertakan: telusuri iterasi pertama dan terakhir dengan tangan.

Memulai dari `args[0]` di Python. `sys.argv[0]` adalah nama skrip; kata pertama adalah `sys.argv[1]`.

Komentar yang mengulang kode. Jelaskan mengapa, bukan apa.

3.15 Ringkasan Bab

- **Kelas** adalah cetak biru; **objek** (**instans**) adalah satu benda yang dibangun darinya, dengan salinan sendiri dari setiap ***field***. **Konstruktor** membangun objek, **metode** memberinya perilaku, dan `this` (Java) atau `self` (Python) berarti "objek saat ini". Objek dibuat dengan `new` di Java dan dengan memanggil kelasnya di Python; *field* privat dijangkau melalui ***getter*** dan ***setter***, dan `toString` atau `__str__` memberikan bentuk teksnya.
- Variabel Java bertipe **primitif** menyimpan nilai; variabel bertipe **referensi** menyimpan panah ke objek. Di Python setiap variabel adalah nama yang terikat ke sebuah objek. Menyalin

referensi menghasilkan **alias**. Identitas ditulis `==` di Java dan `is` di Python; kesetaraan isi ditulis `equals` di Java dan `==` di Python. **Diagram objek** membuat semua ini kasatmata.

- **String** bersifat imutabel, berindeks mulai dari 0, dan kaya metode (serta irisan di Python). Formatlah keluaran dengan `printf`/`String.format` atau f-string; uraikan teks dengan `Integer.parseInt`, `Double.parseDouble`, `int`, dan `float`.
- Java **bertipe statis**; Python **bertipe dinamis** tetapi **kuat**.
- Tiga jebakan aritmetika: **luapan** (Persamaan (3.2)), **pembulatan titik mengambang** (bandingkan dengan toleransi, Persamaan (3.3)), dan **pembagian bilangan bulat** (`1 / 16` bernilai 0 di Java; Python memisahkan `/` dan `//`).
- **Percabangan**: `if`/`else if`/`else` (`elif`) yang diuji dari atas; `switch` (dengan `break`, atau bentuk panah) dan `match` untuk satu variabel terhadap konstanta-konstanta. **Perulangan**: `for` untuk jumlah yang diketahui, `while` sampai sesuatu terjadi, `do-while` untuk bertindak sebelum memeriksa, dan for-each untuk mengunjungi setiap elemen. **Tabel penelusuran** memperlihatkan persis apa yang dilakukan sebuah perulangan.
- Profitina Laundry kini memiliki `Customer` dan `Service`, yang menghitung harga sebuah kantong secara eksak, dalam rupiah bulat, menurut Persamaan (3.9) dan Persamaan (3.10).

Bab 4 membahas soal-soal dengan jenis yang persis seperti ini, dalam kedua bahasa, dan Bab 5 menambahkan larik, *list*, masukan dari papan ketik, eksepsi, dan berkas, sehingga laundry kita akhirnya dapat menyimpan lebih dari segelintir pesanan.

3.16 Soal Latihan

Selesaikan setiap soal dalam Java dan Python. Untuk soal pemrograman, tebaklah keluarannya dengan tangan lebih dulu, lalu jalankan program Anda dan bandingkan.

Soal Latihan 3.1 (anatomi). Dalam `Customer` (**Kode 3.53**), sebutkan *field*, konstruktor, dan metode-metodenya. *Field* mana yang tidak dapat berubah setelah konstruksi, dan bagaimana setiap bahasa menyatakannya? Apa yang salah seandainya konstruktor berbunyi `name = name;` alih-alih `this.name = name;`?

Soal Latihan 3.2 (referensi). Tebaklah keluarannya, lalu gambarlah diagram objek sesudah baris terakhir.

```
Dog a = new Dog("Rex", "Beagle", 2, 10.0);
Dog b = new Dog("Rex", "Beagle", 2, 10.0);
Dog c = a;
c.setAge(3);
b = c;
System.out.println(a.getAge() + " " + b.getAge() + " " + (a == b));
```

Soal Latihan 3.3 (kelas baru). Tulislah kelas `WashingMachine` (mesin cuci) dengan sebuah pengenal, kapasitas dalam kilogram, penanda sibuk, dan muatan saat ini. Metode `load(kg)` memulai pencucian dan mengembalikan `true` hanya bila mesinnya kosong dan `0 < kg <= capacity`; selain itu ia mengembalikan `false` dan tidak mengubah apa pun. `unload()` mengosongkan mesin. Berilah `toString`, dan sebuah `main` yang mencoba muatan 9,5, 6,0, dan 2,0 kg pada mesin berkapasitas 8 kg.

Soal Latihan 3.4 (string). Tulislah program yang menerima nama lengkap seperti `"  siti nur aminah "` lalu mencetak banyaknya kata, inisial dalam huruf kapital (`S.N.A.`), dan label kantong berupa nama belakang dengan huruf pertama kapital diikuti inisial pertama (`Aminah, S.`). Spasi berlebih tidak boleh berpengaruh.

Soal Latihan 3.5 (ungkapan). Tentukan nilai dan tipe setiap ungkapan Java berikut, serta nilai padanannya di Python (`/` menjadi `//` bila Java membagi bilangan bulat): `7 / 2`, `7 / 2.0`, `7 % 3`, `(double) 7 / 2`, `(int) 7.9`, `1000000 * 3000`, `'A' + 2`, `"1" + 2 + 3`, `1 + 2 + "3"`, `-7 / 2`.

Soal Latihan 3.6 (penelusuran). Telusurilah perulangan ini dengan tabel penelusuran dan nyatakan apa yang dihitungnya.

```
int x = 48, y = 18;
while (y != 0) {
    int t = x % y;
    x = y;
    y = t;
}
System.out.println(x);
```

Soal Latihan 3.7 (perulangan bersarang). Cetaklah tabel perkalian untuk 1 sampai 5, dengan baris judul dan kolom selebar 4 karakter.

Soal Latihan 3.8 (perulangan `while`). Berapa tahun penuh yang diperlukan agar GBP 100 dari Subbab 3.11.3 mencapai setidaknya GBP 200 dengan bunga 2,3%? Bandingkan jawaban Anda dengan kaidah praktis yang dikenal sebagai *aturan 72* (*rule of 72*),

$$n \approx \frac{72}{\text{suku bunga}} \tag{3.11}$$

Soal Latihan 3.9 (memakai kelas-kelas laundry). Dengan memakai `Service` dan `Customer` tanpa mengubahnya, cetaklah tabel harga *Wash & Fold* untuk 1 sampai 6 kg, dengan empat kolom: tamu, tamu ekspres, anggota, anggota ekspres.

Soal Latihan 3.10 (pembagian dan sisa bagi). Seorang pelanggan membayar tagihan dengan uang rupiah. Diberikan sebuah jumlah dalam rupiah, cetaklah berapa lembar atau keping setiap pecahan yang dipakai, dimulai dari yang terbesar, dengan nilai 100000, 50000, 20000, 10000, 5000, 2000, 1000, 500, 200, 100, dan 50. Ujilah dengan Rp178.950, total hari itu dari **Kode 3.55**.

3.17 Penyelesaian Soal Latihan

Penyelesaian 3.1. *Field*: `name`, `phone`, `member`, `visits`. Konstruktor: `Customer(String name, String phone, boolean member)`, yang juga mengisi `visits` dengan 0. Metode: `getName`, `getPhone`, `isMember`, `setMember`, `getVisits`, `recordVisit`, `maskedPhone`, `toString`. *Field* `name` dan `phone` tidak pernah berubah: Java menandainya `final`, sehingga kompilator menolak penugasan kedua; Python tidak memiliki kata kunci semacam itu, sehingga kelasnya tidak menyediakan *setter* dan garis bawah meminta orang lain tidak menyentuh *field* itu. Dengan `name = name;`, parameter ditugaskan ke dirinya sendiri dan *field*-nya tidak pernah diisi. Karena *field* itu `final`, Java lalu menolak mengompilasi konstruktornya (“variable name might not have been initialized”); tanpa `final`, program akan terkompilasi dan diam-diam mencetak `null` sebagai nama.

Penyelesaian 3.2. Keluarannya `3 3 true`. Sesudah dua baris pertama ada dua objek terpisah. `c = a` menjadikan `c` alias anjing pertama, sehingga `c.setAge(3)` mengubah anjing pertama. `b = c` memindahkan panah `b` ke anjing pertama pula, sehingga ketiga variabel menunjuk ke satu objek berumur 3, dan `a == b` bernilai `true`. Anjing kedua, yang dibuat pada baris 2, kini tidak ditunjuk panah apa pun: dalam diagram objek ia dicoret sebagai sampah.

Penyelesaian 3.3. Penjaga dalam `load` menolak mesin yang sibuk, muatan yang tidak positif, dan muatan berlebih dalam satu kondisi yang digabung dengan `||`, dan ia kembali *sebelum* mengubah

field apa pun, sehingga muatan yang ditolak membiarkan mesin tak tersentuh. **Kode 3.59** dan **Kode 3.60** memberikan kelasnya. Metode `main` diletakkan di dalam kelas itu, yang praktis untuk latihan kecil.

Kode 3.59 `WashingMachine.java`: penyelesaian Soal Latihan 3.3.

```
public class WashingMachine {
    private final String id;
    private final double capacityKg;
    private boolean busy;
    private double loadKg;

    public WashingMachine(String id, double capacityKg) {
        this.id = id;
        this.capacityKg = capacityKg;
        this.busy = false;
        this.loadKg = 0.0;
    }

    /** Memulai pencucian bila mesin kosong dan muatannya muat. */
    public boolean load(double kg) {
        if (busy || kg <= 0 || kg > capacityKg) {
            return false;
        }
        busy = true;
        loadKg = kg;
        return true;
    }

    public void unload() {
        busy = false;
        loadKg = 0.0;
    }

    public boolean isBusy() {
        return busy;
    }

    @Override
    public String toString() {
        String state = busy ? "busy with " + loadKg + " kg" : "free";
        return "Machine " + id + " (" + capacityKg + " kg): " + state;
    }
```

Kode 3.59 (Lanjutan) *WashingMachine.java: penyelesaian Soal Latihan 3.3.*

```
    public static void main(String[] args) {
        WashingMachine m1 = new WashingMachine("M1", 8.0);
        System.out.println(m1);
        System.out.println("load 9.5 kg -> " + m1.load(9.5));
        System.out.println("load 6.0 kg -> " + m1.load(6.0));
        System.out.println("load 2.0 kg -> " + m1.load(2.0));
        System.out.println(m1);
        m1.unload();
        System.out.println(m1);
    }
}
```

```
Machine M1 (8.0 kg): free
load 9.5 kg -> false
load 6.0 kg -> true
load 2.0 kg -> false
Machine M1 (8.0 kg): busy with 6.0 kg
Machine M1 (8.0 kg): free
```

Kode 3.60 `washing_machine.py`: penyelesaian Soal Latihan 3.3 dalam Python.

```
class WashingMachine:
    def __init__(self, machine_id, capacity_kg):
        self._id = machine_id
        self._capacity_kg = capacity_kg
        self._busy = False
        self._load_kg = 0.0

    def load(self, kg):
        """Memulai pencucian bila mesin kosong dan muatannya muat."""
        if self._busy or kg <= 0 or kg > self._capacity_kg:
            return False
        self._busy = True
        self._load_kg = kg
        return True

    def unload(self):
        self._busy = False
        self._load_kg = 0.0

    def is_busy(self):
        return self._busy

    def __str__(self):
        state = "busy with " + str(self._load_kg) + " kg" if self._busy else "free"
        return "Machine " + self._id + " (" + str(self._capacity_kg) + " kg): " + state


def main():
    m1 = WashingMachine("M1", 8.0)
    print(m1)
    print("load 9.5 kg -> " + str(m1.load(9.5)))
    print("load 6.0 kg -> " + str(m1.load(6.0)))
    print("load 2.0 kg -> " + str(m1.load(2.0)))
    print(m1)
    m1.unload()
    print(m1)


if __name__ == "__main__":
    main()
```

```
Machine M1 (8.0 kg): free
load 9.5 kg -> False
load 6.0 kg -> True
load 2.0 kg -> False
Machine M1 (8.0 kg): busy with 6.0 kg
Machine M1 (8.0 kg): free
```

Muatan pertama ditolak karena 9,5 > 8,0; muatan kedua diterima; muatan ketiga ditolak karena mesinnya sibuk.

Penyelesaian 3.4. Kuncinya adalah memecah pada *satu spasi atau lebih* sesudah memangkas: `split(" +")` di Java memakai pola `" +"`, yang berarti "satu spasi atau lebih", dan `split()` tanpa argumen di Python melakukan hal yang sama secara otomatis. Setiap inisial adalah `charAt(0)` (di Python `part[0]`) yang diubah menjadi huruf kapital, seperti dalam **Kode 3.61** dan **Kode 3.62**.

Kode 3.61 `Initials.java`: penyelesaian Soal Latihan 3.4.

```java
public class Initials {

    public static void main(String[] args) {
        String fullName = args.length > 0 ? args[0] : "  siti nur aminah ";
        String[] parts = fullName.trim().split(" +");   // satu spasi atau lebih
        String initials = "";
        for (String part : parts) {
            initials = initials + Character.toUpperCase(part.charAt(0)) + ".";
        }
        System.out.println("Name     : [" + fullName.trim() + "]");
        System.out.println("Words    : " + parts.length);
        System.out.println("Initials : " + initials);
        String last = parts[parts.length - 1];
        String tag = Character.toUpperCase(last.charAt(0)) +
last.substring(1).toLowerCase();
        System.out.println("Bag tag  : " + tag + ", " + initials.substring(0, 2));
    }
}
```

```
Name     : [siti nur aminah]
Words    : 3
Initials : S.N.A.
Bag tag  : Aminah, S.
```

Kode 3.62 `initials.py`: penyelesaian Soal Latihan 3.4 dalam Python.

```python
import sys


def main():
    full_name = sys.argv[1] if len(sys.argv) > 1 else "  siti nur aminah "
    parts = full_name.split()  # split() tanpa argumen melewati spasi berlebih
    initials = ""
    for part in parts:
        initials = initials + part[0].upper() + "."
    print("Name     : [" + full_name.strip() + "]")
    print("Words    : " + str(len(parts)))
    print("Initials : " + initials)
    last = parts[-1]
    tag = last[0].upper() + last[1:].lower()
    print("Bag tag  : " + tag + ", " + initials[0:2])


if __name__ == "__main__":
    main()
```

```
Name     : [siti nur aminah]
Words    : 3
Initials : S.N.A.
Bag tag  : Aminah, S.
```

Penyelesaian 3.5. **Tabel 3.15** memberikan jawabannya; setiap jawaban sudah diperiksa dengan menjalankannya.

Tabel 3.15 Nilai ungkapan-ungkapan pada Soal Latihan 3.5.

Ungkapan Java	Nilai Java (tipe)	Padanan Python	Nilai Python
`7 / 2`	`3` (`int`)	`7 // 2`	`3`
`7 / 2.0`	`3.5` (`double`)	`7 / 2.0`	`3.5`
`7 % 3`	`1` (`int`)	`7 % 3`	`1`
`(double) 7 / 2`	`3.5` (`double`)	`7 / 2`	`3.5`
`(int) 7.9`	`7` (`int`)	`int(7.9)`	`7`
`1000000 * 3000`	`-1294967296` (`int`, luapan)	`1000000 * 3000`	`3000000000`
`'A' + 2`	`67` (`int`)	`chr(ord('A') + 2)`	`'C'`
`"1" + 2 + 3`	`"123"` (`String`)	`"1" + str(2) + str(3)`	`'123'`
`1 + 2 + "3"`	`"33"` (`String`)	`str(1 + 2) + "3"`	`'33'`
`-7 / 2`	`-3` (`int`)	`-7 // 2`	`-4`

Dua baris patut diberi catatan. Dalam `"1" + 2 + 3`, Java bekerja dari kiri ke kanan: `"1" + 2` adalah string `"12"`, dan `"12" + 3` adalah `"123"`. Dalam `1 + 2 + "3"`, `+` yang pertama menjumlahkan dua bilangan, menghasilkan 3, dan baru kemudian bertemu string, sehingga hasilnya `"33"`. Adapun `1000000 * 3000` = 3×10^9 melampaui $2^{31} - 1$, sehingga menurut Persamaan (3.2) nilainya berputar menjadi $3 \times 10^9 - 2^{32} = -1\,294\,967\,296$.

Penyelesaian 3.6. **Tabel 3.16** menelusuri perulangan itu.

Tabel 3.16 Menelusuri perulangan Soal Latihan 3.6 untuk $x = 48$, $y = 18$.

Iterasi	x	y	t = x % y	x baru	y baru
1	48	18	12	18	12
2	18	12	6	12	6
3	12	6	0	6	0

Kini `y` bernilai 0, perulangan berakhir, dan program mencetak 6. Inilah algoritma Euklides untuk **faktor persekutuan terbesar** (FPB): 6 adalah bilangan terbesar yang membagi habis 48 dan 18. Algoritma ini benar karena setiap bilangan yang membagi habis `x` dan `y` juga membagi habis `x % y`, sehingga setiap langkah mengganti pasangan bilangan dengan pasangan yang lebih kecil yang faktor persekutuannya sama.

Penyelesaian 3.7. Perulangan luar berjalan atas baris-baris, dan untuk setiap baris perulangan dalam berjalan atas kolom-kolom; perulangan dalam menuntaskan semua iterasinya untuk setiap satu iterasi perulangan luar. Perulangan yang diletakkan di dalam perulangan seperti ini disebut **perulangan bersarang** (*nested loops*); lihat **Kode 3.63** dan **Kode 3.64**.

Kode 3.63 `MultiplicationTable.java`: penyelesaian Soal Latihan 3.7.

```java
public class MultiplicationTable {

    public static void main(String[] args) {
        int n = 5;
        System.out.print("   |");
        for (int col = 1; col <= n; col++) {
            System.out.printf("%4d", col);
        }
        System.out.println();
        System.out.println("---+" + "-".repeat(4 * n));
        for (int row = 1; row <= n; row++) {
            System.out.printf("%2d |", row);
            for (int col = 1; col <= n; col++) {
                System.out.printf("%4d", row * col);
            }
            System.out.println();
        }
    }
}
```

```
   |   1   2   3   4   5
---+--------------------
 1 |   1   2   3   4   5
 2 |   2   4   6   8  10
 3 |   3   6   9  12  15
 4 |   4   8  12  16  20
 5 |   5  10  15  20  25
```

Kode 3.64 `multiplication_table.py`: penyelesaian Soal Latihan 3.7 dalam Python.

```python
def main():
    n = 5
    print("   |", end="")
    for col in range(1, n + 1):
        print(f"{col:4d}", end="")
    print()
    print("---+" + "-" * (4 * n))
    for row in range(1, n + 1):
        print(f"{row:2d} |", end="")
        for col in range(1, n + 1):
            print(f"{row * col:4d}", end="")
        print()


if __name__ == "__main__":
    main()
```

```
   |   1   2   3   4   5
---+--------------------
 1 |   1   2   3   4   5
 2 |   2   4   6   8  10
 3 |   3   6   9  12  15
 4 |   4   8  12  16  20
 5 |   5  10  15  20  25
```

Penyelesaian 3.8. Kita tidak tahu sebelumnya berapa tahun yang diperlukan; kita mengulang *sampai* saldonya mencapai 200. Itulah perulangan `while`, dalam **Kode 3.65** dan **Kode 3.66**.

Kode 3.65 `DoublingYears.java`: penyelesaian Soal Latihan 3.8.

```java
public class DoublingYears {

    public static void main(String[] args) {
        double capital = 100;
        double interestRate = 2.3;
        double balance = capital;
        int years = 0;
        while (balance < 2 * capital) {
            balance = balance * (1 + 0.01 * interestRate);
            years++;
        }
        System.out.printf("Doubled after %d years: GBP %.2f%n", years, balance);
        System.out.printf("Rule of 72 estimate: %.1f years%n", 72 / interestRate);
    }
}
```

```
Doubled after 31 years: GBP 202.37
Rule of 72 estimate: 31.3 years
```

Kode 3.66 `doubling_years.py`: penyelesaian Soal Latihan 3.8 dalam Python.

```python
def main():
    capital = 100.0
    interest_rate = 2.3
    balance = capital
    years = 0
    while balance < 2 * capital:
        balance = balance * (1 + 0.01 * interest_rate)
        years += 1
    print(f"Doubled after {years} years: GBP {balance:.2f}")
    print(f"Rule of 72 estimate: {72 / interest_rate:.1f} years")


if __name__ == "__main__":
    main()
```

```
Doubled after 31 years: GBP 202.37
Rule of 72 estimate: 31.3 years
```

Diperlukan 31 tahun; sesudah 30 tahun saldonya masih sedikit di bawah 200 (sekitar 197,8), itulah sebabnya perulangan berjalan sekali lagi. Persamaan (3.11) memperkirakan 72/2.3 ≈ 31.3 tahun, hampiran yang sangat baik. Jawaban yang sama dapat dibaca dari **Gambar 3.18**, tempat garisnya naik dua kali lipat kira-kira setiap 31 tahun.

Penyelesaian 3.9. Tabel itu memakai ulang kelas-kelas yang ada tanpa mengubahnya, dan itulah inti menulis kelas: aturan harga tinggal di satu tempat, `Service.price`. Penyelesaiannya adalah **Kode 3.67** dan **Kode 3.68**.

Kode 3.67 `PriceTable.java`: penyelesaian Soal Latihan 3.9.

```java
public class PriceTable {

    public static void main(String[] args) {
        Service washFold = new Service("WF", "Wash & Fold", 7000, 3.0);
        Customer guest = new Customer("Walk-in", "0000", false);
        Customer member = new Customer("Member", "0000", true);

        System.out.println(" kg   guest   guest+exp   member   member+exp");
        for (double kg = 1.0; kg <= 6.0; kg += 1.0) {
            System.out.printf("%3.0f %7d %11d %8d %12d%n", kg,
                    washFold.price(kg, false, guest),
                    washFold.price(kg, true, guest),
                    washFold.price(kg, false, member),
                    washFold.price(kg, true, member));
        }
    }
}
```

```
 kg   guest   guest+exp   member   member+exp
  1   21000       31500    18900       28350
  2   21000       31500    18900       28350
  3   21000       31500    18900       28350
  4   28000       42000    25200       37800
  5   35000       52500    31500       47250
  6   42000       63000    37800       56700
```

Kode 3.68 `price_table.py`: penyelesaian Soal Latihan 3.9 dalam Python.

```python
from customer import Customer
from service import Service


def main():
    wash_fold = Service("WF", "Wash & Fold", 7000, 3.0)
    guest = Customer("Walk-in", "0000", False)
    member = Customer("Member", "0000", True)

    print(" kg   guest   guest+exp   member   member+exp")
    for kg in range(1, 7):
        print(f"{kg:3d} {wash_fold.price(kg, False, guest):7d} "
              f"{wash_fold.price(kg, True, guest):11d} "
              f"{wash_fold.price(kg, False, member):8d} "
              f"{wash_fold.price(kg, True, member):12d}")


if __name__ == "__main__":
    main()
```

```
 kg   guest   guest+exp   member   member+exp
  1   21000       31500    18900       28350
  2   21000       31500    18900       28350
  3   21000       31500    18900       28350
  4   28000       42000    25200       37800
  5   35000       52500    31500       47250
  6   42000       63000    37800       56700
```

Tiga baris pertama sama karena batas minimum 3 kg dalam Persamaan (3.9). Perulangan Java mencacah dengan `double`, `kg += 1.0`; cara ini aman di sini karena bilangan bulat diwakili secara tepat dalam `double`, tetapi mencacah dengan langkah 0,1 akan terkena galat dari **Gambar 3.14**, dan versi Python mencacah dengan bilangan bulat saja.

Penyelesaian 3.10. Untuk setiap nilai v, dari yang terbesar, banyaknya lembar adalah hasil bagi bulat `rest / v`, dan sisanya adalah `rest % v`, yaitu Persamaan (3.4) yang dipakai dua kali pada setiap langkah; lihat **Kode 3.69** dan **Kode 3.70**.

Kode 3.69 `BanknoteBreakdown.java`: penyelesaian Soal Latihan 3.10.

```java
public class BanknoteBreakdown {

    public static void main(String[] args) {
        long amount = args.length > 0 ? Long.parseLong(args[0]) : 178950;
        long[] values = {100000, 50000, 20000, 10000, 5000, 2000, 1000, 500, 200, 100, 50};

        System.out.printf("Rp%,d is paid as:%n", amount);
        long rest = amount;
        for (long v : values) {
            long count = rest / v;     // berapa lembar pecahan ini yang muat
            rest = rest % v;           // sisa yang belum terbayar
            if (count > 0) {
                System.out.printf("  %d x Rp%,d%n", count, v);
            }
        }
        if (rest > 0) {
            System.out.println("  cannot pay the last Rp" + rest);
        }
    }
}
```

```
Rp178,950 is paid as:
  1 x Rp100,000
  1 x Rp50,000
  1 x Rp20,000
  1 x Rp5,000
  1 x Rp2,000
  1 x Rp1,000
  1 x Rp500
  2 x Rp200
  1 x Rp50
```

Kode 3.70 `banknote_breakdown.py`: penyelesaian Soal Latihan 3.10 dalam Python.

```python
import sys


def main():
    amount = int(sys.argv[1]) if len(sys.argv) > 1 else 178950
    values = [100000, 50000, 20000, 10000, 5000, 2000, 1000, 500, 200, 100, 50]

    print(f"Rp{amount:,d} is paid as:")
    rest = amount
    for v in values:
        count = rest // v  # berapa lembar pecahan ini yang muat
        rest = rest % v    # sisa yang belum terbayar
        if count > 0:
            print(f"  {count} x Rp{v:,d}")
    if rest > 0:
        print("  cannot pay the last Rp" + str(rest))


if __name__ == "__main__":
    main()
```

```
Rp178,950 is paid as:
  1 x Rp100,000
  1 x Rp50,000
  1 x Rp20,000
  1 x Rp5,000
  1 x Rp2,000
  1 x Rp1,000
  1 x Rp500
  2 x Rp200
  1 x Rp50
```

Pemeriksaan: $100\,000 + 50\,000 + 20\,000 + 5\,000 + 2\,000 + 1\,000 + 500 + 2 \times 200 + 50 = 178\,950$. Cara selalu mengambil nilai terbesar yang masih muat disebut metode **rakus** (*greedy*); untuk uang rupiah cara ini selalu menghasilkan jumlah lembar dan keping paling sedikit, tetapi untuk beberapa himpunan nilai lain tidak demikian, sebuah pertanyaan yang dikaji dalam studi tentang algoritma. ## Daftar Pustaka

[1] G. van Rossum, B. Warsaw, and A. Coghlan, "PEP 8 – Style Guide for Python Code," Python Software Foundation, 2001 (updated 2013). [Online]. Available: https://peps.python.org/pep-0008/

[2] D. J. Barnes and M. Kölling, *Objects First with Java: A Practical Introduction Using BlueJ*, 6th ed. Harlow, UK: Pearson, 2016.

[3] The JUnit Team, *JUnit User Guide*, version 6.1.3, Aug. 2026. Accessed: Sep. 24, 2026. [Online]. Available: https://docs.junit.org/current/

[4] Oracle, "Class String," *Java SE 21 & JDK 21 API Specification*, 2023. [Online]. Available: https://docs.oracle.com/en/java/javase/21/docs/api/java.base/java/lang/String.html

[5] Python Software Foundation, "Built-in Types: Text Sequence Type — str," *The Python Standard Library*, 2025. [Online]. Available: https://docs.python.org/3/library/stdtypes.html

[6] J. Gosling, B. Joy, G. Steele, G. Bracha, A. Buckley, D. Smith, and G. Bierman, *The Java Language Specification, Java SE 21 Edition*. Redwood City, CA, USA: Oracle, 2023.

[7] *IEEE Standard for Floating-Point Arithmetic*, IEEE Std 754-2019, 2019.

[8] D. Goldberg, "What every computer scientist should know about floating-point arithmetic," *ACM Computing Surveys*, vol. 23, no. 1, pp. 5–48, 1991.

[9] G. Bierman, "JEP 361: Switch Expressions," OpenJDK, 2019. [Online]. Available: https://openjdk.org/jeps/361

[10] D. F. Moisset, "PEP 636 – Structural Pattern Matching: Tutorial," Python Software Foundation, 2020. [Online]. Available: https://peps.python.org/pep-0636/

[11] R. Sedgewick and K. Wayne, *Computer Science: An Interdisciplinary Approach*. Boston, MA, USA: Addison-Wesley, 2016.

[12] Republik Indonesia, *Undang-Undang Nomor 27 Tahun 2022 tentang Pelindungan Data Pribadi*. Jakarta, Indonesia, 2022.

Bab 4

Evaluasi dan Pembahasan I: Dari Spesifikasi Sederhana ke Kelas yang Teruji

Bab ini tidak menambahkan fitur bahasa yang baru. Sebaliknya, bab ini mengambil semua yang telah diberikan Bab 1 sampai Bab 3 — meja kerja, kelas dan objek, string, tipe, percabangan, dan perulangan — lalu memakainya untuk menyelesaikan empat belas soal yang utuh. Tujuh di antaranya adalah soal-soal pertama yang klasik bagi siapa pun yang mempelajari pemrograman berorientasi objek: menjumlahkan pecahan, mengukur sudut antara jarum-jarum jam, sebuah akun komputer, bilangan kompleks, berat dalam pon dan kilogram, sebuah kelas pecahan, serta pertarungan kecil antara seorang pemain dan seorang musuh. Tujuh lainnya baru dan diambil dari kehidupan sehari-hari: jumlah uang dalam rupiah, waktu dalam sehari, nota Profitina Laundry, nama pelanggan yang dirapikan, aturan kata sandi, perulangan yang ditelusuri dengan tangan, dan tahun kabisat. Setiap soal diselesaikan dari nol, sangat perlahan, dan selalu dengan urutan yang sama: soal dinyatakan ulang dengan kata-kata sederhana, dua atau tiga contoh dikerjakan dengan tangan, kelas, *field*, dan metodenya dikenali, rancangannya digambar, penyelesaian Java beserta driver ujinya ditulis, penyelesaian Python dengan `unittest` ditulis, keluaran nyata keduanya ditampilkan, lalu dijelaskan baris demi baris, disertai sebuah *penyelesaian keliru yang menggoda* lengkap dengan cara persis kegagalannya, dan sebuah varian *uji diri* beserta jawabannya. Di akhir bab ini Anda akan memiliki sebuah metode untuk menggarap soal pemrograman kecil apa pun, dan empat belas contoh yang dikerjakan tuntas dengan metode itu sebagai pegangan.

4.1 Dari Mana Kita Berangkat

Sebelum menyelesaikan apa pun, mari kita gelar dulu kotak perkakasnya. Setiap soal dalam bab ini meraih beberapa perkakas berikut, dan menyebutkan perkakas yang dibutuhkan sudah merupakan separuh pekerjaan.

Dari **Bab 1** Anda memiliki sebuah meja kerja: Java Development Kit (JDK), penerjemah (*interpreter*) Python, dan Visual Studio Code dengan ekstensi Java dan Python-nya. Anda tahu cara membuat berkas, menjalankannya, dan membaca apa yang dicetaknya. Anda juga memegang gagasan pokok buku ini: sebuah *objek* membungkus sejumlah data bersama operasi-operasi yang boleh dikenakan pada data itu.

Dari **Bab 2** Anda dapat menyunting, menjalankan, dan *men-debug*: memasang *breakpoint*, melangkah melalui program baris demi baris, dan mengamati nilai sebuah variabel berubah. Beberapa penyelesaian keliru yang menggoda dalam bab ini adalah jenis *bug* yang ditemukan *debugger* dalam satu menit, dan kita akan menyebutkannya setiap kali.

Dari **Bab 3** Anda memiliki bahasanya sendiri. Sebuah *kelas* (*class*) adalah cetak biru; sebuah *objek* adalah satu benda yang dibangun darinya dengan `new` (Java) atau dengan memanggil kelas itu (Python). Sebuah kelas memiliki *field* (data yang dibawa setiap objek), sebuah *konstruktor* (kode

yang mengisi *field* ketika objek lahir), dan *metode* (operasinya). Anda mengenal tipe primitif `int`, `long`, dan `double` beserta jebakan yang tersembunyi di dalamnya, terutama pembagian bilangan bulat; Anda mengenal `String` dan metode-metodenya `length`, `charAt`, `substring`, dan `equals`; Anda mengenal `if`, `else if`, `else`, dan `switch`; dan Anda mengenal perulangan `while` dan `for`. Bab 3 ditutup dengan latihan-latihan kecil seperti luas lingkaran, pengubah massa imperial ke kilogram, dan bunga majemuk; Soal 4.5 tumbuh langsung dari pengubah massa itu.

Tabel 4.1 memperlihatkan perkakas mana yang paling dilatih oleh setiap soal dalam bab ini. Pakailah sebagai peta: bila Anda masih ragu tentang perulangan, mulailah dari Soal 4.11 sampai 4.13; bila kelas masih terasa asing, mulailah dari Soal 4.3 dan 4.5.

Tabel 4.1 Keempat belas soal dalam bab ini dan perkakas dari Bab 1 sampai 3 yang paling banyak dilatih oleh masing-masing soal.

Soal	Topik	Perkakas utama dari Bab 1–3
4.1	Kalkulator pecahan	aritmetika `int`, metode statis, penggabungan string
4.2	Sudut jarum jam analog	`double`, operator `%`, pembulatan, normalisasi
4.3	Akun komputer	*field* privat, konstruktor, `equals` pada string
4.4	Bilangan kompleks	metode yang mengembalikan objek baru, `toString`, `if`
4.5	Konversi berat	konstanta, nilai turunan yang dihitung saat diminta
4.6	Kelas pecahan	objek sebagai parameter dan hasil, imutabilitas
4.7	Pemain dan musuh	dua kelas yang bekerja sama, `if`, perulangan `while`
4.8	Uang dalam rupiah	`long`, perulangan atas karakter-karakter string
4.9	Waktu dalam sehari	satu *field* sebagai ganti dua, putaran balik dengan `%`, pemformatan
4.10	Nota laundry	`switch`, pembulatan ke atas, satu objek di dalam objek lain
4.11	Merapikan nama	`charAt`, uji karakter, membangun string dalam perulangan
4.12	Aturan kata sandi	menghitung dalam perulangan, memutuskan sesudahnya
4.13	Menelusuri perulangan	tabel jejak, `while`, algoritma Euclid
4.14	Tahun kabisat	ekspresi Boolean, `switch`, metode yang mengembalikan objek berikutnya

4.2 Cara Menggarap Soal Pemrograman

Seorang pemula yang disodori sebuah soal biasanya langsung membuka editor dan mulai mengetik `public class`. Seorang pemrogram berpengalaman biasanya tidak menyentuh papan ketik selama beberapa menit pertama. Bedanya bukan bakat; bedanya adalah kebiasaan kerja. Matematikawan George Pólya menguraikan kebiasaan semacam itu untuk soal-soal matematika — pahami soalnya, susun rencana, laksanakan rencana itu, lalu tengok kembali [1] — dan empat gerakan yang sama, yang direntang menjadi delapan langkah kecil, sama bermanfaatnya bagi pemrogram. **Gambar 4.1** memperlihatkan delapan langkah yang akan kita ikuti untuk setiap soal dalam bab ini.



Gambar 4.1 Delapan langkah yang dipakai untuk setiap soal dalam bab ini; panah putus-putus adalah putaran antara menulis kode dan menjalankan uji.

4.2.1 Delapan langkah

Langkah 1 — Baca soalnya dan nyatakan ulang dengan kata-kata Anda sendiri. Bacalah pernyataan soal dua kali. Lalu tulislah, dalam satu atau dua kalimat Anda sendiri, apa yang *masuk* dan apa yang harus *keluar*. Bila Anda tidak mampu melakukannya, Anda belum memahami soalnya, dan sebanyak apa pun kode tidak akan menolong. Lingkarilah setiap bilangan dalam pernyataan soal: bilangan-bilangan itu biasanya adalah contoh yang akan Anda pakai untuk menguji.

Langkah 2 — Kerjakan dua atau tiga contoh dengan tangan. Ambillah contoh-contoh yang diberikan soal, lalu hitung jawabannya dengan pensil dan kertas *sebelum* menulis kode apa pun. Tuliskan setiap bilangan antara. Di sinilah Anda mengetahui apakah Anda benar-benar memahami rumusnya, dan di sini pula Anda memperoleh keluaran yang diharapkan untuk langkah 5. Tambahkan satu contoh buatan Anda sendiri di tepi: nol, bilangan negatif, nilai terbesar yang diizinkan, string kosong.

Langkah 3 — Temukan kata benda dan kata kerjanya. Dalam program berorientasi objek, *kata benda* dalam pernyataan soal (“sebuah pecahan”, “sebuah akun”, “sebuah nota”) adalah calon kelas; *sifat* setiap kata benda (“pembilangnya”, “kata sandinya”, “totalnya”) adalah calon *field*; dan *kata kerja* (“menjumlahkan”, “mengganti kata sandi”, “mencetak”) adalah calon metode. Buatlah daftarnya.

Langkah 4 — Gambarlah rancangannya. Gambarlah setiap kelas sebagai kotak dengan tiga ruang: namanya, *field*-nya beserta tipenya, dan metode-metodenya beserta parameter dan hasilnya. Ini adalah bentuk sederhana dari *diagram kelas UML* (UML adalah *Unified Modeling Language*, notasi baku untuk menggambar rancangan perangkat lunak). Tanda minus menandai sesuatu yang privat, tanda plus sesuatu yang publik. Dalam bab ini kita menggambar kotak-kotak itu dengan karakter biasa, sehingga Anda dapat menyalinnya ke kertas.

Langkah 5 — Tulis ujinya lebih dulu. Ubahlah contoh-contoh dari langkah 2 menjadi pemeriksaan: “untuk masukan ini, metode itu harus mengembalikan tepat nilai ini”. Menulisnya sebelum kode terdengar terbalik, tetapi cara ini memaksa Anda memutuskan dengan tepat apa arti “benar” selagi kepala Anda masih jernih.

Langkah 6 — Tulis kodenya, sepotong kecil demi sepotong kecil. Tulislah *field* dan konstruktornya lebih dulu, kompilasi, lalu jalankan. Kemudian satu metode, kompilasi, lalu jalankan. Program yang tumbuh dalam langkah-langkah kecil yang teruji tidak pernah menyuguhi Anda lima puluh galat sekaligus.

Langkah 7 — Jalankan, bandingkan, dan debug. Jalankan ujinya. Bila sebuah pemeriksaan gagal, jangan menebak: pasang *breakpoint* di metode yang gagal (Bab 2) dan amati variabel-variabelnya sampai Anda melihat nilai pertama yang berbeda dari hitungan tangan Anda.

Perbedaan itu *adalah* bug-nya. Kembalilah ke langkah 6, perbaiki, dan jalankan lagi semua uji — itulah panah putus-putus dalam **Gambar 4.1**.

Langkah 8 — Tengok kembali. Setelah semuanya lulus, ajukan tiga pertanyaan. Masukan mana yang belum saya uji? Apa cara keliru yang paling *menggoda* untuk menyelesaikan soal ini, dan mengapa cara itu gagal? Dapatkah saya menyelesaikan versi yang sedikit berbeda? Ketiga pertanyaan ini persis merupakan tiga bagian terakhir dari setiap soal dalam bab ini.

> **MENGAPA CONTOH DENGAN TANGAN DIDAHULUKAN.** Sebuah program hanya bisa sebenar pemahaman Anda tentang soalnya. Bila Anda tidak dapat menghitung $\frac{1}{2}+\frac{1}{3}$ di atas kertas, Anda tidak dapat memeriksa program yang mengklaim hasilnya $\frac{5}{6}$. Setiap nilai harapan dalam driver uji bab ini lebih dulu dihitung dengan tangan pada bagian "Memahaminya dengan tangan" dari soalnya.

4.2.2 Cara uji dalam bab ini ditulis

Program Java profesional diuji dengan JUnit, kini JUnit 6 (API ujinya, yang disebut *Jupiter*, adalah API yang diperkenalkan JUnit 5, sehingga uji yang ditulis untuk JUnit 5 tetap berjalan tanpa perubahan), sebuah pustaka yang setiap pemeriksaannya berupa metode bertanda `@Test`, dan pernyataan seperti `assertEquals(5, result)` membandingkan nilai yang diharapkan dengan nilai yang sebenarnya [2]. JUnit adalah unduhan terpisah, dan buku ini tidak ingin uji pertama Anda bergantung pada berhasilnya memasang sebuah pustaka ke *class path*. Karena itu, setiap penyelesaian Java dalam bab ini disertai sebuah *driver uji sederhana*: kelas biasa dengan metode `main` dan metode pembantu kecil `check(label, expected, actual)` yang mencetak `PASS` atau `FAIL` dan menghitung hasilnya. Pembantu itu membandingkan dengan `equals`, persis seperti yang dilakukan `assertEquals` di dalam JUnit, sehingga memindahkan sebuah driver ke JUnit kelak hanyalah pekerjaan mekanis: setiap `check(label, expected, actual)` menjadi `assertEquals(expected, actual, label)` di dalam metode bertanda `@Test`. README pada folder kode bab ini menguraikan langkah itu di VS Code: buka tampilan *Testing*, pilih *Enable Java Tests*, pilih *JUnit Jupiter*, dan VS Code mengunduh pustakanya serta menampilkan tanda hijau atau merah di samping setiap uji.

Python membawa pustaka pengujian di dalam pustaka bakunya, yaitu `unittest` [3]. Karena itu, setiap penyelesaian Python dalam bab ini memuat sebuah kelas yang merupakan turunan `unittest.TestCase`, dan metode-metodenya yang bernama `test_…` memanggil `self.assertEqual(actual, expected)`, `self.assertTrue(…)`, atau `self.assertAlmostEqual(…)`. Baris-baris terakhir setiap berkas menjalankan demonstrasi singkat lalu ujinya, sehingga satu perintah — `python3 file.py` — menampilkan keduanya. Uji yang sama juga dapat dijalankan dengan `python3 -m unittest file` atau dengan tampilan *Testing* di VS Code.

Satu catatan praktis tentang keluaran yang dikutip: `unittest` melaporkan berapa lama ujinya berlangsung, misalnya `Ran 4 tests in 0.000s`. Komputer Anda mungkin mencetak `0.001s`; hanya baris itulah yang boleh berbeda dari buku.

4.2.3 Susunan setiap soal

Setiap subbab soal memiliki bagian-bagian yang sama, sehingga Anda selalu tahu di mana Anda berada:

1. **Soalnya** — pernyataan soal, di dalam kotak.

2. **Memahaminya dengan tangan** — soal dinyatakan ulang dengan kata-kata sederhana dan contoh-contohnya dihitung di atas kertas (langkah 1 dan 2).
3. **Kelas, field, dan metode** — kata benda dan kata kerjanya, serta kotak kelasnya (langkah 3 dan 4).
4. **Penyelesaian Java** — kelasnya, driver ujinya, dan keluaran nyata keduanya (langkah 5 sampai 7).
5. **Penyelesaian Python** — rancangan yang sama dalam Python, dengan `unittest`, beserta keluaran nyatanya.
6. **Baris demi baris** — apa yang dikerjakan setiap baris penting dan mengapa.
7. **Penyelesaian keliru yang menggoda** — program yang tampak benar, keluaran nyatanya, dan mengapa ia gagal (langkah 8).
8. **Uji diri** — sebuah varian soal beserta jawabannya.

Semua program ada di folder kode bab ini, `Chapter04/java` dan `Chapter04/python`; penyelesaian keliru dalam Java disimpan di `Chapter04/java/tempting`, karena salah satunya *memang dimaksudkan* gagal dikompilasi dan akan menghentikan `javac *.java` dari mengompilasi sisanya. Penyelesaian Java mempertahankan nama-nama kelas dari rangkaian soal klasik, seperti `Q11Fraction` dan `Q16Fraction`: awalan itu memisahkan kedua kelas pecahan dari Soal 4.1 dan 4.6 di dalam satu folder.

4.3 Soal 4.1 — Kalkulator Pecahan

4.3.1 Soalnya

> **SOAL 4.1.** Let `e1` and `d1` represent the fraction $\frac{e_1}{d_1}$, and correspondingly `e2` and `d2` the fraction $\frac{e_2}{d_2}$, where e_1 and e_2 are integers and d_1 and d_2 are positive integers. Write a Java class `Q11Fraction` that computes the numbers `es`, `ds` and `ep`, `dp`, which stand for the sum and the product of the two fractions. For example, $\frac{1}{2}+\frac{1}{3}=\frac{5}{6}$ and $\frac{1}{2}\cdot\frac{1}{3}=\frac{1}{6}$. Addition and multiplication of fractions are defined by Persamaan (4.1) and Persamaan (4.2). Test the class for $\frac{1}{2}+\frac{1}{3}$, $\frac{1}{3}+\frac{3}{4}$, $\frac{1}{2}\cdot\frac{2}{3}$ and $\frac{1}{4}\cdot\frac{2}{3}$.

Dengan kata lain: pasangan `e1` dan `d1` menyatakan pecahan $\frac{e_1}{d_1}$, dan pasangan `e2` dan `d2` menyatakan pecahan $\frac{e_2}{d_2}$; pembilang e_1 dan e_2 bilangan bulat, sedangkan penyebut d_1 dan d_2 bilangan bulat positif. Tulislah kelas Java `Q11Fraction` yang menghitung bilangan `es` dan `ds` (pembilang dan penyebut jumlahnya) serta `ep` dan `dp` (pembilang dan penyebut hasil kalinya), lalu ujilah untuk keempat contoh yang disebutkan.

$$\frac{e_1}{d_1}+\frac{e_2}{d_2}=\frac{e_1 d_2+e_2 d_1}{d_1 d_2} \tag{4.1}$$

$$\frac{e_1}{d_1}\cdot\frac{e_2}{d_2}=\frac{e_1 e_2}{d_1 d_2} \tag{4.2}$$

4.3.2 Memahaminya dengan tangan

Dengan kata-kata sederhana: kita diberi empat bilangan bulat yang menggambarkan dua pecahan, dan kita harus menghasilkan empat bilangan bulat lagi yang menggambarkan jumlah dan hasil kali keduanya. Huruf e menandai bagian atas sebuah pecahan (*pembilang*, *numerator*) dan d bagian bawahnya (*penyebut*, *denominator*). Jawabannya *bukan* bilangan desimal seperti 0.8333; jawabannya adalah sepasang bilangan bulat.

Mengapa Persamaan (4.1) benar? Untuk menjumlahkan dua pecahan, kita harus lebih dulu memotong keduanya menjadi potongan-potongan yang sama besar. Bila pembilang dan penyebut $\frac{e_1}{d_1}$ kita kalikan dengan d_2, dan pembilang dan penyebut $\frac{e_2}{d_2}$ kita kalikan dengan d_1, kedua pecahan memiliki penyebut bersama $d_1 d_2$, lalu kita cukup menjumlahkan pembilangnya. Perkalian tidak memerlukan penyebut bersama: "setengah dari sepertiga" adalah satu potong dari $2 \cdot 3 = 6$ potong, jadi pembilang dikalikan dengan pembilang dan penyebut dengan penyebut.

Sekarang keempat contohnya, satu langkah per baris:

- $\frac{1}{2} + \frac{1}{3}$: $e_s = 1 \cdot 3 + 1 \cdot 2 = 3 + 2 = 5$ dan $d_s = 2 \cdot 3 = 6$, jadi jumlahnya $\frac{5}{6}$.
- $\frac{1}{3} + \frac{3}{4}$: $e_s = 1 \cdot 4 + 3 \cdot 3 = 4 + 9 = 13$ dan $d_s = 3 \cdot 4 = 12$, jadi jumlahnya $\frac{13}{12}$.
- $\frac{1}{2} \cdot \frac{2}{3}$: $e_p = 1 \cdot 2 = 2$ dan $d_p = 2 \cdot 3 = 6$, jadi hasil kalinya $\frac{2}{6}$.
- $\frac{1}{4} \cdot \frac{2}{3}$: $e_p = 1 \cdot 2 = 2$ dan $d_p = 4 \cdot 3 = 12$, jadi hasil kalinya $\frac{2}{12}$.

Perhatikan bahwa $\frac{2}{6}$ sama dengan $\frac{1}{3}$ dan $\frac{2}{12}$ sama dengan $\frac{1}{6}$. Soal tidak meminta kita menyederhanakan, jadi program seharusnya mencetak $\frac{2}{6}$, dan uji kita akan mengharapkan tepat itu. (Penyederhanaan adalah *uji diri* Soal 4.6.) Contoh tepi buatan kita sendiri: pecahan pertama yang negatif, $-\frac{1}{2} + \frac{1}{3} = \frac{-3+2}{6} = \frac{-1}{6}$.

4.3.3 Kelas, field, dan metode

Di sini belum ada objek yang memiliki ingatan: soal hanya meminta empat perhitungan. Maka kelas `Q11Fraction` tidak memiliki *field*, hanya metode *statis* — metode yang dimiliki oleh kelas itu sendiri dan dipanggil sebagai `Q11Fraction.sumNumerator(…)` tanpa membuat objek (Bab 3). Satu metode untuk setiap bilangan yang harus kita hitung menjaga setiap rumus berada tepat di satu tempat:

```
+-------------------------------------------------------------+
| Q11Fraction                                                 |
+-------------------------------------------------------------+
| (tanpa field)                                               |
+-------------------------------------------------------------+
| + sumNumerator(e1, d1, e2, d2) : int        (static)        |
| + sumDenominator(d1, d2) : int              (static)        |
| + productNumerator(e1, e2) : int            (static)        |
| + productDenominator(d1, d2) : int          (static)        |
| + printSum(e1, d1, e2, d2), printProduct(e1, d1, e2, d2)    |
| + main(args)                                                |
+-------------------------------------------------------------+
```

Bahwa sebuah pecahan harus diedarkan sebagai dua bilangan lepas terasa canggung; peganglah perasaan itu, karena Soal 4.6 menyembuhkannya.

4.3.4 Penyelesaian Java

Kode 4.1 adalah kelasnya. Metode `main`-nya mencetak keempat hasil yang diminta soal.

Kode 4.1 `Q11Fraction.java`: jumlah dan hasil kali dua pecahan hanya dengan bilangan bulat.

```java
/**
 * Soal 4.1: jumlah dan hasil kali dua pecahan e1/d1 dan e2/d2,
 * hanya dengan bilangan bulat:
 *   e1/d1 + e2/d2 = (e1*d2 + e2*d1) / (d1*d2)
 *   e1/d1 * e2/d2 = (e1*e2) / (d1*d2)
 * Hasilnya tidak disederhanakan ke bentuk paling sederhana.
 */
public class Q11Fraction {

    public static int sumNumerator(int e1, int d1, int e2, int d2) {
        return e1 * d2 + e2 * d1;
    }

    public static int sumDenominator(int d1, int d2) {
        return d1 * d2;
    }

    public static int productNumerator(int e1, int e2) {
        return e1 * e2;
    }

    public static int productDenominator(int d1, int d2) {
        return d1 * d2;
    }

    public static void printSum(int e1, int d1, int e2, int d2) {
        int es = sumNumerator(e1, d1, e2, d2);
        int ds = sumDenominator(d1, d2);
        System.out.println(e1 + "/" + d1 + " + " + e2 + "/" + d2 + " = " + es + "/" + ds);
    }

    public static void printProduct(int e1, int d1, int e2, int d2) {
        int ep = productNumerator(e1, e2);
        int dp = productDenominator(d1, d2);
        System.out.println(e1 + "/" + d1 + " * " + e2 + "/" + d2 + " = " + ep + "/" + dp);
    }

    public static void main(String[] args) {
        printSum(1, 2, 1, 3);
        printSum(1, 3, 3, 4);
        printProduct(1, 2, 2, 3);
        printProduct(1, 4, 2, 3);
    }
}
```

Dikompilasi dengan `javac Q11Fraction.java` dan dijalankan dengan `java Q11Fraction`, program ini mencetak:

```
1/2 + 1/3 = 5/6
1/3 + 3/4 = 13/12
1/2 * 2/3 = 2/6
1/4 * 2/3 = 2/12
```

Keempat baris itu cocok dengan hitungan tangan kita. Driver uji dalam **Kode 4.2** mengubah contoh-contoh yang sama, ditambah contoh negatif, menjadi pemeriksaan otomatis.

Kode 4.2 `Q11FractionTest.java`: driver uji sederhana untuk Soal 4.1.

```java
/** Driver uji sederhana untuk Q11Fraction: jalankan dengan "java Q11FractionTest". */
public class Q11FractionTest {
    private static int passed = 0;
    private static int failed = 0;

    private static void check(String label, Object expected, Object actual) {
        if (expected.equals(actual)) {
            passed++;
            System.out.println("PASS " + label);
        } else {
            failed++;
            System.out.println("FAIL " + label + ": expected " + expected + ", got " +
actual);
        }
    }

    public static void main(String[] args) {
        check("1/2 + 1/3 numerator", 5, Q11Fraction.sumNumerator(1, 2, 1, 3));
        check("1/2 + 1/3 denominator", 6, Q11Fraction.sumDenominator(2, 3));
        check("1/3 + 3/4 numerator", 13, Q11Fraction.sumNumerator(1, 3, 3, 4));
        check("1/3 + 3/4 denominator", 12, Q11Fraction.sumDenominator(3, 4));
        check("1/2 * 2/3 numerator", 2, Q11Fraction.productNumerator(1, 2));
        check("1/4 * 2/3 denominator", 12, Q11Fraction.productDenominator(4, 3));
        check("-1/2 + 1/3 numerator", -1, Q11Fraction.sumNumerator(-1, 2, 1, 3));
        System.out.println(passed + " passed, " + failed + " failed");
    }
}
```

```
PASS 1/2 + 1/3 numerator
PASS 1/2 + 1/3 denominator
PASS 1/3 + 3/4 numerator
PASS 1/3 + 3/4 denominator
PASS 1/2 * 2/3 numerator
PASS 1/4 * 2/3 denominator
PASS -1/2 + 1/3 numerator
7 passed, 0 failed
```

4.3.5 Penyelesaian Python

Kode 4.3 adalah rancangan yang sama dalam Python: empat fungsi, dua fungsi pembantu pencetak, dan sebuah kelas uji `unittest`.

Kode 4.3 `q11_fraction.py`: Soal 4.1 dalam Python, beserta uji `unittest`-nya.

```
"""Soal 4.1: jumlah dan hasil kali dua pecahan e1/d1 dan e2/d2,
hanya dengan bilangan bulat:
    e1/d1 + e2/d2 = (e1*d2 + e2*d1) / (d1*d2)
    e1/d1 * e2/d2 = (e1*e2) / (d1*d2)
Hasilnya tidak disederhanakan ke bentuk paling sederhana.
"""
import unittest


def sum_numerator(e1, d1, e2, d2):
    return e1 * d2 + e2 * d1


def sum_denominator(d1, d2):
    return d1 * d2


def product_numerator(e1, e2):
    return e1 * e2


def product_denominator(d1, d2):
    return d1 * d2


def print_sum(e1, d1, e2, d2):
    es = sum_numerator(e1, d1, e2, d2)
    ds = sum_denominator(d1, d2)
    print(f"{e1}/{d1} + {e2}/{d2} = {es}/{ds}")


def print_product(e1, d1, e2, d2):
    ep = product_numerator(e1, e2)
    dp = product_denominator(d1, d2)
    print(f"{e1}/{d1} * {e2}/{d2} = {ep}/{dp}")


class TestQ11Fraction(unittest.TestCase):
    def test_sum_half_and_third(self):
        self.assertEqual(sum_numerator(1, 2, 1, 3), 5)
        self.assertEqual(sum_denominator(2, 3), 6)

    def test_sum_third_and_three_quarters(self):
        self.assertEqual(sum_numerator(1, 3, 3, 4), 13)
        self.assertEqual(sum_denominator(3, 4), 12)
```

Kode 4.3 (Lanjutan) *q11_fraction.py: Soal 4.1 dalam Python, beserta uji unittest-nya.*

```
    def test_products(self):
        self.assertEqual(product_numerator(1, 2), 2)
        self.assertEqual(product_denominator(4, 3), 12)

    def test_negative_numerator(self):
        self.assertEqual(sum_numerator(-1, 2, 1, 3), -1)


if __name__ == "__main__":
    print_sum(1, 2, 1, 3)
    print_sum(1, 3, 3, 4)
    print_product(1, 2, 2, 3)
    print_product(1, 4, 2, 3)
    unittest.main(verbosity=2)
```

Menjalankan `python3 q11_fraction.py` mencetak keempat hasil lalu putusan setiap uji:

```
1/2 + 1/3 = 5/6
1/3 + 3/4 = 13/12
1/2 * 2/3 = 2/6
1/4 * 2/3 = 2/12
test_negative_numerator (__main__.TestQ11Fraction.test_negative_numerator) ... ok
test_products (__main__.TestQ11Fraction.test_products) ... ok
test_sum_half_and_third (__main__.TestQ11Fraction.test_sum_half_and_third) ... ok
test_sum_third_and_three_quarters
(__main__.TestQ11Fraction.test_sum_third_and_three_quarters) ... ok

----------------------------------------------------------------------
Ran 4 tests in 0.000s

OK
```

4.3.6 Baris demi baris

- `public static int sumNumerator(int e1, int d1, int e2, int d2)` — `public` berarti kelas mana pun boleh memanggilnya, `static` berarti metode ini milik kelas, bukan milik sebuah objek, dan `int` berarti metode ini mengembalikan bilangan bulat. Badannya adalah Persamaan (4.1) yang diketik nyaris huruf demi huruf: `e1 * d2 + e2 * d1`. Java, seperti aritmetika sekolah, mengalikan sebelum menjumlahkan, jadi tanda kurung tidak diperlukan.
- `sumDenominator` dan `productDenominator` memuat ekspresi yang sama, `d1 * d2`. Itu bukan duplikasi *pengetahuan*: yang satu adalah "penyebut sebuah jumlah" dan yang lain "penyebut sebuah hasil kali", yang kebetulan saja sama.
- `System.out.println(e1 + "/" + d1 + " + " + …)` — operator `+` di antara string dan bilangan melakukan *penggabungan* (menyambung teks), sehingga `1 + "/" + 2` menjadi teks `1/2`. Ini berhasil karena Java membaca ekspresi dari kiri ke kanan, dan operan pertama yang bersebelahan dengan string mengubah seluruh ekspresi menjadi teks.
- Dalam driver uji, `check("1/2 + 1/3 numerator", 5, Q11Fraction.sumNumerator(1, 2, 1, 3))` meneruskan bilangan `5` ke parameter bertipe `Object`. Java membungkus `int` itu secara otomatis menjadi objek `Integer` (ini disebut *autoboxing*), dan `expected.equals(actual)` lalu membandingkan dua objek `Integer` berdasarkan nilainya.
- Dalam Python, `f"{e1}/{d1} + {e2}/{d2} = {es}/{ds}"` adalah sebuah *f-string*: setiap ekspresi di dalam kurung kurawal diganti dengan nilainya. Blok `if __name__ == "__main__":` hanya berjalan bila berkas itu dijalankan langsung, bukan ketika berkas lain mengimpornya; `unittest.main(verbosity=2)` menemukan setiap metode yang namanya diawali `test` dan mencetak satu baris per uji.

4.3.7 Penyelesaian keliru yang menggoda

Tiga gagasan hampir selalu muncul pada pemula. **Kode 4.4** memperlihatkan ketiganya.

Kode 4.4 `WrongFractionSum.java`: tiga cara menggoda untuk menjumlahkan $\frac{1}{2}$ dan $\frac{1}{3}$.

```java
/** Tiga cara menggoda tetapi salah untuk menjumlahkan 1/2 dan 1/3. */
public class WrongFractionSum {
    public static void main(String[] args) {
        int e1 = 1, d1 = 2, e2 = 1, d2 = 3;
        System.out.println("tops and bottoms added: " + (e1 + e2) + "/" + (d1 + d2));
        System.out.println("int division:           " + (e1 / d1 + e2 / d2));
        System.out.println("double division:        " + ((double) e1 / d1 + (double) e2 /
d2));
    }
}
```

```
tops and bottoms added: 2/5
int division:           0
double division:        0.8333333333333333
```

Baris pertama menjumlahkan pembilang dengan pembilang dan penyebut dengan penyebut: $\frac{1+1}{2+3} = \frac{2}{5}$. Kelihatannya rapi, tetapi jelas salah — setengah ditambah sepertiga lebih besar daripada setengah, sedangkan $\frac{2}{5}$ lebih kecil. Baris kedua adalah jebakan *pembagian bilangan bulat* yang terkenal dari Bab 3: dalam Java, `1 / 2` di antara dua nilai `int` membuang sisanya dan menghasilkan `0`, demikian pula `1 / 3`, sehingga "jumlahnya" `0`. Baris ketiga mengubah ke `double` lebih dulu dan memperoleh bilangan yang mendekati kebenaran, 0.8333 …, tetapi bukan itu yang diminta: soal menginginkan dua bilangan bulat, dan desimal tak berhingga $0.8\overline{3}$ bahkan tidak dapat disimpan dengan tepat. **Kode 4.5** membuat kesalahan yang sama dalam Python, tempat `//` adalah pembagian bilangan bulat (pembagian lantai) dan `/` selalu menghasilkan bilangan titik-mengambang.

Kode 4.5 `wrong_fraction_sum.py`: tiga kesalahan yang sama dalam Python.

```python
"""Tiga cara menggoda tetapi salah untuk menjumlahkan 1/2 dan 1/3."""
e1, d1, e2, d2 = 1, 2, 1, 3
print(f"tops and bottoms added: {e1 + e2}/{d1 + d2}")
print(f"floor division:         {e1 // d1 + e2 // d2}")
print(f"true division:          {e1 / d1 + e2 / d2}")
```

```
tops and bottoms added: 2/5
floor division:         0
true division:          0.8333333333333333
```

Debugger memperlihatkan kesalahan kedua seketika: lewati `e1 / d1` dengan *step over*, dan jendela pengamatan menampilkan `0`, bukan `0.5`.

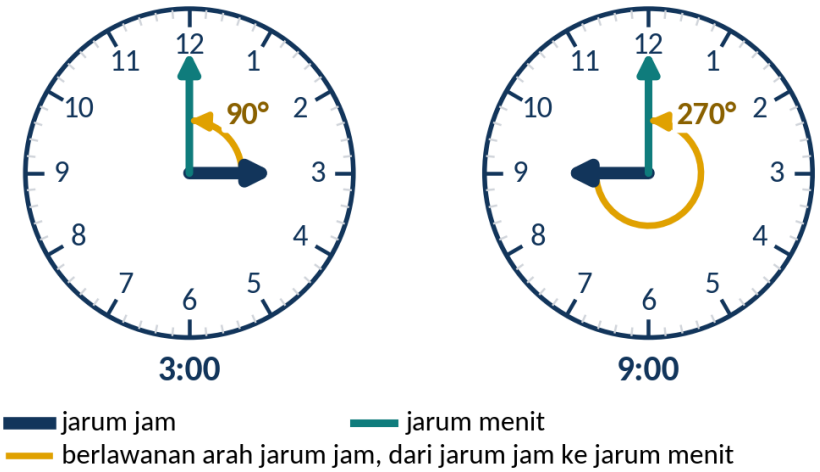
> **UJI DIRI.** Tambahkan dua metode `differenceNumerator(e1, d1, e2, d2)` dan `differenceDenominator(d1, d2)` untuk $\frac{e_1}{d_1} - \frac{e_2}{d_2}$. Apa yang harus dicetak oleh `printDifference(1, 3, 3, 4)`?
>
> *Jawaban.* Dengan argumen penyebut bersama yang sama, $\frac{e_1}{d_1} - \frac{e_2}{d_2} = \frac{e_1 d_2 - e_2 d_1}{d_1 d_2}$, sehingga `differenceNumerator` mengembalikan `e1 * d2 - e2 * d1` dan `differenceDenominator` mengembalikan `d1 * d2`. Untuk $\frac{1}{3} - \frac{3}{4}$: $1 \cdot 4 - 3 \cdot 3 = 4 - 9 = -5$ dan $3 \cdot 4 = 12$, sehingga barisnya `1/3 - 3/4 = -5/12`.

4.4 Soal 4.2 — Sudut Jarum Jam Analog

4.4.1 Soalnya

> **SOAL 4.2.** The time is 13:03. We represent the time by two variables, `hours = 13;` and `minutes = 3;`. Write a Java class `Q12Time` that computes the angle between the hour hand and the minute hand on a traditional analog clock. Angles are measured counterclockwise from the hour hand to the minute hand. The result is rounded and normalized so that it lies between 0 and 359 inclusive. For instance, the angles at 3:00 and at 9:00 are 90° and 270° (**Gambar 4.2**). Hint: one minute corresponds to 6° and one hour to 30°; start at 12 o'clock. The program must work for any input with `hours` from 0 to 23 and `minutes` from 0 to 59, and its comments should explain why it is correct. Test it for 9:00, 3:00, 18:00, 1:00, 2:30 and 4:41, whose results are 270, 90, 180, 30, 255 and 254.

Dengan kata lain: sebuah waktu dinyatakan dengan dua variabel, `hours` (0 sampai 23) dan `minutes` (0 sampai 59). Tulislah kelas Java `Q12Time` yang menghitung sudut antara jarum jam dan jarum menit pada jam analog biasa, diukur *berlawanan* arah jarum jam dari jarum jam ke jarum menit, dibulatkan, dan dinormalisasi agar berada di antara 0 dan 359. Satu menit setara dengan 6° dan satu jam dengan 30°, dihitung mulai dari angka 12. Komentar program harus menjelaskan mengapa program itu benar, dan keenam waktu uji di atas harus menghasilkan 270, 90, 180, 30, 255, dan 254.



Gambar 4.2 Jam pada pukul 3:00 (kiri) dan pukul 9:00 (kanan); busur jingga bergerak berlawanan arah jarum jam dari jarum jam (pendek, gelap) ke jarum menit (panjang, hijau kebiruan), menghasilkan 90° dan 270°.

4.4.2 Memahaminya dengan tangan

Kedua jarum berputar *searah* jarum jam, dan kita mengukur arah tunjuk masing-masing sebagai sudut searah jarum jam dari angka 12. Jarum menit menempuh satu putaran penuh, 360°, dalam 60 menit, jadi ia berputar $360/60 = 6°$ per menit. Jarum jam menempuh satu putaran dalam 12 jam, $360/12 = 30°$ per jam — tetapi ia tidak melompat dari satu angka ke angka berikutnya tepat pada pergantian jam; ia merayap maju terus-menerus, sehingga setelah m menit ia telah bergeser tambahan $\frac{m}{60}$ jam. Dengan h sebagai jam dan m sebagai menit:

$$\theta_{\text{jam}} = 30\left(h \bmod 12 + \frac{m}{60}\right) \tag{4.3}$$

$$\theta_{\text{menit}} = 6m \tag{4.4}$$

Persamaan (4.3) adalah jarum jam dan Persamaan (4.4) jarum menit. Di sini h mod 12 adalah sisa pembagian h oleh 12, yang mengubah pukul 18 menjadi pukul 6. Sekarang arahnya. Bergerak *berlawanan* arah jarum jam dari jarum jam berarti bergerak *mundur* pada muka jam, sehingga sudut dari jarum jam ke jarum menit adalah posisi jarum jam *dikurangi* posisi jarum menit. Selisih itu bisa negatif, dan bisa berjarak sampai 360° dari rentang yang kita inginkan, jadi kita membawanya ke [0,360) dengan mengambil modulo 360, dan baru sesudah itu membulatkannya:

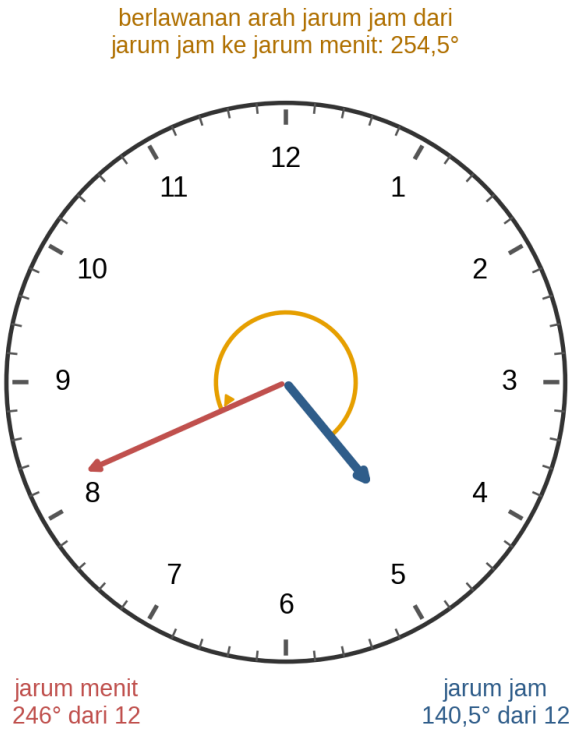
$$\alpha = \text{round}((\theta_{\text{jam}} - \theta_{\text{menit}}) \bmod 360) \bmod 360 \tag{4.5}$$

mod 360 yang terakhir dalam Persamaan (4.5) menangkap satu kasus langka yang akan kita temui di bawah. **Tabel 4.2** menerapkan Persamaan (4.3) sampai Persamaan (4.5) pada keenam waktu uji, satu kolom per langkah.

Tabel 4.2 Keenam waktu uji yang dikerjakan dengan tangan: jarum jam, jarum menit, selisih keduanya, selisih yang dibawa ke [0,360), dan hasil yang dibulatkan.

Waktu	θ_{jam}	θ_{menit}	Selisih	mod 360	Hasil
9:00	$30 \cdot 9 = 270$	0	270	270	270
3:00	$30 \cdot 3 = 90$	0	90	90	90
18:00	$30 \cdot 6 = 180$	0	180	180	180
1:00	$30 \cdot 1 = 30$	0	30	30	30
2:30	$30 \cdot 2.5 = 75$	180	-105	255	255
4:41	$120 + 20.5 = 140.5$	246	-105.5	254.5	254

Baris terakhir adalah yang paling menarik (**Gambar 4.3**). Pada pukul 4:41 sudut eksaknya 254.5°, tepat di tengah-tengah antara 254 dan 255, dan jawaban yang diharapkan adalah 254. Jadi "dibulatkan" di sini tidak mungkin berarti aturan sekolah "setengah selalu dibulatkan ke atas", yang akan memberi 255. Artinya adalah *pembulatan setengah ke genap* (*round half to even*): nilai yang tepat di tengah antara dua bilangan bulat dibawa ke bilangan yang *genap*. Inilah pembulatan bawaan standar titik-mengambang IEEE 754 [4], inilah yang dilakukan fungsi bawaan `round` di Python [5], dan di Java fungsinya adalah `Math.rint` [6]. Apakah nilai setengah sering muncul? Ya: selisihnya adalah $30h - 5.5m$, dan setiap kali m ganjil, hasilnya berakhir dengan .5. Jadi separuh menit dalam setiap jam adalah kasus seri, dan aturan pembulatan terus-menerus berperan.



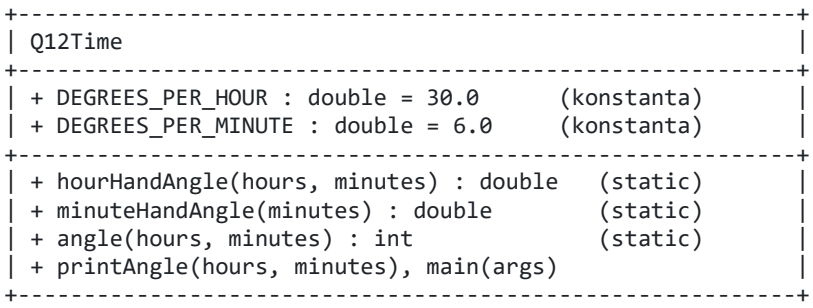
Gambar 4.3 Jam pada pukul 4:41: jarum jam telah merayap ke 140.5°, jarum menit menunjuk 246°, dan sudut berlawanan arah jarum jam di antara keduanya 254.5°, yang dibulatkan setengah ke genap menjadi 254.

Dua contoh tepi buatan kita sendiri. Pada pukul 2:11, $\theta_{\text{jam}} = 60 + 5.5 = 65.5$ dan $\theta_{\text{menit}} = 66$; selisihnya -0.5, modulo 360 menjadi 359.5, dan 359.5 dibulatkan (setengah ke genap) menjadi 360 — yang berada di luar 0..359. mod 360 yang terakhir mengubahnya menjadi 0, jawaban

yang benar, karena kedua jarum hampir berimpit. Pada pukul 23:59, $\theta_{\text{jam}} = 30 \cdot \left(11 + \frac{59}{60}\right) = 359.5$ dan $\theta_{\text{menit}} = 354$, selisihnya 5.5, dan 5.5 dibulatkan setengah ke genap menjadi 6.

4.4.3 Kelas, field, dan metode

Lagi-lagi tidak ada objek yang memiliki ingatan, hanya perhitungan, yang dipecah menjadi tiga besaran dari Persamaan (4.3) sampai Persamaan (4.5). Dua konstanta bernama menggantikan "bilangan ajaib" 30 dan 6.



4.4.4 Penyelesaian Java

Kode 4.6 adalah kelasnya, dengan penalaran di atas dituliskan ke dalam komentarnya, seperti yang diminta soal.

Kode 4.6 `Q12Time.java`: sudut berlawanan arah jarum jam dari jarum jam ke jarum menit.

```java
/**
 * Soal 4.2: sudut yang diukur berlawanan arah jarum jam dari jarum jam
 * ke jarum menit pada jam analog.
 * Jarum jam berputar 360/12 = 30 derajat per jam, dan ikut bergeser
 * seiring berlalunya menit; jarum menit berputar 360/60 = 6 derajat per menit.
 * Kedua sudut diukur searah jarum jam dari angka 12, sehingga sudut berlawanan
 * arah jarum jam dari jarum jam ke jarum menit adalah (sudut jam - sudut menit).
 */
public class Q12Time {
    public static final double DEGREES_PER_HOUR = 30.0;
    public static final double DEGREES_PER_MINUTE = 6.0;

    public static double hourHandAngle(int hours, int minutes) {
        return DEGREES_PER_HOUR * (hours % 12 + minutes / 60.0);
    }

    public static double minuteHandAngle(int minutes) {
        return DEGREES_PER_MINUTE * minutes;
    }

    public static int angle(int hours, int minutes) {
        double difference = hourHandAngle(hours, minutes) - minuteHandAngle(minutes);
        double normalized = (difference % 360 + 360) % 360; // 0 <= normalized < 360
        int rounded = (int) Math.rint(normalized);          // nilai tepat setengah ke
tetangga genap
        return rounded % 360;                               // 359.5 dibulatkan ke 360,
yaitu 0
    }

    public static void printAngle(int hours, int minutes) {
        System.out.println("At " + hours + ":" + (minutes < 10 ? "0" : "") + minutes
                + " the hour hand is at " + hourHandAngle(hours, minutes)
                + ", the minute hand at " + minuteHandAngle(minutes)
                + "; the angle is " + angle(hours, minutes) + " degrees.");
    }

    public static void main(String[] args) {
        printAngle(9, 0);
        printAngle(3, 0);
        printAngle(18, 0);
        printAngle(1, 0);
        printAngle(2, 30);
        printAngle(4, 41);
    }
}
```

```
At 9:00 the hour hand is at 270.0, the minute hand at 0.0; the angle is 270 degrees.
At 3:00 the hour hand is at 90.0, the minute hand at 0.0; the angle is 90 degrees.
At 18:00 the hour hand is at 180.0, the minute hand at 0.0; the angle is 180 degrees.
At 1:00 the hour hand is at 30.0, the minute hand at 0.0; the angle is 30 degrees.
At 2:30 the hour hand is at 75.0, the minute hand at 180.0; the angle is 255 degrees.
At 4:41 the hour hand is at 140.5, the minute hand at 246.0; the angle is 254 degrees.
```

Posisi jarum yang tercetak adalah kolom kedua dan ketiga **Tabel 4.2**, dan sudut-sudutnya adalah kolom terakhir. **Kode 4.7** memeriksa keenam waktu yang diberikan dan ketiga waktu tepi.

Kode 4.7 `Q12TimeTest.java`: keenam waktu yang diberikan dan tiga kasus tepi.

```java
/** Driver uji sederhana untuk Q12Time: jalankan dengan "java Q12TimeTest". */
public class Q12TimeTest {
    private static int passed = 0;
    private static int failed = 0;

    private static void check(String label, Object expected, Object actual) {
        if (expected.equals(actual)) {
            passed++;
            System.out.println("PASS " + label);
        } else {
            failed++;
            System.out.println("FAIL " + label + ": expected " + expected + ", got " +
actual);
        }
    }

    public static void main(String[] args) {
        check("9:00", 270, Q12Time.angle(9, 0));
        check("3:00", 90, Q12Time.angle(3, 0));
        check("18:00", 180, Q12Time.angle(18, 0));
        check("1:00", 30, Q12Time.angle(1, 0));
        check("2:30", 255, Q12Time.angle(2, 30));
        check("4:41", 254, Q12Time.angle(4, 41));
        check("0:00", 0, Q12Time.angle(0, 0));
        check("2:11", 0, Q12Time.angle(2, 11));
        check("23:59", 6, Q12Time.angle(23, 59));
        System.out.println(passed + " passed, " + failed + " failed");
    }
}
```

```
PASS 9:00
PASS 3:00
PASS 18:00
PASS 1:00
PASS 2:30
PASS 4:41
PASS 0:00
PASS 2:11
PASS 23:59
9 passed, 0 failed
```

4.4.5 Penyelesaian Python

Kode 4.8 `q12_time.py`: Soal 4.2 dalam Python, tempat `%` dan `round` sudah berperilaku seperti yang dituntut Persamaan (4.5).

```
"""Soal 4.2: sudut yang diukur berlawanan arah jarum jam dari jarum jam
ke jarum menit pada jam analog (penalarannya ada di Q12Time.java).
"""
import unittest

DEGREES_PER_HOUR = 30.0
DEGREES_PER_MINUTE = 6.0


def hour_hand_angle(hours, minutes):
    return DEGREES_PER_HOUR * (hours % 12 + minutes / 60)


def minute_hand_angle(minutes):
    return DEGREES_PER_MINUTE * minutes


def angle(hours, minutes):
    difference = hour_hand_angle(hours, minutes) - minute_hand_angle(minutes)
    normalized = difference % 360   # % di Python sudah memberi 0 <= normalized < 360
    return round(normalized) % 360  # round() mengirim nilai setengah ke tetangga genap


def print_angle(hours, minutes):
    print(f"At {hours}:{minutes:02d} the hour hand is at {hour_hand_angle(hours, minutes)},
"
          f"the minute hand at {minute_hand_angle(minutes)}; "
          f"the angle is {angle(hours, minutes)} degrees.")


class TestQ12Time(unittest.TestCase):
    def test_given_times(self):
        self.assertEqual(angle(9, 0), 270)
        self.assertEqual(angle(3, 0), 90)
        self.assertEqual(angle(18, 0), 180)
        self.assertEqual(angle(1, 0), 30)
        self.assertEqual(angle(2, 30), 255)
        self.assertEqual(angle(4, 41), 254)

    def test_edges(self):
        self.assertEqual(angle(0, 0), 0)
        self.assertEqual(angle(2, 11), 0)
        self.assertEqual(angle(23, 59), 6)
```

Kode 4.8 (Lanjutan) *q12_time.py: Soal 4.2 dalam Python, tempat % dan round sudah berperilaku seperti yang dituntut Persamaan (4.5).*

```
if __name__ == "__main__":
    for h, m in [(9, 0), (3, 0), (18, 0), (1, 0), (2, 30), (4, 41)]:
        print_angle(h, m)
    unittest.main(verbosity=2)
```

```
At 9:00 the hour hand is at 270.0, the minute hand at 0.0; the angle is 270 degrees.
At 3:00 the hour hand is at 90.0, the minute hand at 0.0; the angle is 90 degrees.
At 18:00 the hour hand is at 180.0, the minute hand at 0.0; the angle is 180 degrees.
At 1:00 the hour hand is at 30.0, the minute hand at 0.0; the angle is 30 degrees.
At 2:30 the hour hand is at 75.0, the minute hand at 180.0; the angle is 255 degrees.
At 4:41 the hour hand is at 140.5, the minute hand at 246.0; the angle is 254 degrees.
test_edges (__main__.TestQ12Time.test_edges) ... ok
test_given_times (__main__.TestQ12Time.test_given_times) ... ok

----------------------------------------------------------------------
Ran 2 tests in 0.000s

OK
```

Kode 4.8 lebih pendek daripada **Kode 4.6** di satu tempat, dan alasannya layak diketahui.

4.4.6 Baris demi baris

- `minutes / 60.0` di Java — `.0` itu menjadikan pembagiannya pembagian `double`. Menulis `minutes / 60` akan membagi dua nilai `int` dan menghasilkan `0` untuk setiap menit di bawah 60, sehingga jarum jam membeku tepat di angka jamnya. Di Python `/` selalu membagi secara eksak, jadi `minutes / 60` benar di sana.
- `(difference % 360 + 360) % 360` — di Java, operator sisa `%` mempertahankan tanda operan *kiri*: `-105.0 % 360` adalah `-105.0` [7]. Menambahkan 360 membuatnya positif, dan `% 360` yang kedua membuang 360 tambahan itu lagi bila nilainya memang sudah positif. Di Python, `%` dengan operan kanan positif selalu menghasilkan nilai antara 0 dan pembaginya [5], sehingga `difference % 360` saja sudah `255.0`. Satu perbedaan antara kedua bahasa inilah yang menyebabkan lebih banyak jawaban salah daripada apa pun dalam bab ini.
- `(int) Math.rint(normalized)` — `Math.rint` mengembalikan bilangan bulat terdekat dengan argumennya, memilih yang genap bila seri, sebagai `double` seperti `254.0`; *cast* `(int)` mengubahnya menjadi `int` 254. `round(normalized)` di Python mengerjakan keduanya sekaligus dan mengembalikan `int`.
- `return rounded % 360;` — kasus 2:11: `359.5` dibulatkan menjadi `360`, dan `360 % 360` adalah `0`.
- `(minutes < 10 ? "0" : "") + minutes` — *operator kondisional* `kondisi ? a : b` dari Bab 3 memilih `"0"` atau tidak apa-apa, sehingga pukul sembilan tercetak sebagai `9:00` dan bukan `9:0`. Python menulis hal yang sama sebagai format `{minutes:02d}`: sekurang-kurangnya dua angka, diisi nol di depan.
- `hours % 12` sebenarnya tidak mutlak diperlukan, karena modulo 360 terakhir akan menyerap tambahan $12 \cdot 30 = 360$ derajat. Kita mempertahankannya karena ia menyatakan kebenaran fisiknya — muka jam memiliki 12 jam — dan membuat posisi jarum yang tercetak pada pukul 18:00 menjadi `180.0`, bukan `540.0`.

4.4.7 Penyelesaian keliru yang menggoda

Kode 4.9 memperlihatkan tiga kekeliruan yang paling menggoda pada pukul 4:41, yang jawaban benarnya 254.

Kode 4.9 `WrongClockAngle.java`: tiga kekeliruan menggoda pada sudut jam pukul 4:41.

```java
/** Tiga versi sudut jam 4:41 yang menggoda tetapi salah (yang benar: 254). */
public class WrongClockAngle {
    public static void main(String[] args) {
        int hours = 4, minutes = 41;
        double difference = 30 * (hours % 12 + minutes / 60.0) - 6 * minutes;
        System.out.println("difference:        " + difference);
        System.out.println("no +360:           " + Math.round(difference) % 360);
        System.out.println("Math.round:        " + (Math.round(difference) + 360) % 360);
        int frozen = 30 * (hours % 12 + minutes / 60) - 6 * minutes;
        System.out.println("minutes / 60 int:  " + (frozen % 360 + 360) % 360);
    }
}
```

```
difference:        -105.5
no +360:           -105
Math.round:        255
minutes / 60 int:  234
```

Melupakan `+ 360` meninggalkan sudut negatif, -105. Memakai `Math.round`, yang membulatkan setengah *ke atas* (ke arah tak hingga positif), mengubah -105.5 menjadi -105 dan akhirnya memberi 255, meleset satu derajat. Dan menulis `minutes / 60` dengan dua nilai `int` membekukan jarum jam di 120°, menghasilkan 234. **Kode 4.10** adalah versi Python-nya, dan ia memuat peringatan dari jenis yang lain.

Kode 4.10 `wrong_clock_angle.py`: di Python, membulatkan sebelum menormalisasi kebetulan memberi jawaban yang benar.

```python
"""Tiga versi sudut jam 4:41 (yang benar: 254); satu di antaranya benar hanya karena
kebetulan."""
import math

hours, minutes = 4, 41
difference = 30 * (hours % 12 + minutes / 60) - 6 * minutes
print(f"difference:        {difference}")
print(f"round, then %:     {round(difference) % 360}")
print(f"round half up:     {math.floor(difference + 0.5) % 360}")
frozen = 30 * (hours % 12 + minutes // 60) - 6 * minutes
print(f"minutes // 60:     {frozen % 360}")
```

```
difference:        -105.5
round, then %:     254
round half up:     255
minutes // 60:     234
```

Baris kedua mencetak 254, jawaban yang benar — tetapi karena kebetulan: `round(-105.5)` adalah -106 karena -106 genap, dan $-106 \bmod 360 = 254$. Kode itu membulatkan *sebelum* menormalisasi, urutan yang salah; ia hanya sepakat dengan Persamaan (4.5) karena pergeseran sebesar 360 tidak mengubah tetangga mana yang genap. Program yang benar karena kebetulan adalah program yang akan salah setelah perubahan kecil berikutnya, dan karena itulah setiap baris sebuah penyelesaian harus memiliki alasan yang dapat Anda ucapkan dengan lantang.

> **UJI DIRI.** Tulislah metode `clockwiseAngle(hours, minutes)` yang mengukur sudut *searah* jarum jam dari jarum jam ke jarum menit. Berapa nilai yang dikembalikannya pada pukul 4:41 dan pukul 3:00?
>
> *Jawaban.* Searah jarum jam adalah arah sebaliknya pada muka jam, jadi sudutnya $\theta_{\text{menit}} - \theta_{\text{jam}}$, dinormalisasi dan dibulatkan dengan cara yang sama; setara dengan $(360 - \alpha) \bmod 360$ sebelum pembulatan. Pada pukul 4:41: $246 - 140.5 = 105.5$, yang dibulatkan setengah ke genap menjadi 106. Pada pukul 3:00: $0 - 90 = -90$, dinormalisasi menjadi 270. (Periksa: $254.5 + 105.5 = 360$ dan $90 + 270 = 360$.)

4.5 Soal 4.3 — Akun Komputer

4.5.1 Soalnya

> **SOAL 4.3.** Define a class `Q13ComputerAccount` and a constructor to create it. A computer account is built from three strings, `realName`, `userName` and `password`. It implements the methods `printRealName`, `printUserName`, `printPassword` and `changePassword`. The first three take no arguments. The last takes one argument of type `String`, the new password, and sets the password to it; its return type is `void`. Comment and test the program appropriately, in a separate class `Q13ComputerAccountTest`.

Dengan kata lain: definisikan kelas `Q13ComputerAccount` beserta konstruktornya. Sebuah akun dibangun dari tiga string — nama asli, nama pengguna, dan kata sandi — dan menyediakan metode `printRealName`, `printUserName`, `printPassword` (ketiganya tanpa argumen) serta `changePassword`, yang menerima kata sandi baru bertipe `String`, menggantinya, dan tidak mengembalikan apa-apa (`void`). Beri komentar yang layak dan ujilah di kelas terpisah `Q13ComputerAccountTest`.

4.5.2 Memahaminya dengan tangan

Inilah soal pertama dengan objek yang sungguhan: sesuatu yang *mengingat*. Akun untuk John Smith, nama pengguna `JXS`, kata sandi `sesame`, menyimpan ketiga string itu selama objeknya hidup. Bila kemudian kita memanggil `changePassword("senoJ yraM")`, pemanggilan `printPassword()` sesudahnya harus mencetak kata sandi yang baru, bukan yang lama. Dan akun kedua, untuk Mary Jones, memiliki tiga string *miliknya sendiri*: mengganti kata sandi John tidak boleh menyentuh milik Mary. Di atas kertas, jalan cerita uji ini adalah tabel keadaan kecil:

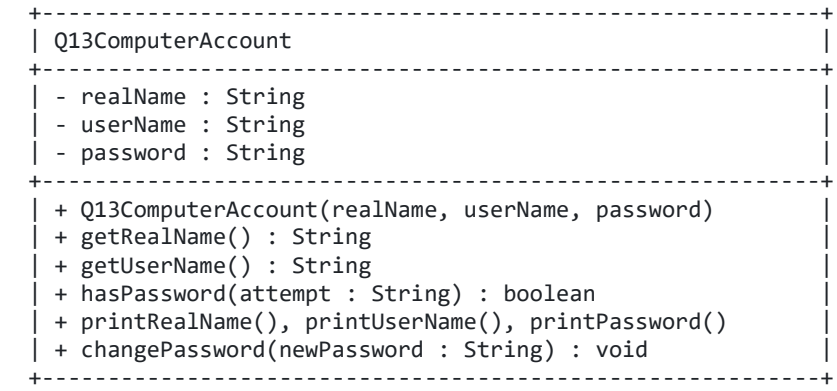
Tabel 4.3 Kata sandi dua akun setelah setiap langkah uji; baris terakhir memperlihatkan bahwa setiap objek memiliki *field*-nya sendiri.

Langkah	Kata sandi John	Kata sandi Mary
buat John dengan `sesame`	`sesame`	(Mary belum ada)
`john.changePassword("senoJ yraM")`	`senoJ yraM`	(Mary belum ada)
buat Mary dengan `senoJ yraM`	`senoJ yraM`	`senoJ yraM`
`john.changePassword("sesame")`	`sesame`	`senoJ yraM`

Baris terakhir **Tabel 4.3** adalah inti dari objek: satu kelas, banyak objek yang saling bebas.

4.5.3 Kelas, field, dan metode

Kata benda: *akun* (kelasnya), *nama asli*, *nama pengguna*, *kata sandi* (tiga *field* `String`). Kata kerja: *mencetak* setiap *field*, *mengganti* kata sandi. Kita menambahkan tiga metode kecil yang tidak diminta soal tetapi dibutuhkan uji: `getRealName` dan `getUserName` mengembalikan nilai sebuah *field* agar uji dapat membandingkannya, dan `hasPassword(attempt)` menjawab "inikah kata sandinya?" dengan `true` atau `false`, sehingga kita dapat menguji `changePassword` tanpa *membaca* kata sandinya.



Field-nya privat (tanda minus): satu-satunya jalan masuk adalah melalui metode. Itulah *enkapsulasi* (*encapsulation*), dan itulah sebabnya `hasPassword` ada — sengaja tidak ada `getPassword`.

4.5.4 Penyelesaian Java

Kode 4.11 adalah kelasnya dan **Kode 4.12** driver ujinya.

Kode 4.11 `Q13ComputerAccount.java`: akun dengan *field* privat dan cara terkendali untuk memeriksa kata sandi.

```java
/**
 * Soal 4.3: akun komputer yang dibangun dari nama asli, nama pengguna,
 * dan kata sandi. Field-nya privat, sehingga satu-satunya cara membaca atau
 * mengubahnya adalah melalui metode di bawah ini.
 */
public class Q13ComputerAccount {
    private String realName;
    private String userName;
    private String password;

    public Q13ComputerAccount(String realName, String userName, String password) {
        this.realName = realName;
        this.userName = userName;
        this.password = password;
    }

    public String getRealName() {
        return realName;
    }

    public String getUserName() {
        return userName;
    }

    // Menjawab "inikah kata sandinya?" tanpa pernah menyerahkan kata sandi itu.
    public boolean hasPassword(String attempt) {
        return password.equals(attempt);
    }

    public void printRealName() {
        System.out.println(realName);
    }

    public void printUserName() {
        System.out.println(userName);
    }

    // Diminta oleh soal; sistem sungguhan tidak akan pernah mencetak kata sandi.
    public void printPassword() {
        System.out.println(password);
    }

    public void changePassword(String newPassword) {
        password = newPassword;
    }
}
```

Kode 4.12 `Q13ComputerAccountTest.java`: jalan cerita **Tabel 4.3** sebagai pemeriksaan otomatis.

```java
/** Driver uji sederhana untuk Q13ComputerAccount: jalankan dengan "java
Q13ComputerAccountTest". */
public class Q13ComputerAccountTest {
    private static int passed = 0;
    private static int failed = 0;

    private static void check(String label, Object expected, Object actual) {
        if (expected.equals(actual)) {
            passed++;
            System.out.println("PASS " + label);
        } else {
            failed++;
            System.out.println("FAIL " + label + ": expected " + expected + ", got " +
actual);
        }
    }

    public static void main(String[] args) {
        Q13ComputerAccount john = new Q13ComputerAccount("John Smith", "JXS", "sesame");
        john.printRealName();
        john.printUserName();
        john.printPassword();
        john.changePassword("senoJ yraM");
        john.printPassword();

        check("real name", "John Smith", john.getRealName());
        check("user name", "JXS", john.getUserName());
        check("new password accepted", true, john.hasPassword("senoJ yraM"));
        check("old password rejected", false, john.hasPassword("sesame"));

        Q13ComputerAccount mary = new Q13ComputerAccount("Mary Jones", "MYJ", "senoJ
yraM");
        check("second account is separate", "MYJ", mary.getUserName());
        john.changePassword("sesame");
        check("changing John leaves Mary alone", true, mary.hasPassword("senoJ yraM"));
        String typed = new String("sesame");
        check("equals compares characters", true, john.hasPassword(typed));
        System.out.println(passed + " passed, " + failed + " failed");
    }
}
```

Kompilasi keduanya dengan `javac Q13ComputerAccount.java Q13ComputerAccountTest.java` dan jalankan driver-nya dengan `java Q13ComputerAccountTest`:

```
John Smith
JXS
sesame
senoJ yraM
PASS real name
PASS user name
PASS new password accepted
PASS old password rejected
PASS second account is separate
PASS changing John leaves Mary alone
PASS equals compares characters
7 passed, 0 failed
```

Keempat baris cetakan berasal dari keempat pemanggilan `print…`; pemeriksaannya mengikuti **Tabel 4.3** baris demi baris.

4.5.5 Penyelesaian Python

Kode 4.13 mengikuti kotak kelas yang sama; di Python, getter dan kelas ujinya tinggal di satu berkas.

Kode 4.13 `q13_computer_account.py`: Soal 4.3 dalam Python.

```
"""Soal 4.3: akun komputer yang dibangun dari nama asli, nama pengguna,
dan kata sandi. Garis bawah di depan menandai setiap atribut sebagai privat
menurut konvensi: kode di luar kelas sebaiknya memakai metodenya.
"""
import unittest


class Q13ComputerAccount:
    def __init__(self, real_name, user_name, password):
        self._real_name = real_name
        self._user_name = user_name
        self._password = password

    def get_real_name(self):
        return self._real_name

    def get_user_name(self):
        return self._user_name

    def has_password(self, attempt):
        # Menjawab "inikah kata sandinya?" tanpa pernah menyerahkan kata sandi itu.
        return self._password == attempt

    def print_real_name(self):
        print(self._real_name)

    def print_user_name(self):
        print(self._user_name)

    def print_password(self):
        # Diminta oleh soal; sistem sungguhan tidak akan pernah mencetak kata sandi.
        print(self._password)

    def change_password(self, new_password):
        self._password = new_password


class TestQ13ComputerAccount(unittest.TestCase):
    def setUp(self):
        self.john = Q13ComputerAccount("John Smith", "JXS", "sesame")

    def test_names(self):
        self.assertEqual(self.john.get_real_name(), "John Smith")
        self.assertEqual(self.john.get_user_name(), "JXS")
```

Kode 4.13 (Lanjutan) *q13_computer_account.py: Soal 4.3 dalam Python.*

```
    def test_change_password(self):
        self.john.change_password("senoJ yraM")
        self.assertTrue(self.john.has_password("senoJ yraM"))
        self.assertFalse(self.john.has_password("sesame"))

    def test_accounts_are_separate(self):
        mary = Q13ComputerAccount("Mary Jones", "MYJ", "senoJ yraM")
        self.john.change_password("open")
        self.assertTrue(mary.has_password("senoJ yraM"))

    def test_typed_text_compares_by_characters(self):
        typed = "".join(["ses", "ame"])
        self.assertTrue(self.john.has_password(typed))


if __name__ == "__main__":
    john = Q13ComputerAccount("John Smith", "JXS", "sesame")
    john.print_real_name()
    john.print_user_name()
    john.print_password()
    john.change_password("senoJ yraM")
    john.print_password()
    unittest.main(verbosity=2)
```

```
John Smith
JXS
sesame
senoJ yraM
test_accounts_are_separate (__main__.TestQ13ComputerAccount.test_accounts_are_separate) ...
ok
test_change_password (__main__.TestQ13ComputerAccount.test_change_password) ... ok
test_names (__main__.TestQ13ComputerAccount.test_names) ... ok
test_typed_text_compares_by_characters
(__main__.TestQ13ComputerAccount.test_typed_text_compares_by_characters) ... ok

----------------------------------------------------------------------
Ran 4 tests in 0.000s

OK
```

4.5.6 Baris demi baris

- `private String realName;` — setiap objek mendapat salinannya sendiri dari ketiga *field*; `private` berarti hanya kode di dalam `Q13ComputerAccount` yang boleh menyebut namanya.
- `this.realName = realName;` — di dalam konstruktor, parameter `realName` menutupi *field* yang bernama sama. `this.realName` berarti "*field* milik objek yang sedang dibangun", sehingga baris ini menyalin parameter ke dalam *field*. Melupakan `this.` adalah bug senyap yang klasik: `realName = realName;` memberikan parameter kepada dirinya sendiri, dan *field*-nya tetap `null`.
- `return password.equals(attempt);` — `equals` membandingkan dua string karakter demi karakter. Inilah satu-satunya cara yang benar untuk membandingkan *isi* string di Java; lihat penyelesaian keliru di bawah.
- `public void changePassword(String newPassword) { password = newPassword; }` — di sini `this.` tidak diperlukan, karena tidak ada parameter yang bernama `password`.
- `String typed = new String("sesame");` dalam uji sengaja membuat objek string *kedua* dengan karakter yang sama, seperti yang terjadi ketika teks diketik di papan ketik atau dibaca dari berkas. Pemeriksaan itu membuktikan bahwa `hasPassword` membandingkan karakter, bukan objek.

- Di Python, *field* dibuat dengan penugasan di dalam `__init__`, metode yang dipanggil Python ketika sebuah objek dibuat. Garis bawah di depan `self._password` berkata "privat, mohon dihormati" — sebuah *konvensi* yang tidak dipaksakan Python (Soal 4.7 memperlihatkan artinya). Metode `setUp` pada kelas uji berjalan sebelum *setiap* uji, sehingga setiap uji dimulai dengan John yang baru.

Dua catatan rancangan. Pertama, soal meminta `printPassword`, maka kita menyediakannya; sistem sungguhan tidak akan pernah mencetak kata sandi, dan orang bisa saja menjadikan metode itu privat. Kedua, `changePassword` di sini tidak menanyakan kata sandi lama, karena soal tidak memintanya; *uji diri* menambahkan pengaman itu.

4.5.7 Penyelesaian keliru yang menggoda

Jalan pintas yang menggoda adalah membandingkan string dengan `==`, seperti membandingkan bilangan. **Kode 4.14** memperlihatkan mengapa cara itu gagal.

Kode 4.14 `WrongPasswordCheck.java`: `==` membandingkan identitas objek, `equals` membandingkan karakter.

```
/** Membandingkan string dengan == bertanya "objek yang sama?", bukan "karakter yang
sama?". */
public class WrongPasswordCheck {
    public static void main(String[] args) {
        String stored = "sesame";
        String typed = new String("sesame"); // seolah-olah diketik di papan ketik
        System.out.println("typed == stored:      " + (typed == stored));
        System.out.println("typed.equals(stored): " + typed.equals(stored));
    }
}
```

```
typed == stored:      false
typed.equals(stored): true
```

Untuk objek, `==` bertanya "apakah keduanya *objek yang sama*?" [7]. `stored` dan `typed` adalah dua objek berbeda yang kebetulan memuat enam karakter yang sama, sehingga `==` menjawab `false`, dan pengguna yang mengetik kata sandi yang benar justru ditolak. Lebih buruk lagi, `==` kadang-kadang *tampak* berhasil — ketika kedua string adalah literal yang ditulis di dalam program, Java memakai satu objek bersama untuk keduanya — sehingga bug ini bersembunyi selama uji sekilas dan baru muncul dengan masukan sungguhan. `==` di Python membandingkan isi, dan `is`-nya adalah uji identitas, seperti diperlihatkan **Kode 4.15**.

Kode 4.15 `wrong_password_check.py`: di Python, `is` adalah uji identitas dan `==` membandingkan isi.

```
"""Membandingkan string dengan 'is' bertanya "objek yang sama?", bukan "karakter yang
sama?"."""
stored = "sesame"
typed = "".join(["ses", "ame"])  # seolah-olah diketik di papan ketik
print(f"typed is stored: {typed is stored}")
print(f"typed == stored: {typed == stored}")
```

```
typed is stored: False
typed == stored: True
```

> **UJI DIRI.** Ubahlah `changePassword` sehingga menerima dua argumen, kata sandi lama dan kata sandi baru, hanya mengganti kata sandi bila yang lama benar, dan mengembalikan `true` atau `false` untuk menyatakan apakah penggantian terjadi. Apa yang dikembalikan

> `changePassword("wrong", "x")` lalu `changePassword("sesame", "x")` untuk akun baru dengan kata sandi `sesame`?
>
> *Jawaban.* Di Java: `public boolean changePassword(String oldPassword, String newPassword) { if (!password.equals(oldPassword)) { return false; } password = newPassword; return true; }`. Pemanggilan pertama mengembalikan `false` dan membiarkan kata sandi tetap `sesame`; yang kedua mengembalikan `true` dan kata sandinya menjadi `x`. Di Python badannya adalah `if self._password != old_password: return False`, lalu penugasannya dan `return True`.

4.6 Soal 4.4 — Bilangan Kompleks

4.6.1 Soalnya

> **SOAL 4.4.** Define a class `Q14ComplexNumber` with a constructor and getter methods. A `Q14ComplexNumber` represents a complex number with a real and an imaginary part. Implement methods to (1) add two complex numbers, (2) subtract two complex numbers, (3) multiply two complex numbers, and (4) convert the complex number to a string `"a + bi"`, where a is the real and b the imaginary part. The operations are $(a + bi) + (c + di) = (a + c) + (b + d)i$, $(a + bi) - (c + di) = (a - c) + (b - d)i$ and Persamaan (4.6). Test the class in `Q14ComplexNumberTest`.

Dengan kata lain: definisikan kelas `Q14ComplexNumber` dengan konstruktor dan metode getter, yang menyatakan bilangan kompleks dengan bagian riil dan bagian imajiner. Sediakan metode untuk menjumlahkan, mengurangkan, dan mengalikan dua bilangan kompleks, serta mengubah bilangan itu menjadi string berbentuk `"a + bi"`, lalu ujilah di `Q14ComplexNumberTest`.

$$(a + bi)(c + di) = (ac - bd) + (ad + bc)i \tag{4.6}$$

4.6.2 Memahaminya dengan tangan

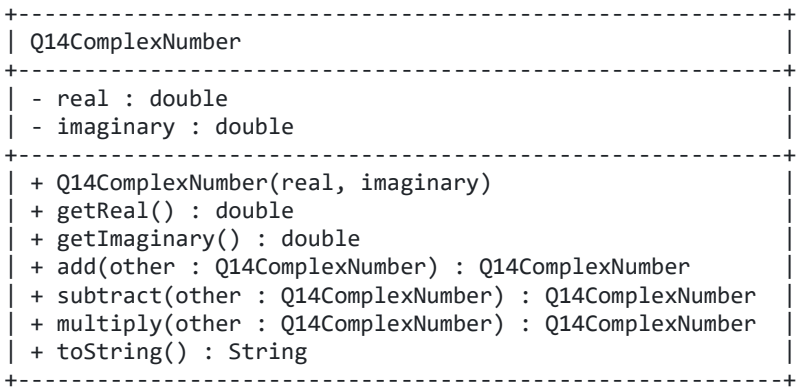
Bilangan kompleks adalah sepasang bilangan biasa, *bagian riil* a dan *bagian imajiner* b, yang ditulis $a + bi$, dengan i adalah bilangan yang kuadratnya -1. Anda tidak perlu tahu matematika lebih dari itu: ketiga rumus sudah memberi tahu dengan tepat apa yang harus dihitung. Persamaan (4.6) diperoleh dengan menguraikan tanda kurung seperti biasa — $ac + adi + bci + bd\,i^2$ — lalu mengganti i^2 dengan -1, yang memindahkan bd ke bagian riil dengan tanda minus.

Ambillah $c_1 = 1 + 2i$ dan $c_2 = 3 + 4i$:

- jumlah: $(1 + 3) + (2 + 4)i = 4 + 6i$;
- selisih: $(1 - 3) + (2 - 4)i = -2 - 2i$;
- hasil kali: bagian riil $1 \cdot 3 - 2 \cdot 4 = 3 - 8 = -5$, bagian imajiner $1 \cdot 4 + 2 \cdot 3 = 4 + 6 = 10$, jadi $-5 + 10i$;
- contoh tepi: $i \cdot i$, yaitu $(0 + 1i)(0 + 1i)$: bagian riil $0 \cdot 0 - 1 \cdot 1 = -1$, bagian imajiner $0 \cdot 1 + 1 \cdot 0 = 0$, jadi $-1 + 0i$, seperti yang dituntut definisi i.

Selisih itu memunculkan pertanyaan yang tidak dijawab pernyataan soal: haruskah $-2 - 2i$ dicetak sebagai `-2.0 + -2.0i`, yang mengikuti pola `"a + bi"` secara harfiah, atau sebagai `-2.0 - 2.0i`, seperti yang ditulis siapa pun? Kita memilih yang kedua, dengan sebuah `if`, dan uji kita menyatakannya.

4.6.3 Kelas, field, dan metode



Keputusan rancangan yang utama ada pada tipe kembaliannya: `add` menerima bilangan kompleks *lain* dan mengembalikan bilangan kompleks yang *baru*. Metode ini tidak mengubah `this`, dan tidak mengubah `other`. Dengan demikian `c1.add(c2)` berperilaku seperti $c_1 + c_2$ dalam matematika, tempat penjumlahan tidak mengubah bilangan-bilangan yang dijumlahkan.

4.6.4 Penyelesaian Java

Kode 4.16 adalah kelasnya; **Kode 4.17** memeriksa keempat contoh tangan dan bahwa `c1` tidak berubah setelah dipakai.

Kode 4.16 `Q14ComplexNumber.java`: bilangan kompleks yang operasi-operasinya mengembalikan objek baru.

```java
/**
 * Soal 4.4: bilangan kompleks a + bi dengan bagian riil a dan bagian imajiner b.
 *   (a + bi) + (c + di) = (a + c) + (b + d)i
 *   (a + bi) - (c + di) = (a - c) + (b - d)i
 *   (a + bi) * (c + di) = (ac - bd) + (ad + bc)i
 * Setiap operasi mengembalikan objek baru dan membiarkan kedua operan tidak berubah.
 */
public class Q14ComplexNumber {
    private double real;
    private double imaginary;

    public Q14ComplexNumber(double real, double imaginary) {
        this.real = real;
        this.imaginary = imaginary;
    }

    public double getReal() {
        return real;
    }

    public double getImaginary() {
        return imaginary;
    }

    public Q14ComplexNumber add(Q14ComplexNumber other) {
        return new Q14ComplexNumber(real + other.real, imaginary + other.imaginary);
    }

    public Q14ComplexNumber subtract(Q14ComplexNumber other) {
        return new Q14ComplexNumber(real - other.real, imaginary - other.imaginary);
    }

    public Q14ComplexNumber multiply(Q14ComplexNumber other) {
        double newReal = real * other.real - imaginary * other.imaginary;
        double newImaginary = real * other.imaginary + imaginary * other.real;
        return new Q14ComplexNumber(newReal, newImaginary);
    }

    @Override
    public String toString() {
        if (imaginary < 0) {
            return real + " - " + (-imaginary) + "i";
        }
        return real + " + " + imaginary + "i";
    }
}
```

Kode 4.17 `Q14ComplexNumberTest.java`: keempat contoh tangan dan pemeriksaan bahwa operan-operannya tidak berubah.

```java
/** Driver uji sederhana untuk Q14ComplexNumber: jalankan dengan "java
Q14ComplexNumberTest". */
public class Q14ComplexNumberTest {
    private static int passed = 0;
    private static int failed = 0;

    private static void check(String label, Object expected, Object actual) {
        if (expected.equals(actual)) {
            passed++;
            System.out.println("PASS " + label);
        } else {
            failed++;
            System.out.println("FAIL " + label + ": expected " + expected + ", got " +
actual);
        }
    }

    public static void main(String[] args) {
        Q14ComplexNumber c1 = new Q14ComplexNumber(1, 2);
        Q14ComplexNumber c2 = new Q14ComplexNumber(3, 4);
        System.out.println("Sum: (" + c1 + ") + (" + c2 + ") = " + c1.add(c2));
        System.out.println("Difference: (" + c1 + ") - (" + c2 + ") = " + c1.subtract(c2));
        System.out.println("Product: (" + c1 + ") * (" + c2 + ") = " + c1.multiply(c2));

        check("sum", "4.0 + 6.0i", c1.add(c2).toString());
        check("difference", "-2.0 - 2.0i", c1.subtract(c2).toString());
        check("product", "-5.0 + 10.0i", c1.multiply(c2).toString());
        check("i * i = -1", "-1.0 + 0.0i",
                new Q14ComplexNumber(0, 1).multiply(new Q14ComplexNumber(0,
1)).toString());
        check("operand unchanged", "1.0 + 2.0i", c1.toString());
        check("getReal", 3.0, c2.getReal());
        check("getImaginary", 4.0, c2.getImaginary());
        System.out.println(passed + " passed, " + failed + " failed");
    }
}
```

```
Sum: (1.0 + 2.0i) + (3.0 + 4.0i) = 4.0 + 6.0i
Difference: (1.0 + 2.0i) - (3.0 + 4.0i) = -2.0 - 2.0i
Product: (1.0 + 2.0i) * (3.0 + 4.0i) = -5.0 + 10.0i
PASS sum
PASS difference
PASS product
PASS i * i = -1
PASS operand unchanged
PASS getReal
PASS getImaginary
7 passed, 0 failed
```

4.6.5 Penyelesaian Python

Dalam **Kode 4.18**, `__str__` memainkan peran `toString`.

Kode 4.18 `q14_complex_number.py`: Soal 4.4 dalam Python, tempat `__str__` memainkan peran `toString`.

```python
"""Soal 4.4: bilangan kompleks a + bi (rumusnya seperti di Q14ComplexNumber.java).
Setiap operasi mengembalikan objek baru dan membiarkan kedua operan tidak berubah.
"""
import unittest


class Q14ComplexNumber:
    def __init__(self, real, imaginary):
        self._real = float(real)            # float, agar 1 tercetak sebagai 1.0, seperti di Java
        self._imaginary = float(imaginary)

    def get_real(self):
        return self._real

    def get_imaginary(self):
        return self._imaginary

    def add(self, other):
        return Q14ComplexNumber(self._real + other._real, self._imaginary + other._imaginary)

    def subtract(self, other):
        return Q14ComplexNumber(self._real - other._real, self._imaginary - other._imaginary)

    def multiply(self, other):
        new_real = self._real * other._real - self._imaginary * other._imaginary
        new_imaginary = self._real * other._imaginary + self._imaginary * other._real
        return Q14ComplexNumber(new_real, new_imaginary)

    def __str__(self):
        if self._imaginary < 0:
            return f"{self._real} - {-self._imaginary}i"
        return f"{self._real} + {self._imaginary}i"


class TestQ14ComplexNumber(unittest.TestCase):
    def setUp(self):
        self.c1 = Q14ComplexNumber(1, 2)
        self.c2 = Q14ComplexNumber(3, 4)

    def test_add(self):
        self.assertEqual(str(self.c1.add(self.c2)), "4.0 + 6.0i")

    def test_subtract(self):
        self.assertEqual(str(self.c1.subtract(self.c2)), "-2.0 - 2.0i")
```

Kode 4.18 (Lanjutan) *q14_complex_number.py: Soal 4.4 dalam Python, tempat __str__ memainkan peran toString.*

```
    def test_multiply(self):
        self.assertEqual(str(self.c1.multiply(self.c2)), "-5.0 + 10.0i")

    def test_i_squared(self):
        i = Q14ComplexNumber(0, 1)
        self.assertEqual(str(i.multiply(i)), "-1.0 + 0.0i")

    def test_operands_unchanged(self):
        self.c1.add(self.c2)
        self.assertEqual(str(self.c1), "1.0 + 2.0i")
        self.assertEqual(self.c2.get_real(), 3.0)
        self.assertEqual(self.c2.get_imaginary(), 4.0)


if __name__ == "__main__":
    c1 = Q14ComplexNumber(1, 2)
    c2 = Q14ComplexNumber(3, 4)
    print(f"Sum: ({c1}) + ({c2}) = {c1.add(c2)}")
    print(f"Difference: ({c1}) - ({c2}) = {c1.subtract(c2)}")
    print(f"Product: ({c1}) * ({c2}) = {c1.multiply(c2)}")
    unittest.main(verbosity=2)
```

```
Sum: (1.0 + 2.0i) + (3.0 + 4.0i) = 4.0 + 6.0i
Difference: (1.0 + 2.0i) - (3.0 + 4.0i) = -2.0 - 2.0i
Product: (1.0 + 2.0i) * (3.0 + 4.0i) = -5.0 + 10.0i
test_add (__main__.TestQ14ComplexNumber.test_add) ... ok
test_i_squared (__main__.TestQ14ComplexNumber.test_i_squared) ... ok
test_multiply (__main__.TestQ14ComplexNumber.test_multiply) ... ok
test_operands_unchanged (__main__.TestQ14ComplexNumber.test_operands_unchanged) ... ok
test_subtract (__main__.TestQ14ComplexNumber.test_subtract) ... ok

----------------------------------------------------------------------
Ran 5 tests in 0.000s

OK
```

4.6.6 Baris demi baris

- `return new Q14ComplexNumber(real + other.real, imaginary + other.imaginary);` — `other.real` membaca *field privat* milik objek lain. Itu diperbolehkan, karena di Java sifat privat dimiliki oleh *kelas*, bukan oleh objek: kode di dalam `Q14ComplexNumber` boleh melihat *field* privat setiap `Q14ComplexNumber`.
- Dalam `multiply`, dua variabel lokal `newReal` dan `newImaginary` menampung kedua belahan Persamaan (4.6); masing-masing dihitung dari nilai-nilai *lama*, dan baru sesudah itu objek baru dibangun.
- `@Override public String toString()` — setiap objek Java memiliki metode `toString`, yang dipanggil Java setiap kali objek itu disambungkan ke string dengan `+`, seperti dalam `"Sum: " + c1`. Menulis versi kita sendiri menggantikan versi bawaan, yang akan mencetak sesuatu seperti `Q14ComplexNumber@1b6d3586`. `@Override` meminta kompilator memeriksa bahwa kita memang sedang menggantikan metode yang sudah ada.
- `if (imaginary < 0) { return real + " - " + (-imaginary) + "i"; }` — untuk bagian imajiner yang negatif, cetak tanda minus dan nilai *positif*-nya. Tanda kurung di sekitar `-imaginary` penting: tanpa tanda kurung, `" - " + -imaginary` tetap berjalan, tetapi pembaca harus memikirkannya.

- Di Python, `float(real)` di dalam `__init__` membuat `Q14ComplexNumber(1, 2)` menyimpan `1.0` dan `2.0`, sehingga Python mencetak `1.0 + 2.0i` persis seperti Java. Metode `__str__` adalah yang dipanggil `print` dan f-string, kembaran `toString` di Python.

4.6.7 Penyelesaian keliru yang menggoda

Kode 4.19 memperlihatkan dua kekeliruan yang menggoda: mengalikan bagian demi bagian, dan mencetak tanda secara naif.

Kode 4.19 `WrongComplex.java`: mengalikan bagian demi bagian, dan mencetak bagian negatif secara naif.

```
/** Mengalikan bagian demi bagian melupakan i * i = -1; pencetakan naif menampilkan "+ -".
*/
public class WrongComplex {
    public static void main(String[] args) {
        double a = 1, b = 2, c = 3, d = 4;
        System.out.println("part by part: " + (a * c) + " + " + (b * d) + "i");
        System.out.println("correct:      " + (a * c - b * d) + " + " + (a * d + b * c) +
"i");
        double re = a - c, im = b - d;
        System.out.println("naive text:   " + re + " + " + im + "i");
    }
}
```

```
part by part: 3.0 + 8.0i
correct:      -5.0 + 10.0i
naive text:   -2.0 + -2.0i
```

Mengalikan riil dengan riil dan imajiner dengan imajiner terasa wajar, karena begitulah cara penjumlahan bekerja — tetapi cara itu melupakan suku silang ad dan bc serta kenyataan bahwa $i \cdot i = -1$, sehingga hasilnya $3 + 8i$, bukan $-5 + 10i$. String naif menampilkan `-2.0 + -2.0i`, yang secara matematis tidak salah tetapi buruk rupa. **Kode 4.20** mengulang keduanya dalam Python.

Kode 4.20 `wrong_complex.py`: dua kekeliruan yang sama dalam Python.

```
"""Mengalikan bagian demi bagian melupakan i * i = -1; pencetakan naif menampilkan "+ -
"."""
a, b, c, d = 1.0, 2.0, 3.0, 4.0
print(f"part by part: {a * c} + {b * d}i")
print(f"correct:      {a * c - b * d} + {a * d + b * c}i")
re, im = a - c, b - d
print(f"naive text:   {re} + {im}i")
```

```
part by part: 3.0 + 8.0i
correct:      -5.0 + 10.0i
naive text:   -2.0 + -2.0i
```

> **UJI DIRI.** Tambahkan metode `conjugate()` yang mengembalikan $a - bi$ untuk $a + bi$. Apa yang dicetak `c1.multiply(c1.conjugate())` untuk $c_1 = 1 + 2i$, dan mengapa bagian imajinernya nol?
>
> *Jawaban.* `conjugate` mengembalikan `new Q14ComplexNumber(real, -imaginary)`. Maka $(1 + 2i)(1 - 2i)$ memiliki bagian riil $1 \cdot 1 - 2 \cdot (-2) = 1 + 4 = 5$ dan bagian imajiner $1 \cdot (-2) + 2 \cdot 1 = 0$, sehingga tercetak `5.0 + 0.0i`. Secara umum $(a + bi)(a - bi) = a^2 + b^2$, karena suku silang $-abi$ dan $+abi$ saling meniadakan.

4.7 Soal 4.5 — Konversi Berat

4.7.1 Soalnya

> **SOAL 4.5.** Chapter 3 converted masses given in the imperial system into the metric system. Write a program that deals with weights given in pounds, using Persamaan (4.7). Define a class `Q15Weight` with a constructor `Q15Weight(double p)` that creates a weight given in pounds, and methods `getPounds` and `getKilograms`, which return the weight in pounds and in kilograms. Test the class in `Q15WeightTest`.

Dengan kata lain: Bab 3 mengubah massa dari sistem imperial ke sistem metrik. Kini tulislah program untuk berat yang diberikan dalam pon (*pound*): kelas `Q15Weight` dengan konstruktor `Q15Weight(double p)` yang membuat berat dalam pon, serta metode `getPounds` dan `getKilograms` yang mengembalikan berat itu dalam pon dan dalam kilogram. Ujilah di `Q15WeightTest`.

$$1 \text{ pon} = 0.45359237 \text{ kilogram} \tag{4.7}$$

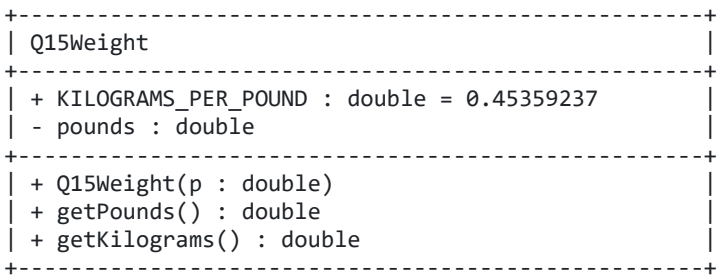
4.7.2 Memahaminya dengan tangan

Sebuah objek `Q15Weight` mengingat satu berat. Ditanya pon, ia mengembalikan apa yang diberikan kepadanya; ditanya kilogram, ia mengalikannya dengan 0.45359237. Faktor dalam Persamaan (4.7) itu eksak berdasarkan kesepakatan internasional, bukan hasil pengukuran, jadi kita menuliskan kedelapan angkanya. Dengan tangan:

- $150 \text{ lb} \times 0.45359237 = 68.0388555 \text{ kg}$;
- $140 \text{ lb} \times 0.45359237 = 63.5029318 \text{ kg}$;
- orang dalam pengubah massa di Bab 3, 11 *stone* dan 6 pon, beratnya $11 \cdot 14 + 6 = 160$ pon, jadi $160 \times 0.45359237 = 72.5747792$ kg.

Satu-satunya pertanyaan rancangan: haruskah objek menyimpan kedua bilangan, pon dan kilogram? Tidak. Bila keduanya disimpan, suatu hari metode yang mengubah salah satunya bisa lupa mengubah yang lain, dan objek itu akan bertentangan dengan dirinya sendiri. Objek hanya menyimpan pon dan *menurunkan* kilogram setiap kali diminta.

4.7.3 Kelas, field, dan metode



4.7.4 Penyelesaian Java

Kode 4.21 adalah seluruh kelasnya: satu konstanta, satu *field*, sebuah konstruktor, dan dua getter. **Kode 4.22** mengujinya dengan ketiga berat yang dikerjakan dengan tangan.

Kode 4.21 `Q15Weight.java`: berat yang disimpan dalam pon, dengan kilogram yang dihitung saat diminta.

```java
/**
 * Soal 4.5: berat yang diberikan dalam pon. Hanya pon yang disimpan;
 * kilogram dihitung saat diminta, sehingga keduanya tidak mungkin berselisih.
 */
public class Q15Weight {
    public static final double KILOGRAMS_PER_POUND = 0.45359237;

    private double pounds;

    public Q15Weight(double p) {
        pounds = p;
    }

    public double getPounds() {
        return pounds;
    }

    public double getKilograms() {
        return pounds * KILOGRAMS_PER_POUND;
    }
}
```

Kode 4.22 `Q15WeightTest.java`: hasil titik-mengambang dibandingkan dengan toleransi.

```java
/** Driver uji sederhana untuk Q15Weight: jalankan dengan "java Q15WeightTest". */
public class Q15WeightTest {
    private static int passed = 0;
    private static int failed = 0;

    private static void checkClose(String label, double expected, double actual) {
        if (Math.abs(expected - actual) < 1e-9) {
            passed++;
            System.out.println("PASS " + label);
        } else {
            failed++;
            System.out.println("FAIL " + label + ": expected " + expected + ", got " +
actual);
        }
    }

    public static void main(String[] args) {
        Q15Weight john = new Q15Weight(150);
        Q15Weight mary = new Q15Weight(140);
        Q15Weight stones = new Q15Weight(11 * 14 + 6);
        System.out.println("John: " + john.getPounds() + " lb = " + john.getKilograms() + "
kg");
        System.out.println("Mary: " + mary.getPounds() + " lb = " + mary.getKilograms() + "
kg");
        System.out.println("11 st 6 lb: " + stones.getPounds() + " lb = " +
stones.getKilograms() + " kg");

        checkClose("150 lb in pounds", 150.0, john.getPounds());
        checkClose("150 lb in kg", 68.0388555, john.getKilograms());
        checkClose("140 lb in kg", 63.5029318, mary.getKilograms());
        checkClose("1 lb in kg", 0.45359237, new Q15Weight(1).getKilograms());
        checkClose("0 lb in kg", 0.0, new Q15Weight(0).getKilograms());
        System.out.println(passed + " passed, " + failed + " failed");
    }
}
```

```
John: 150.0 lb = 68.0388555 kg
Mary: 140.0 lb = 63.502931800000006 kg
11 st 6 lb: 160.0 lb = 72.57477920000001 kg
PASS 150 lb in pounds
PASS 150 lb in kg
PASS 140 lb in kg
PASS 1 lb in kg
PASS 0 lb in kg
5 passed, 0 failed
```

Perhatikan baris Mary: `63.502931800000006`, bukan `63.5029318`. Tidak ada yang salah. Sebuah `double` menyimpan bilangan dalam biner, dan 0.45359237, seperti 0.1, tidak memiliki bentuk biner yang eksak, sehingga hasil kalinya membawa galat kecil pada angka penting keenam belas [4]. Inilah sebabnya driver uji memakai `checkClose`, yang menerima selisih lebih kecil dari 10^{-9}, alih-alih `equals`.

4.7.5 Penyelesaian Python

Kode 4.23 memakai `assertAlmostEqual`, padanan `checkClose` dalam `unittest`.

Kode 4.23 `q15_weight.py`: Soal 4.5 dalam Python, dengan `assertAlmostEqual` untuk pemeriksaan titik-mengambang.

```python
"""Soal 4.5: berat yang diberikan dalam pon. Hanya pon yang disimpan;
kilogram dihitung saat diminta, sehingga keduanya tidak mungkin berselisih.
"""
import unittest


class Q15Weight:
    KILOGRAMS_PER_POUND = 0.45359237

    def __init__(self, p):
        self._pounds = float(p)

    def get_pounds(self):
        return self._pounds

    def get_kilograms(self):
        return self._pounds * Q15Weight.KILOGRAMS_PER_POUND


class TestQ15Weight(unittest.TestCase):
    def test_pounds_kept(self):
        self.assertEqual(Q15Weight(150).get_pounds(), 150.0)

    def test_kilograms(self):
        self.assertAlmostEqual(Q15Weight(150).get_kilograms(), 68.0388555, places=9)
        self.assertAlmostEqual(Q15Weight(140).get_kilograms(), 63.5029318, places=9)
        self.assertAlmostEqual(Q15Weight(1).get_kilograms(), 0.45359237, places=9)
        self.assertEqual(Q15Weight(0).get_kilograms(), 0.0)


if __name__ == "__main__":
    john = Q15Weight(150)
    mary = Q15Weight(140)
    stones = Q15Weight(11 * 14 + 6)
    print(f"John: {john.get_pounds()} lb = {john.get_kilograms()} kg")
    print(f"Mary: {mary.get_pounds()} lb = {mary.get_kilograms()} kg")
    print(f"11 st 6 lb: {stones.get_pounds()} lb = {stones.get_kilograms()} kg")
    unittest.main(verbosity=2)
```

```
John: 150.0 lb = 68.0388555 kg
Mary: 140.0 lb = 63.502931800000006 kg
11 st 6 lb: 160.0 lb = 72.57477920000001 kg
test_kilograms (__main__.TestQ15Weight.test_kilograms) ... ok
test_pounds_kept (__main__.TestQ15Weight.test_pounds_kept) ... ok

----------------------------------------------------------------------
Ran 2 tests in 0.000s

OK
```

Python mencetak angka-angka yang persis sama dengan Java, karena keduanya memakai `double` IEEE 754 yang sama [4] dan keduanya mencetak desimal terpendek yang bila dibaca kembali menghasilkan bilangan yang sama.

4.7.6 Baris demi baris

- `public static final double KILOGRAMS_PER_POUND = 0.45359237;` — `static` berarti satu salinan untuk seluruh kelas, `final` berarti nilainya tidak pernah dapat berubah. Konstanta diberi nama dengan huruf kapital dan garis bawah, sehingga pembaca langsung tahu bahwa nilainya tetap. Di Python, variabel yang ditugaskan langsung di badan kelas, `KILOGRAMS_PER_POUND = 0.45359237`, memainkan peran yang sama; huruf kapital kembali memberi isyarat "jangan diubah".
- `public Q15Weight(double p) { pounds = p; }` — Java mengubah `int` 150 dalam `new Q15Weight(150)` menjadi `double` 150.0 secara otomatis, sebuah *konversi pelebaran* (*widening conversion*) yang tidak pernah kehilangan informasi.
- `return pounds * KILOGRAMS_PER_POUND;` — kilogram dihitung ulang pada setiap pemanggilan, yang hanya membutuhkan satu perkalian, praktis tanpa biaya.
- `Math.abs(expected - actual) < 1e-9` — `1e-9` adalah notasi ilmiah untuk 10^{-9}. Membandingkan dua nilai `double` hasil perhitungan dengan `==` hampir selalu merupakan kesalahan; membandingkan jarak keduanya dengan toleransi kecil adalah obat bakunya. `assertAlmostEqual(a, b, places=9)` di Python membulatkan selisihnya sampai sembilan angka desimal dan memeriksa bahwa hasilnya nol.

4.7.7 Penyelesaian keliru yang menggoda

Kode 4.24 menyimpan kedua satuan dan menambahkan setter yang melupakan salah satunya.

Kode 4.24 `WrongWeight.java`: dua *field* yang bisa saling tertinggal.

```java
/** Menyimpan kilogram sebagai field kedua membuatnya bisa tertinggal dari pon. */
public class WrongWeight {
    private double pounds;
    private double kilograms;

    public WrongWeight(double p) {
        pounds = p;
        kilograms = p * 0.45359237;
    }

    public void setPounds(double p) {
        pounds = p; // lupa memperbarui kilogram
    }

    public static void main(String[] args) {
        WrongWeight w = new WrongWeight(150);
        w.setPounds(140);
        System.out.println(w.pounds + " lb = " + w.kilograms + " kg ?");
    }
}
```

```
140.0 lb = 68.0388555 kg ?
```

Objek itu kini mengklaim bahwa 140 pon sama dengan 68.04 kilogram, yang merupakan nilai untuk 150 pon. Tidak ada kompilator yang memperingatkan hal ini; hanya uji yang mengatur lalu membaca yang dapat menangkapnya. **Kode 4.25** membuat kesalahan yang sama dalam Python.

Kode 4.25 `wrong_weight.py`: nilai kedua yang basi, kali ini dalam Python.

```python
"""Menyimpan kilogram sebagai atribut kedua membuatnya bisa tertinggal dari pon."""


class WrongWeight:
    def __init__(self, p):
        self.pounds = p
        self.kilograms = p * 0.45359237

    def set_pounds(self, p):
        self.pounds = p  # lupa memperbarui kilogram


w = WrongWeight(150.0)
w.set_pounds(140.0)
print(f"{w.pounds} lb = {w.kilograms} kg ?")
```

```
140.0 lb = 68.0388555 kg ?
```

Aturan umumnya layak diingat: *simpan setiap fakta satu kali saja, dan hitunglah segala sesuatu yang mengikutinya*.

> **UJI DIRI.** Tambahkan metode `getStonesAndPounds()` yang mengembalikan string seperti `"11 st 6.0 lb"` untuk berat 160 pon (satu *stone* sama dengan 14 pon). Apa yang dikembalikannya untuk 150 pon?
>
> *Jawaban.* Jumlah *stone* utuh adalah `(int) (pounds / 14)`, dan sisanya `pounds - 14 * stones`; di Python `int(self._pounds // 14)` dan pengurangan yang sama. Untuk 150 pon: $150/14 = 10.71\ldots$, jadi 10 *stone*, dan $150 - 140 = 10$ pon: `"10 st 10.0 lb"`.

4.8 Soal 4.6 — Kelas Pecahan

4.8.1 Soalnya

> **SOAL 4.6.** Problem 4.1 added and multiplied two fractions given as loose numbers. Now represent fractions by a Java class `Q16Fraction`. Besides the constructor and the accessor methods `getNumerator` and `getDenominator`, the class has a method `toString` (to print a fraction) and methods `getSum` and `getProduct`, which produce the sum and the product of two fractions and return them as a `Q16Fraction`. For instance, after `Q16Fraction f1 = new Q16Fraction(1, 2);` and `Q16Fraction f2 = new Q16Fraction(3, 7);`, `f1.toString()` returns the string `"1/2"`, `System.out.print(f2.getProduct(f1));` prints `3/14`, and `System.out.print(f2.getSum(f1));` prints `13/14`. Test the class in `Q16FractionTest`.

Dengan kata lain: Soal 4.1 menjumlahkan dan mengalikan dua pecahan yang diberikan sebagai bilangan lepas. Sekarang nyatakan pecahan dengan kelas Java `Q16Fraction` yang memiliki konstruktor, metode akses `getNumerator` dan `getDenominator`, metode `toString`, serta metode `getSum` dan `getProduct` yang mengembalikan jumlah dan hasil kali sebagai objek `Q16Fraction` baru. Untuk $f_1 = \frac{1}{2}$ dan $f_2 = \frac{3}{7}$, keluaran yang diminta adalah `1/2`, `3/14`, dan `13/14`.

4.8.2 Memahaminya dengan tangan

Ini Soal 4.1 lagi, tetapi setiap pecahan kini adalah *satu benda*, sebuah objek yang mengenal pembilang dan penyebutnya sendiri. Rumusnya tetap Persamaan (4.1) dan Persamaan (4.2). Dengan $f_2 = \frac{3}{7}$ dan $f_1 = \frac{1}{2}$:

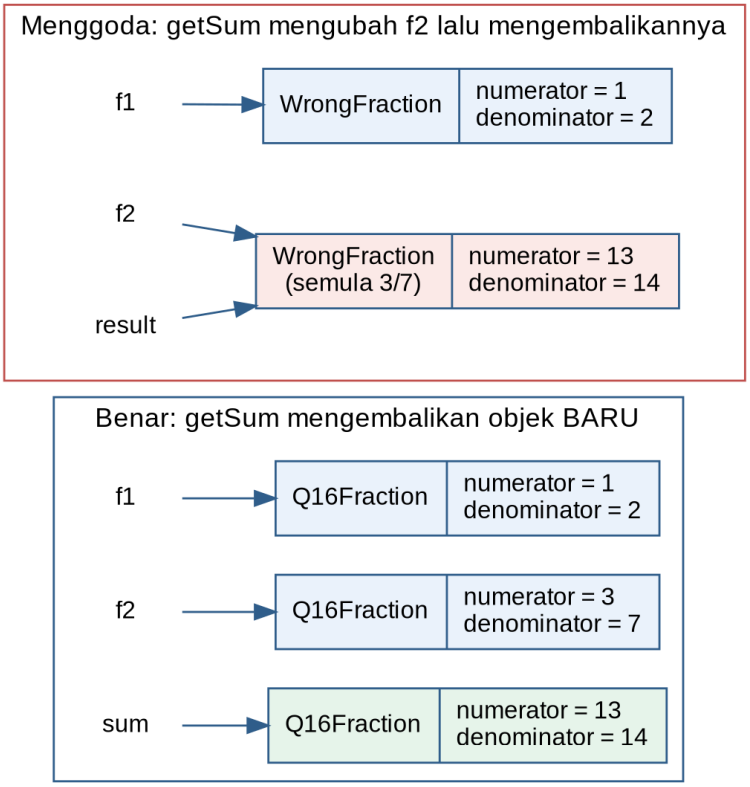
- `f2.getProduct(f1)`: $\frac{3 \cdot 1}{7 \cdot 2} = \frac{3}{14}$;
- `f2.getSum(f1)`: $\frac{3 \cdot 2 + 7 \cdot 1}{7 \cdot 2} = \frac{6+7}{14} = \frac{13}{14}$;
- setelah kedua pemanggilan, f_2 harus tetap $\frac{3}{7}$: meminta sebuah jumlah tidak boleh mengubah pecahan yang dimintai.

Karena hasil-hasilnya sendiri berupa pecahan, hasil itu dapat langsung dipakai lagi — inilah keuntungan sesungguhnya dibandingkan Soal 4.1. Untuk mengujinya, rangkaikan dua operasi: $(f_1 + f_1) \cdot (f_2 + f_2)$. Dengan tangan, $f_1 + f_1 = \frac{1 \cdot 2 + 2 \cdot 1}{2 \cdot 2} = \frac{4}{4}$ dan $f_2 + f_2 = \frac{3 \cdot 7 + 7 \cdot 3}{7 \cdot 7} = \frac{42}{49}$, dan hasil kali keduanya $\frac{4 \cdot 42}{4 \cdot 49} = \frac{168}{196}$. Pecahan yang tidak disederhanakan cepat membengkak; $\frac{168}{196}$ sebenarnya hanya $\frac{6}{7}$.

4.8.3 Kelas, field, dan metode

```
+-------------------------------------------------+
| Q16Fraction                                     |
+-------------------------------------------------+
| - numerator : int                               |
| - denominator : int                             |
+-------------------------------------------------+
| + Q16Fraction(numerator, denominator)           |
| + getNumerator() : int                          |
| + getDenominator() : int                        |
| + toString() : String                           |
| + getProduct(other : Q16Fraction) : Q16Fraction |
| + getSum(other : Q16Fraction) : Q16Fraction     |
+-------------------------------------------------+
```

Gambar 4.4 memperlihatkan, pada kotak bawahnya, apa yang terjadi di memori: `f1` dan `f2` merujuk ke dua objek, dan `f2.getSum(f1)` membuat objek *ketiga* untuk jumlahnya. Objek yang tidak pernah berubah setelah dibangun disebut *immutable* (tak dapat diubah); objek *immutable* lebih mudah dinalar dan dibagi-pakai, dan itulah sebabnya `String` milik Java sendiri *immutable* [8].



Gambar 4.4 Kotak bawah: `getSum` yang benar membiarkan `f1` dan `f2` dan mengembalikan objek baru. Kotak atas: versi menggoda dari Subbab 4.8.7 menimpa `f2` lalu mengembalikannya, sehingga `f2` dan hasilnya adalah objek yang sama yang telah berubah.

4.8.4 Penyelesaian Java

Kode 4.26 adalah kelasnya dan **Kode 4.27** driver-nya, yang mencetak ketiga keluaran yang diminta sebelum pemeriksaan-pemeriksaannya.

Kode 4.26 `Q16Fraction.java`: pecahan *immutable* yang operasi-operasinya mengembalikan pecahan baru.

```java
/**
 * Soal 4.6: pecahan pembilang/penyebut sebagai objek.
 * getSum dan getProduct membangun dan mengembalikan pecahan BARU; tidak satu
 * operan pun diubah. Hasilnya tidak disederhanakan.
 */
public class Q16Fraction {
    private int numerator;
    private int denominator;

    public Q16Fraction(int numerator, int denominator) {
        this.numerator = numerator;
        this.denominator = denominator;
    }

    public int getNumerator() {
        return numerator;
    }

    public int getDenominator() {
        return denominator;
    }

    @Override
    public String toString() {
        return numerator + "/" + denominator;
    }

    public Q16Fraction getProduct(Q16Fraction other) {
        return new Q16Fraction(numerator * other.getNumerator(),
                denominator * other.getDenominator());
    }

    public Q16Fraction getSum(Q16Fraction other) {
        return new Q16Fraction(numerator * other.getDenominator() + denominator *
other.getNumerator(),
                denominator * other.getDenominator());
    }
}
```

Kode 4.27 `Q16FractionTest.java`: ketiga keluaran yang diminta, imutabilitas, dan perhitungan berantai.

```java
/** Driver uji sederhana untuk Q16Fraction: jalankan dengan "java Q16FractionTest". */
public class Q16FractionTest {
    private static int passed = 0;
    private static int failed = 0;

    private static void check(String label, Object expected, Object actual) {
        if (expected.equals(actual)) {
            passed++;
            System.out.println("PASS " + label);
        } else {
            failed++;
            System.out.println("FAIL " + label + ": expected " + expected + ", got " +
actual);
        }
    }

    public static void main(String[] args) {
        Q16Fraction f1 = new Q16Fraction(1, 2);
        Q16Fraction f2 = new Q16Fraction(3, 7);
        System.out.println(f1);
        System.out.println(f2.getProduct(f1));
        System.out.println(f2.getSum(f1));

        check("toString", "1/2", f1.toString());
        check("product", "3/14", f2.getProduct(f1).toString());
        check("sum", "13/14", f2.getSum(f1).toString());
        check("f2 unchanged", "3/7", f2.toString());
        check("chained", "168/196", f1.getSum(f1).getProduct(f2.getSum(f2)).toString());
        check("getNumerator", 3, f2.getNumerator());
        check("getDenominator", 7, f2.getDenominator());
        System.out.println(passed + " passed, " + failed + " failed");
    }
}
```

```
1/2
3/14
13/14
PASS toString
PASS product
PASS sum
PASS f2 unchanged
PASS chained
PASS getNumerator
PASS getDenominator
7 passed, 0 failed
```

Tiga baris pertama persis merupakan keluaran yang dituntut soal: `System.out.println(f1)` memanggil `toString` secara otomatis.

4.8.5 Penyelesaian Python

Kode 4.28 adalah terjemahan baris demi baris.

Kode 4.28 `q16_fraction.py`: Soal 4.6 dalam Python.

```
"""Soal 4.6: pecahan pembilang/penyebut sebagai objek.
get_sum dan get_product membangun dan mengembalikan pecahan BARU; tidak satu
operan pun diubah. Hasilnya tidak disederhanakan.
"""
import unittest


class Q16Fraction:
    def __init__(self, numerator, denominator):
        self._numerator = numerator
        self._denominator = denominator

    def get_numerator(self):
        return self._numerator

    def get_denominator(self):
        return self._denominator

    def __str__(self):
        return f"{self._numerator}/{self._denominator}"

    def get_product(self, other):
        return Q16Fraction(self._numerator * other.get_numerator(),
                           self._denominator * other.get_denominator())

    def get_sum(self, other):
        return Q16Fraction(self._numerator * other.get_denominator()
                           + self._denominator * other.get_numerator(),
                           self._denominator * other.get_denominator())


class TestQ16Fraction(unittest.TestCase):
    def setUp(self):
        self.f1 = Q16Fraction(1, 2)
        self.f2 = Q16Fraction(3, 7)

    def test_str(self):
        self.assertEqual(str(self.f1), "1/2")

    def test_product_and_sum(self):
        self.assertEqual(str(self.f2.get_product(self.f1)), "3/14")
        self.assertEqual(str(self.f2.get_sum(self.f1)), "13/14")
```

Kode 4.28 (Lanjutan) *q16_fraction.py: Soal 4.6 dalam Python.*

```
    def test_operands_unchanged(self):
        self.f2.get_sum(self.f1)
        self.assertEqual(str(self.f2), "3/7")

    def test_chained(self):
        result = self.f1.get_sum(self.f1).get_product(self.f2.get_sum(self.f2))
        self.assertEqual(str(result), "168/196")


if __name__ == "__main__":
    f1 = Q16Fraction(1, 2)
    f2 = Q16Fraction(3, 7)
    print(f1)
    print(f2.get_product(f1))
    print(f2.get_sum(f1))
    unittest.main(verbosity=2)
```

```
1/2
3/14
13/14
test_chained (__main__.TestQ16Fraction.test_chained) ... ok
test_operands_unchanged (__main__.TestQ16Fraction.test_operands_unchanged) ... ok
test_product_and_sum (__main__.TestQ16Fraction.test_product_and_sum) ... ok
test_str (__main__.TestQ16Fraction.test_str) ... ok

----------------------------------------------------------------------
Ran 4 tests in 0.000s

OK
```

4.8.6 Baris demi baris

- `public Q16Fraction getProduct(Q16Fraction other)` — tipe parameter dan tipe kembaliannya sama-sama kelas buatan kita. Sebuah kelas adalah tipe seperti `int`, dan dapat dipakai di mana pun sebuah tipe dapat dipakai.
- `return new Q16Fraction(numerator * other.getNumerator(), denominator * other.getDenominator());` — ini terbaca sebagai Persamaan (4.2): `numerator` dan `denominator` adalah *field* objek ini, dan `other.getNumerator()` menanyakan pembilang kepada objek lain.
- *Field*-nya tidak memiliki setter. Soal 4.6 tidak memintanya, dan meniadakannya itulah yang membuat kelas ini *immutable*.
- `f1.getSum(f1).getProduct(f2.getSum(f2))` — Java mengevaluasi ini dari dalam ke luar dan dari kiri ke kanan: pertama `f1.getSum(f1)`, yang berupa objek, lalu `f2.getSum(f2)`, objek lain, lalu `getProduct` dari yang pertama dengan yang kedua. *Perangkaian* (*chaining*) seperti ini hanya mungkin karena setiap metode mengembalikan objek.
- Di Python, `str(self.f2.get_sum(self.f1))` memanggil `__str__` pada objek yang dikembalikan, dan begitulah uji-ujinya membandingkan teks `"13/14"`.

4.8.7 Penyelesaian keliru yang menggoda

Versi yang menggoda menghitung jumlahnya *di tempat*: ia menimpa *field* milik `this` lalu mengembalikan `this`. Pada pemanggilan pertama ia mencetak jawaban yang benar, dan justru itulah yang membuatnya berbahaya (**Kode 4.29**).

Kode 4.29 `WrongFraction.java`: `getSum` yang mengubah objek tempat ia dipanggil.

```java
/** getSum yang mengubah pecahan tempat ia dipanggil, alih-alih membuat pecahan baru. */
public class WrongFraction {
    private int numerator;
    private int denominator;

    public WrongFraction(int numerator, int denominator) {
        this.numerator = numerator;
        this.denominator = denominator;
    }

    public WrongFraction getSum(WrongFraction other) {
        numerator = numerator * other.denominator + denominator * other.numerator;
        denominator = denominator * other.denominator;
        return this;
    }

    @Override
    public String toString() {
        return numerator + "/" + denominator;
    }

    public static void main(String[] args) {
        WrongFraction f1 = new WrongFraction(1, 2);
        WrongFraction f2 = new WrongFraction(3, 7);
        System.out.println("f2 + f1 = " + f2.getSum(f1));
        System.out.println("f2 is now " + f2);
        System.out.println("f2 + f1 = " + f2.getSum(f1) + " (same call, new answer)");
    }
}
```

```
f2 + f1 = 13/14
f2 is now 13/14
f2 + f1 = 40/28 (same call, new answer)
```

Baris pertama benar. Baris kedua memperlihatkan bahwa `f2` sendiri telah menjadi $\frac{13}{14}$, seperti pada kotak atas **Gambar 4.4**. Baris ketiga memperlihatkan akibatnya: ekspresi yang persis sama, `f2.getSum(f1)`, kini menghasilkan $\frac{40}{28}$. Metode yang jawabannya bergantung pada berapa kali ia telah dipanggil adalah perangkap bagi setiap orang yang memakainya. **Kode 4.30** memperlihatkan bahwa Python berperilaku sama.

Kode 4.30 `wrong_fraction.py`: penjumlahan di tempat yang sama dalam Python.

```
"""get_sum yang mengubah pecahan tempat ia dipanggil, alih-alih membuat pecahan baru."""


class WrongFraction:
    def __init__(self, numerator, denominator):
        self.numerator = numerator
        self.denominator = denominator

    def get_sum(self, other):
        self.numerator = self.numerator * other.denominator + self.denominator *
other.numerator
        self.denominator = self.denominator * other.denominator
        return self

    def __str__(self):
        return f"{self.numerator}/{self.denominator}"


f1 = WrongFraction(1, 2)
f2 = WrongFraction(3, 7)
print(f"f2 + f1 = {f2.get_sum(f1)}")
print(f"f2 is now {f2}")
print(f"f2 + f1 = {f2.get_sum(f1)} (same call, new answer)")
```

```
f2 + f1 = 13/14
f2 is now 13/14
f2 + f1 = 40/28 (same call, new answer)
```

> **UJI DIRI.** Buatlah konstruktornya menyimpan setiap pecahan dalam bentuk paling sederhana, sehingga `new Q16Fraction(168, 196)` mencetak `6/7`. Pakailah faktor persekutuan terbesar, yang dihitung dengan algoritma Euclid memakai perulangan `while` (Soal 4.13 menelusurinya). Apa yang kemudian dicetak `f2.getSum(f2)` dan `new Q16Fraction(2, 6)`?
>
> *Jawaban.* Di dalam konstruktor, hitung `int g = gcd(Math.abs(numerator), denominator);` dengan `gcd(a, b)` = `while (b != 0) { int r = a % b; a = b; b = r; } return a;`, lalu simpan `numerator / g` dan `denominator / g` (bila `g` bernilai 0, yang hanya terjadi untuk $\frac{0}{0}$, simpan bilangannya tanpa perubahan). Maka $\frac{42}{49}$ menjadi $\frac{6}{7}$ karena $\gcd(42,49) = 7$, dan $\frac{2}{6}$ menjadi $\frac{1}{3}$ karena $\gcd(2,6) = 2$. Uji berantai dalam **Kode 4.27** kemudian harus mengharapkan `6/7`.

4.9 Soal 4.7 — Pemain dan Musuh

4.9.1 Soalnya

> **SOAL 4.7.** Define two opposing classes, `Player` and `Enemy`. Both have a name, health, power and defense. Define a constructor, getters and setters for these attributes. Implement in both classes: (a) `attack`, which takes the opposing class as its parameter; the opposing object then calls the method below with this object's attack; (b) `takeDamage`, which takes the opposing attack as its parameter and reduces this object's health by (opposing attack − this defense). If the health falls below zero, it prints `"{name} died!"`. Test both classes in `EnemyPlayerTest`.

Dengan kata lain: definisikan dua kelas yang saling berhadapan, `Player` dan `Enemy`, masing-masing dengan nama, kesehatan (*health*), kekuatan (*power*), dan pertahanan (*defense*), beserta konstruktor, getter, dan setter. Kedua kelas memiliki (a) `attack`, yang menerima objek lawan sebagai parameter dan meminta lawan itu memanggil metode berikut dengan kekuatan serangan objek ini; dan (b) `takeDamage`, yang menerima kekuatan serangan lawan dan mengurangi

kesehatan objek ini sebesar (serangan lawan − pertahanan objek ini). Bila kesehatannya jatuh di bawah nol, cetak `"{name} died!"`. Ujilah keduanya di `EnemyPlayerTest`.

4.9.2 Memahaminya dengan tangan

Dua objek saling berbicara. Ketika sang pahlawan menyerang sang penjahat, pahlawan itu *tidak* merogoh ke dalam penjahat dan menurunkan kesehatannya; ia memberi tahu penjahat "engkau terkena serangan berkekuatan 25", dan penjahat itulah yang menghitung kerusakannya dari pertahanannya *sendiri*. Pembagian kerja itu adalah jantung berorientasi objek dari soal ini: setiap objek bertanggung jawab atas *field*-nya sendiri.

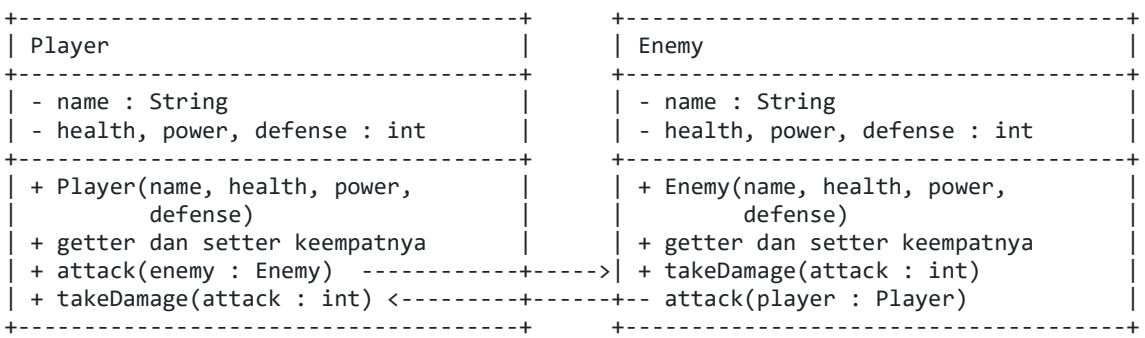
Ambillah pahlawan dengan kesehatan 100, kekuatan 25, pertahanan 10, dan penjahat dengan kesehatan 80, kekuatan 20, pertahanan 5. **Tabel 4.4** mengikuti jalannya pertarungan.

Tabel 4.4 Pertarungan Soal 4.7 yang dikerjakan dengan tangan: setiap serangan merugikan pihak bertahan sebesar (kekuatan penyerang − pertahanan pihak bertahan).

Aksi	Kerusakan	Kesehatan pahlawan	Kesehatan penjahat
awal		100	80
pahlawan menyerang penjahat	$25 - 5 = 20$	100	60
penjahat menyerang pahlawan	$20 - 10 = 10$	90	60
pahlawan menyerang penjahat (ronde 1)	20	90	40
pahlawan menyerang penjahat (ronde 2)	20	90	20
pahlawan menyerang penjahat (ronde 3)	20	90	0 — "Villain died!"

Dua pertanyaan dibiarkan terbuka oleh pernyataan soal. Pertama, bagaimana bila pertahanan lebih besar daripada serangan — sebuah "tank" dengan pertahanan 40 dipukul dengan kekuatan 25? Rumusnya memberi kerusakan $25 - 40 = -15$, dan "mengurangi kesehatan sebesar -15" justru akan *menyembuhkan* tank itu. Tidak ada permainan yang bekerja seperti itu, jadi kita memperlakukan kerusakan negatif sebagai nol. Kedua, pernyataan soal berkata "di bawah nol", padahal tokoh yang tersisa dengan kesehatan tepat 0, seperti penjahat kita, tentu juga sudah mati. Kita mencetak pesannya bila kesehatan *nol atau kurang*, dan menyatakannya dalam komentar. Membaca spesifikasi dengan cermat termasuk menyadari di mana spesifikasi itu diam atau sedikit meleset, lalu menuliskan keputusan Anda.

4.9.3 Kelas, field, dan metode



Kedua kelas itu saling bercermin. (Bab 7 akan memperlihatkan bagaimana *pewarisan* memungkinkan keduanya berbagi kode ini alih-alih mengulanginya; Bab 8 kembali ke pasangan yang persis sama ini.)

4.9.4 Penyelesaian Java

Kode 4.31 adalah `Player`. Berkas `Enemy.java` di folder kode adalah kelas yang sama dengan kata `Player` dan `Enemy` dipertukarkan (demikian pula `enemy` dan `player` pada parameter `attack`), jadi kita tidak mencetaknya dua kali.

Kode 4.31 `Player.java`: satu pihak dalam pertarungan; `Enemy.java` adalah bayangan cerminnya.

```
/**
 * Soal 4.7: satu pihak dari pertarungan dua pihak. Player menyerang Enemy;
 * Enemy sendiri yang menentukan seberapa sakit serangan itu.
 */
public class Player {
    private String name;
    private int health;
    private int power;
    private int defense;

    public Player(String name, int health, int power, int defense) {
        this.name = name;
        this.health = health;
        this.power = power;
        this.defense = defense;
    }

    public String getName() {
        return name;
    }

    public int getHealth() {
        return health;
    }

    public int getPower() {
        return power;
    }

    public int getDefense() {
        return defense;
    }

    public void setName(String name) {
        this.name = name;
    }

    public void setHealth(int health) {
        this.health = health;
    }

    public void setPower(int power) {
        this.power = power;
    }
```

Kode 4.31 (Lanjutan) *Player.java: satu pihak dalam pertarungan; Enemy.java adalah bayangan cerminnya.*

```
    public void setDefense(int defense) {
        this.defense = defense;
    }

    public void attack(Enemy enemy) {
        enemy.takeDamage(power);
    }

    public void takeDamage(int attack) {
        int damage = attack - defense;
        if (damage < 0) {
            damage = 0; // pertahanan yang kuat menahan pukulan; tidak pernah menyembuhkan
        }
        health = health - damage;
        if (health <= 0) {
            System.out.println(name + " died!");
        }
    }
}
```

Kode 4.32 memutar ulang **Tabel 4.4**.

Kode 4.32 `EnemyPlayerTest.java`: pertarungan **Tabel 4.4**, dengan perulangan `while` untuk ronde-ronde terakhir.

```java
/** Driver uji sederhana untuk Player dan Enemy: jalankan dengan "java EnemyPlayerTest". */
public class EnemyPlayerTest {
    private static int passed = 0;
    private static int failed = 0;

    private static void check(String label, Object expected, Object actual) {
        if (expected.equals(actual)) {
            passed++;
            System.out.println("PASS " + label);
        } else {
            failed++;
            System.out.println("FAIL " + label + ": expected " + expected + ", got " +
actual);
        }
    }

    public static void main(String[] args) {
        Player hero = new Player("Hero", 100, 25, 10);
        Enemy villain = new Enemy("Villain", 80, 20, 5);

        hero.attack(villain);
        check("villain loses 25 - 5 = 20", 60, villain.getHealth());
        villain.attack(hero);
        check("hero loses 20 - 10 = 10", 90, hero.getHealth());

        int round = 1;
        while (villain.getHealth() > 0) {
            hero.attack(villain);
            System.out.println("Round " + round + ": " + villain.getName()
                    + " has " + villain.getHealth() + " health left");
            round++;
        }
        check("villain ends at 0", 0, villain.getHealth());

        Enemy tank = new Enemy("Tank", 50, 5, 40);
        hero.attack(tank);
        check("defense 40 blocks attack 25", 50, tank.getHealth());
        System.out.println(passed + " passed, " + failed + " failed");
    }
}
```

```
PASS villain loses 25 - 5 = 20
PASS hero loses 20 - 10 = 10
Round 1: Villain has 40 health left
Round 2: Villain has 20 health left
Villain died!
Round 3: Villain has 0 health left
PASS villain ends at 0
PASS defense 40 blocks attack 25
4 passed, 0 failed
```

Perhatikan urutan baris di sekitar kematian sang penjahat: `Villain died!` dicetak *di dalam* pemanggilan `hero.attack(villain)` yang ketiga, sebelum perulangan mencetak `Round 3`. Urutan keluaran adalah catatan yang setia tentang urutan pemanggilan.

4.9.5 Penyelesaian Python

Kode 4.33 menyimpan kedua kelas di satu modul.

Kode 4.33 `player_enemy.py`: kedua kelas Soal 4.7 dalam satu modul Python.

```
"""Soal 4.7: dua pihak yang bertarung. Setiap pihak menyerang pihak lain;
pihak yang terkena sendiri yang menentukan seberapa sakit serangan itu.
"""
import unittest


class Player:
    def __init__(self, name, health, power, defense):
        self._name = name
        self._health = health
        self._power = power
        self._defense = defense

    def get_name(self):
        return self._name

    def get_health(self):
        return self._health

    def get_power(self):
        return self._power

    def get_defense(self):
        return self._defense

    def set_name(self, name):
        self._name = name

    def set_health(self, health):
        self._health = health

    def set_power(self, power):
        self._power = power

    def set_defense(self, defense):
        self._defense = defense

    def attack(self, enemy):
        enemy.take_damage(self._power)
```

Kode 4.33 (Lanjutan) *player_enemy.py: kedua kelas Soal 4.7 dalam satu modul Python.*

```
    def take_damage(self, attack):
        damage = attack - self._defense
        if damage < 0:
            damage = 0  # pertahanan yang kuat menahan pukulan; tidak pernah menyembuhkan
        self._health = self._health - damage
        if self._health <= 0:
            print(f"{self._name} died!")


class Enemy:
    def __init__(self, name, health, power, defense):
        self._name = name
        self._health = health
        self._power = power
        self._defense = defense

    def get_name(self):
        return self._name

    def get_health(self):
        return self._health

    def get_power(self):
        return self._power

    def get_defense(self):
        return self._defense

    def set_name(self, name):
        self._name = name

    def set_health(self, health):
        self._health = health

    def set_power(self, power):
        self._power = power

    def set_defense(self, defense):
        self._defense = defense

    def attack(self, player):
        player.take_damage(self._power)
```

Kode 4.33 (Lanjutan) *player_enemy.py: kedua kelas Soal 4.7 dalam satu modul Python.*

```
    def take_damage(self, attack):
        damage = attack - self._defense
        if damage < 0:
            damage = 0  # pertahanan yang kuat menahan pukulan; tidak pernah menyembuhkan
        self._health = self._health - damage
        if self._health <= 0:
            print(f"{self._name} died!")


class TestPlayerEnemy(unittest.TestCase):
    def test_exchange_of_blows(self):
        hero = Player("Hero", 100, 25, 10)
        villain = Enemy("Villain", 80, 20, 5)
        hero.attack(villain)
        self.assertEqual(villain.get_health(), 60)
        villain.attack(hero)
        self.assertEqual(hero.get_health(), 90)

    def test_defense_blocks(self):
        hero = Player("Hero", 100, 25, 10)
        tank = Enemy("Tank", 50, 5, 40)
        hero.attack(tank)
        self.assertEqual(tank.get_health(), 50)


if __name__ == "__main__":
    hero = Player("Hero", 100, 25, 10)
    villain = Enemy("Villain", 80, 20, 5)
    hero.attack(villain)
    villain.attack(hero)
    round_number = 1
    while villain.get_health() > 0:
        hero.attack(villain)
        print(f"Round {round_number}: {villain.get_name()} has {villain.get_health()}
health left")
        round_number += 1
    unittest.main(verbosity=2)
```

```
Round 1: Villain has 40 health left
Round 2: Villain has 20 health left
Villain died!
Round 3: Villain has 0 health left
test_defense_blocks (__main__.TestPlayerEnemy.test_defense_blocks) ... ok
test_exchange_of_blows (__main__.TestPlayerEnemy.test_exchange_of_blows) ... ok

----------------------------------------------------------------------
Ran 2 tests in 0.000s

OK
```

4.9.6 Baris demi baris

- `public void attack(Enemy enemy) { enemy.takeDamage(power); }` — seluruh metode ini adalah satu *pesan*: “terimalah kerusakan dari serangan sekuat ini”. Pemain tidak pernah menyentuh *field* milik musuh.
- `int damage = attack - defense; if (damage < 0) { damage = 0; }` — rumus dari pernyataan soal, lalu penjaga terhadap penyembuhan.
- `health = health - damage; if (health <= 0) { … }` — kurangi dulu, baru uji; menguji sebelum mengurangi akan mengumumkan kematian satu pukulan terlambat.

- `while (villain.getHealth() > 0)` di dalam driver — perulangan itu tidak tahu sebelumnya berapa pukulan yang dibutuhkan; ia mengulang *selama* kondisinya terpenuhi, persis jenis situasi yang oleh Bab 3 dianjurkan memakai `while` alih-alih `for`.
- Di Python kedua kelas tinggal di satu berkas. Python tidak mengharuskan satu kelas per berkas; di sini cara itu sekadar praktis karena kedua kelas selalu bepergian bersama.

4.9.7 Penyelesaian keliru yang menggoda

Menggoda sekali membiarkan `attack` mengerjakan semuanya: `villain.health = villain.health - power;`. **Kode 4.34** mencoba sesuatu yang lebih nekat lagi — ia langsung mengatur kesehatan musuh menjadi nol dari luar.

Kode 4.34 `Cheater.java`: kode di luar `Enemy` mencoba menulis *field* privatnya.

```java
/** Merogoh field privat objek lain: kompilator menolak. */
public class Cheater {
    public static void main(String[] args) {
        Enemy villain = new Enemy("Villain", 80, 20, 5);
        villain.health = 0;
        System.out.println(villain.getHealth());
    }
}
```

Dikompilasi dengan `javac -cp .. Cheater.java` dari folder `tempting` (`-cp ..` memberi tahu kompilator di mana menemukan `Enemy`), program ini bahkan tidak dapat dikompilasi:

```
Cheater.java:5: error: health has private access in Enemy
        villain.health = 0;
               ^
1 error
```

Galat ini adalah hal yang *baik*: kompilator menegakkan aturan bahwa hanya `Enemy` yang boleh mengubah kesehatan sebuah `Enemy`. Seandainya tidak, setiap kelas dalam program dapat melangkahi aturan pertahanan dan pesan kematian. Python, seperti diperlihatkan **Kode 4.35**, hanya bersandar pada konvensi garis bawah, dan membiarkan si curang lewat tanpa sepatah kata pun.

Kode 4.35 `cheater.py`: Python tidak memaksakan sifat privat, sehingga aturannya bergantung pada disiplin pemrogram.

```python
"""Merogoh atribut 'privat' objek lain: Python tidak mencegah Anda."""
from player_enemy import Enemy

villain = Enemy("Villain", 80, 20, 5)
villain._health = 0
print(villain.get_health())
```

```
0
```

Penjahat itu kini berkesehatan 0, tetapi ia tidak pernah mencetak “Villain died!”, karena satu-satunya metode yang tahu tentang kematian telah dilangkahi. Di Python, garis bawah adalah janji antarpemrogram, dan menepatinya adalah tanggung jawab Anda.

> **UJI DIRI.** Tambahkan metode `isAlive()` pada kedua kelas, dan ubahlah `takeDamage` sehingga tokoh yang sudah mati mengabaikan serangan berikutnya (dan tidak mencetak “died!” untuk kedua kalinya). Setelah pertarungan **Tabel 4.4**, apa yang dicetak oleh satu `hero.attack(villain)` tambahan, dan berapa kesehatan sang penjahat?

Jawaban. `public boolean isAlive() { return health > 0; }`, dan `takeDamage` diawali dengan `if (!isAlive()) { return; }`. Serangan tambahan itu tidak mencetak apa-apa, dan kesehatan penjahat tetap 0. Tanpa penjaga itu kesehatannya akan menjadi -20 dan "Villain died!" akan muncul lagi.

4.10 Soal 4.8 — Uang dalam Rupiah

Tujuh soal sisanya baru. Soal-soal ini tidak memakai apa pun di luar Bab 1 sampai 3, dan berasal dari jenis program usaha kecil yang dibutuhkan setiap hari oleh Profitina Laundry, studi kasus buku ini.

4.10.1 Soalnya

SOAL 4.8. Tulislah kelas `Rupiah` untuk sejumlah uang dalam rupiah utuh. Kelas ini memiliki konstruktor yang menerima jumlahnya, getter `getAmount`, dan tiga metode yang mengembalikan objek `Rupiah` baru: `plus(other)` untuk penjumlahan, `times(quantity)` untuk harga dikalikan jumlah barang yang utuh, dan `discount(percent)` untuk harga setelah diskon sebesar persen utuh yang diberikan, dibulatkan ke bawah menjadi rupiah utuh. `toString`-nya menuliskan jumlah uang dengan cara Indonesia, dengan titik di antara setiap kelompok tiga angka yang dihitung dari kanan: `Rp1.250.000`, `Rp999`, `-Rp5.000`.

4.10.2 Memahaminya dengan tangan

Uang harus tepat. Sebuah laundry yang menjumlahkan seribu nota tidak dapat menoleransi total yang meleset sepecahan rupiah, jadi kita menyimpan jumlahnya sebagai bilangan bulat bertipe `long` (bilangan bulat 64 bit, cukup besar untuk 9.2×10^{18} rupiah) dan tidak pernah sebagai `double`.

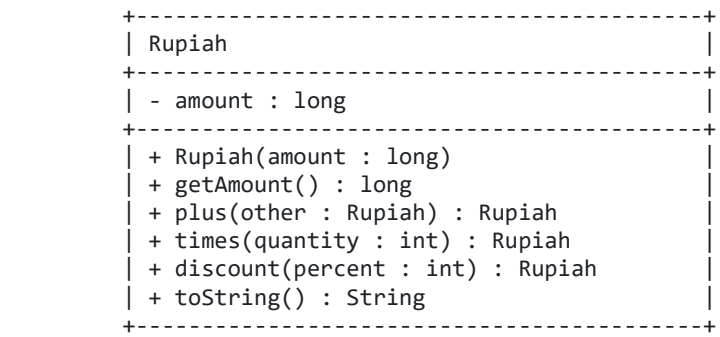
Pesanan yang akan dibutuhkan Soal 4.10: empat kilogram cucian dengan harga Rp7.000 ditambah satu barang seharga Rp12.500 adalah $4 \cdot 7000 + 12500 = 28000 + 12500 = 40500$. Diskon sepuluh persen menghasilkan $40500 \cdot \frac{90}{100} = \frac{3645000}{100} = 36450$. Diskon yang canggung: 15 persen dari Rp9.999 adalah $\frac{9999 \cdot 85}{100} = \frac{849915}{100} = 8499.15$, dibulatkan ke bawah menjadi 8499. Mengalikan *sebelum* membagi menjaga aritmetikanya tetap bulat sampai langkah terakhir.

Formatnya adalah latihan pengolahan string yang sesungguhnya. Ambillah angka-angka `1250000`. Berjalanlah dari *kanan*, salin satu angka setiap kali ke depan hasil, dan setelah setiap angka ketiga sisipkan sebuah titik — kecuali bila angka itu adalah yang paling kiri:

Tabel 4.5 Mengelompokkan angka-angka 1250000 dari kanan: sebuah titik ditambahkan setelah setiap angka ketiga, kecuali setelah angka paling kiri.

Angka yang dibaca (dari kanan)	Hitungan	Hasil sejauh ini
`0`	1	`0`
`0`	2	`00`
`0`	3	`.000`
`0`	4	`0.000`
`5`	5	`50.000`
`2`	6	`.250.000`
`1`	7	`1.250.000`

4.10.3 Kelas, field, dan metode



Seperti `Q16Fraction`, `Rupiah` bersifat *immutable*: setiap operasi mengembalikan objek baru.

4.10.4 Penyelesaian Java

Kode 4.36 adalah kelasnya; `toString`-nya adalah prosedur **Tabel 4.5** yang ditulis sebagai perulangan. **Kode 4.37** menguji batas kelompok 999 dan 1000.

Kode 4.36 `Rupiah.java`: uang yang tepat dalam rupiah utuh, diformat dengan titik dari kanan.

```java
/**
 * Soal 4.8: sejumlah uang dalam rupiah utuh. Jumlahnya bertipe long,
 * tidak pernah double, sehingga penjumlahan tepat. toString mengelompokkan angka per tiga
 * dengan titik, cara Indonesia: 1250000 menjadi "Rp1.250.000".
 */
public class Rupiah {
    private long amount;

    public Rupiah(long amount) {
        this.amount = amount;
    }

    public long getAmount() {
        return amount;
    }

    public Rupiah plus(Rupiah other) {
        return new Rupiah(amount + other.amount);
    }

    public Rupiah times(int quantity) {
        return new Rupiah(amount * quantity);
    }

    // Harga setelah diskon sekian persen, dibulatkan ke bawah menjadi rupiah utuh.
    public Rupiah discount(int percent) {
        return new Rupiah(amount * (100 - percent) / 100);
    }

    @Override
    public String toString() {
        String digits = Long.toString(Math.abs(amount));
        String grouped = "";
        int count = 0;
        for (int i = digits.length() - 1; i >= 0; i--) { // berjalan dari KANAN
            grouped = digits.charAt(i) + grouped;
            count++;
            if (count % 3 == 0 && i > 0) {
                grouped = "." + grouped;
            }
        }
        if (amount < 0) {
            return "-Rp" + grouped;
        }
        return "Rp" + grouped;
    }
```

Kode 4.36 (Lanjutan) *Rupiah.java: uang yang tepat dalam rupiah utuh, diformat dengan titik dari kanan.*

```java
}
```

Kode 4.37 `RupiahTest.java`: pemformatan di batas-batas kelompok, dan aritmetika yang tepat.

```java
/** Driver uji sederhana untuk Rupiah: jalankan dengan "java RupiahTest". */
public class RupiahTest {
    private static int passed = 0;
    private static int failed = 0;

    private static void check(String label, Object expected, Object actual) {
        if (expected.equals(actual)) {
            passed++;
            System.out.println("PASS " + label);
        } else {
            failed++;
            System.out.println("FAIL " + label + ": expected " + expected + ", got " +
actual);
        }
    }

    public static void main(String[] args) {
        Rupiah wash = new Rupiah(7000);
        Rupiah order = wash.times(4).plus(new Rupiah(12500));
        System.out.println("Order: " + order);
        System.out.println("After 10% off: " + order.discount(10));

        check("zero", "Rp0", new Rupiah(0).toString());
        check("three digits", "Rp999", new Rupiah(999).toString());
        check("four digits", "Rp1.000", new Rupiah(1000).toString());
        check("seven digits", "Rp1.250.000", new Rupiah(1250000).toString());
        check("negative", "-Rp5.000", new Rupiah(-5000).toString());
        check("sum", 40500L, order.getAmount());
        check("discount rounds down", 36450L, order.discount(10).getAmount());
        check("8499.15 rounds down", 8499L, new Rupiah(9999).discount(15).getAmount());
        System.out.println(passed + " passed, " + failed + " failed");
    }
}
```

```
Order: Rp40.500
After 10% off: Rp36.450
PASS zero
PASS three digits
PASS four digits
PASS seven digits
PASS negative
PASS sum
PASS discount rounds down
PASS 8499.15 rounds down
8 passed, 0 failed
```

4.10.5 Penyelesaian Python

Kode 4.38 mengikuti perulangan yang sama.

Kode 4.38 `rupiah.py`: Soal 4.8 dalam Python.

```
"""Soal 4.8: sejumlah uang dalam rupiah utuh. Jumlahnya bertipe int,
tidak pernah float, sehingga penjumlahan tepat. str() mengelompokkan angka per tiga
dengan titik, cara Indonesia: 1250000 menjadi "Rp1.250.000".
"""
import unittest


class Rupiah:
    def __init__(self, amount):
        self._amount = amount

    def get_amount(self):
        return self._amount

    def plus(self, other):
        return Rupiah(self._amount + other._amount)

    def times(self, quantity):
        return Rupiah(self._amount * quantity)

    def discount(self, percent):
        # Harga setelah diskon sekian persen, dibulatkan ke bawah menjadi rupiah utuh.
        return Rupiah(self._amount * (100 - percent) // 100)

    def __str__(self):
        digits = str(abs(self._amount))
        grouped = ""
        count = 0
        for i in range(len(digits) - 1, -1, -1):  # berjalan dari KANAN
            grouped = digits[i] + grouped
            count += 1
            if count % 3 == 0 and i > 0:
                grouped = "." + grouped
        if self._amount < 0:
            return "-Rp" + grouped
        return "Rp" + grouped


class TestRupiah(unittest.TestCase):
    def test_format(self):
        self.assertEqual(str(Rupiah(0)), "Rp0")
        self.assertEqual(str(Rupiah(999)), "Rp999")
        self.assertEqual(str(Rupiah(1000)), "Rp1.000")
        self.assertEqual(str(Rupiah(1250000)), "Rp1.250.000")
        self.assertEqual(str(Rupiah(-5000)), "-Rp5.000")
```

Kode 4.38 (Lanjutan) *rupiah.py: Soal 4.8 dalam Python.*

```
    def test_arithmetic(self):
        order = Rupiah(7000).times(4).plus(Rupiah(12500))
        self.assertEqual(order.get_amount(), 40500)
        self.assertEqual(order.discount(10).get_amount(), 36450)
        self.assertEqual(Rupiah(9999).discount(15).get_amount(), 8499)


if __name__ == "__main__":
    order = Rupiah(7000).times(4).plus(Rupiah(12500))
    print(f"Order: {order}")
    print(f"After 10% off: {order.discount(10)}")
    unittest.main(verbosity=2)
```

```
Order: Rp40.500
After 10% off: Rp36.450
test_arithmetic (__main__.TestRupiah.test_arithmetic) ... ok
test_format (__main__.TestRupiah.test_format) ... ok

----------------------------------------------------------------------
Ran 2 tests in 0.000s

OK
```

4.10.6 Baris demi baris

- `String digits = Long.toString(Math.abs(amount));` — ubah nilai mutlaknya menjadi deretan angka; tanda negatif ditangani terpisah di bagian akhir.
- `for (int i = digits.length() - 1; i >= 0; i--)` — perulangan `for` yang menghitung *mundur*, dari indeks karakter terakhir sampai 0. `digits.charAt(i)` adalah karakter pada posisi `i` (karakter pertama berada di posisi 0).
- `grouped = digits.charAt(i) + grouped;` — menambahkan `char` ke `String` meletakkan karakter itu di depan. Membangun string dengan `+` berulang kali membuat string baru setiap kali; untuk selusin angka biayanya tidak berarti, dan untuk teks yang panjang kelas baku `StringBuilder` mengerjakan tugas yang sama tanpa penyalinan itu.
- `if (count % 3 == 0 && i > 0)` — `count % 3 == 0` berarti "hitungan adalah kelipatan 3"; `i > 0` berarti "masih ada angka di sebelah kiri". Keduanya harus terpenuhi, dan itulah yang dinyatakan `&&`. Tanpa `i > 0`, `999` akan menjadi `.999`.
- `amount * (100 - percent) / 100` — Java mengevaluasi dari kiri ke kanan: pertama perkalian, lalu pembagian, yang membuang pecahannya. Python menulis hal yang sama sebagai `// 100`. Untuk jumlah negatif kedua bahasa berbeda (`/` di Java memotong ke arah nol, `//` di Python membulatkan ke bawah), tetapi diskon selalu dikenakan pada harga yang positif.
- `range(len(digits) - 1, -1, -1)` di Python menghasilkan indeks dari yang terakhir turun sampai 0: nilai henti `-1` tidak ikut, dan langkah `-1` menghitung mundur.

4.10.7 Penyelesaian keliru yang menggoda

Kode 4.39 menelusuri angka-angka dengan arah baca yang wajar, dari kiri.

Kode 4.39 `WrongRupiahFormat.java`: mengelompokkan dari kiri, dan uang dalam `double`.

```java
/** Mengelompokkan angka dari kiri, dan menyimpan uang dalam double. */
public class WrongRupiahFormat {
    public static void main(String[] args) {
        String digits = "12500";
        String grouped = "";
        for (int i = 0; i < digits.length(); i++) {
            if (i > 0 && i % 3 == 0) {
                grouped = grouped + ".";
            }
            grouped = grouped + digits.charAt(i);
        }
        System.out.println("from the left: Rp" + grouped);
        System.out.println("double money:  " + 7000 * 1.1);
    }
}
```

```
from the left: Rp125.00
double money:  7700.000000000001
```

Menghitung dari kiri meletakkan titik sesudah `125`, sehingga dua belas ribu lima ratus berubah menjadi sesuatu yang tampak seperti seratus dua puluh lima. Kelompok tiga angka adalah milik *ribuan*, *jutaan*, dan seterusnya, yang dihitung dari kanan. Baris kedua memperlihatkan mengapa uang tidak pernah disimpan dalam `double`: menaikkan Rp7.000 sebesar 10 persen dengan mengalikannya dengan `1.1` menghasilkan `7700.000000000001`, karena 1.1 tidak memiliki bentuk biner yang eksak. **Kode 4.40** memperlihatkan dua hasil yang sama dalam Python.

Kode 4.40 `wrong_rupiah_format.py`: dua kesalahan yang sama dalam Python.

```
"""Mengelompokkan angka dari kiri, dan menyimpan uang dalam float."""
digits = "12500"
grouped = ""
for i in range(len(digits)):
    if i > 0 and i % 3 == 0:
        grouped = grouped + "."
    grouped = grouped + digits[i]
print(f"from the left: Rp{grouped}")
print(f"float money:   {7000 * 1.1}")
```

```
from the left: Rp125.00
float money:   7700.000000000001
```

> **UJI DIRI.** Tambahkan metode `split(int people)` yang mengembalikan bagian setiap orang ketika tagihan dibagi rata, dibulatkan *ke bawah*, dan metode `remainder(int people)` yang mengembalikan sisanya. Untuk Rp100.000 dan 3 orang, apa yang dikembalikan keduanya, dan mengapa sisa itu diperlukan?
>
> *Jawaban.* `split` mengembalikan `new Rupiah(amount / people)` dan `remainder` mengembalikan `new Rupiah(amount % people)`: masing-masing Rp33.333, dan tersisa Rp1. Tanpa sisa itu, $3 \cdot 33333 = 99999$ dan satu rupiah akan lenyap diam-diam; dengan sisa itu, bagian-bagian ditambah sisanya tepat sama dengan tagihannya.

4.11 Soal 4.9 — Waktu dalam Sehari

4.11.1 Soalnya

> **SOAL 4.9.** Profitina Laundry menjanjikan bahwa sebuah layanan selesai sekian menit setelah pelanggan menyerahkan cuciannya. Tulislah kelas `TimeOfDay` untuk waktu pada jam 24 jam dengan konstruktor `TimeOfDay(hours, minutes)`, getter `getHours` dan `getMinutes`, metode `plusMinutes(m)` yang mengembalikan waktu `m` menit kemudian (atau sebelumnya, untuk `m` negatif) dengan berputar melewati tengah malam, metode `isBefore(other)`, metode `minutesUntil(other)` yang mengembalikan berapa menit terbentang dari waktu ini maju ke waktu lain, dan `toString` yang selalu mencetak dua angka untuk jam dan dua angka untuk menit, seperti `08:05`.

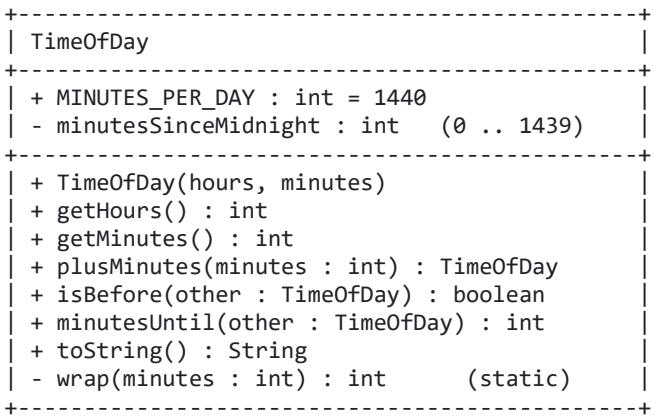
4.11.2 Memahaminya dengan tangan

Tiga contoh. Diserahkan pukul 22:30, selesai 45 menit kemudian: $30 + 45 = 75$ menit, yaitu 1 jam 15 menit, jadi $22 + 1 = 23$ dan waktunya 23:15 — menitnya *menyimpan* ke jam. Selesai 150 menit setelah 22:30: 150 menit adalah 2 jam 30 menit, jadi $22{:}30 + 2{:}30 = 24{:}60$, yaitu 25:00, yang pada jam 24 jam berarti 01:00 keesokan paginya — jamnya *berputar balik* melewati tengah malam. Dan 20 menit sebelum 00:10 adalah 23:50 malam sebelumnya.

Penyimpanan dan putaran balik itu menjadi sepele dengan satu gagasan: simpan bukan dua bilangan melainkan satu, yaitu *menit sejak tengah malam*, dari 0 (00:00) sampai 1439 (23:59).

Maka 22:30 adalah $22 \cdot 60 + 30 = 1350$; ditambah 150 menjadi 1500; sehari memiliki $24 \cdot 60 = 1440$ menit, dan $1500 \bmod 1440 = 60$, yaitu $60/60 = 1$ jam dan $60 \bmod 60 = 0$ menit: 01:00. Mundur 20 menit dari 00:10 memberi $10 - 20 = -10$, dan -10 harus berputar menjadi 1430, yaitu 23:50 — inilah sisa negatif dari Soal 4.2 sekali lagi.

4.11.3 Kelas, field, dan metode



Dunia luar tetap berbicara dalam jam dan menit — konstruktor menerima keduanya, dan getter mengembalikan keduanya — sementara bagian dalamnya memakai representasi yang membuat aritmetikanya mudah. Memilih *representasi internal* yang baik, lalu menyembunyikannya, itulah yang diberikan enkapsulasi.

4.11.4 Penyelesaian Java

Kode 4.41 adalah kelasnya dan **Kode 4.42** driver-nya, yang baris pertamanya adalah janji yang tercetak pada tiket laundry.

Kode 4.41 `TimeOfDay.java`: waktu dalam sehari yang disimpan sebagai menit sejak tengah malam.

```java
/**
 * Soal 4.9: waktu dalam sehari pada jam 24 jam, disimpan sebagai banyaknya
 * menit sejak tengah malam (0 sampai 1439). Penambahan menit berputar melewati tengah
malam.
 */
public class TimeOfDay {
    public static final int MINUTES_PER_DAY = 24 * 60;

    private int minutesSinceMidnight;

    public TimeOfDay(int hours, int minutes) {
        minutesSinceMidnight = wrap(hours * 60 + minutes);
    }

    private static int wrap(int minutes) {
        return (minutes % MINUTES_PER_DAY + MINUTES_PER_DAY) % MINUTES_PER_DAY;
    }

    public int getHours() {
        return minutesSinceMidnight / 60;
    }

    public int getMinutes() {
        return minutesSinceMidnight % 60;
    }

    public TimeOfDay plusMinutes(int minutes) {
        return new TimeOfDay(0, minutesSinceMidnight + minutes);
    }

    public boolean isBefore(TimeOfDay other) {
        return minutesSinceMidnight < other.minutesSinceMidnight;
    }

    // Menit dari waktu ini maju ke waktu lain, mungkin melintasi tengah malam.
    public int minutesUntil(TimeOfDay other) {
        return wrap(other.minutesSinceMidnight - minutesSinceMidnight);
    }

    @Override
    public String toString() {
        return String.format("%02d:%02d", getHours(), getMinutes());
    }
}
```

Kode 4.42 `TimeOfDayTest.java`: penyimpanan ke jam, putaran balik ke dua arah, pengisian nol, dan perbandingan.

```java
/** Driver uji sederhana untuk TimeOfDay: jalankan dengan "java TimeOfDayTest". */
public class TimeOfDayTest {
    private static int passed = 0;
    private static int failed = 0;

    private static void check(String label, Object expected, Object actual) {
        if (expected.equals(actual)) {
            passed++;
            System.out.println("PASS " + label);
        } else {
            failed++;
            System.out.println("FAIL " + label + ": expected " + expected + ", got " +
actual);
        }
    }

    public static void main(String[] args) {
        TimeOfDay dropOff = new TimeOfDay(22, 30);
        TimeOfDay ready = dropOff.plusMinutes(150);
        System.out.println("Dropped off at " + dropOff + ", ready at " + ready);

        check("zero padding", "08:05", new TimeOfDay(8, 5).toString());
        check("carry into the hour", "23:15", dropOff.plusMinutes(45).toString());
        check("wrap past midnight", "01:00", ready.toString());
        check("going back", "23:50", new TimeOfDay(0, 10).plusMinutes(-20).toString());
        check("one whole day", "22:30", dropOff.plusMinutes(1440).toString());
        check("isBefore", true, new TimeOfDay(9, 0).isBefore(new TimeOfDay(9, 1)));
        check("minutesUntil across midnight", 150, dropOff.minutesUntil(ready));
        check("getHours", 1, ready.getHours());
        System.out.println(passed + " passed, " + failed + " failed");
    }
}
```

```
Dropped off at 22:30, ready at 01:00
PASS zero padding
PASS carry into the hour
PASS wrap past midnight
PASS going back
PASS one whole day
PASS isBefore
PASS minutesUntil across midnight
PASS getHours
8 passed, 0 failed
```

4.11.5 Penyelesaian Python

Kode 4.43 tidak memerlukan pembantu untuk putaran balik, karena `%` di Python tidak pernah menghasilkan bilangan negatif di sini.

Kode 4.43 `time_of_day.py`: Soal 4.9 dalam Python.

```
"""Soal 4.9: waktu dalam sehari pada jam 24 jam, disimpan sebagai banyaknya
menit sejak tengah malam (0 sampai 1439). Penambahan menit berputar melewati tengah malam.
"""
import unittest

MINUTES_PER_DAY = 24 * 60


class TimeOfDay:
    def __init__(self, hours, minutes):
        # % di Python dengan pembagi positif tidak pernah menghasilkan bilangan negatif.
        self._minutes_since_midnight = (hours * 60 + minutes) % MINUTES_PER_DAY

    def get_hours(self):
        return self._minutes_since_midnight // 60

    def get_minutes(self):
        return self._minutes_since_midnight % 60

    def plus_minutes(self, minutes):
        return TimeOfDay(0, self._minutes_since_midnight + minutes)

    def is_before(self, other):
        return self._minutes_since_midnight < other._minutes_since_midnight

    def minutes_until(self, other):
        # Menit dari waktu ini maju ke waktu lain, mungkin melintasi tengah malam.
        return (other._minutes_since_midnight - self._minutes_since_midnight) %
MINUTES_PER_DAY

    def __str__(self):
        return f"{self.get_hours():02d}:{self.get_minutes():02d}"


class TestTimeOfDay(unittest.TestCase):
    def test_format_and_carry(self):
        drop_off = TimeOfDay(22, 30)
        self.assertEqual(str(TimeOfDay(8, 5)), "08:05")
        self.assertEqual(str(drop_off.plus_minutes(45)), "23:15")
        self.assertEqual(str(drop_off.plus_minutes(150)), "01:00")
        self.assertEqual(str(drop_off.plus_minutes(1440)), "22:30")

    def test_going_back(self):
        self.assertEqual(str(TimeOfDay(0, 10).plus_minutes(-20)), "23:50")
```

Kode 4.43 (Lanjutan) *time_of_day.py: Soal 4.9 dalam Python.*

```
    def test_comparisons(self):
        drop_off = TimeOfDay(22, 30)
        ready = drop_off.plus_minutes(150)
        self.assertTrue(TimeOfDay(9, 0).is_before(TimeOfDay(9, 1)))
        self.assertEqual(drop_off.minutes_until(ready), 150)
        self.assertEqual(ready.get_hours(), 1)


if __name__ == "__main__":
    drop_off = TimeOfDay(22, 30)
    ready = drop_off.plus_minutes(150)
    print(f"Dropped off at {drop_off}, ready at {ready}")
    unittest.main(verbosity=2)
```

```
Dropped off at 22:30, ready at 01:00
test_comparisons (__main__.TestTimeOfDay.test_comparisons) ... ok
test_format_and_carry (__main__.TestTimeOfDay.test_format_and_carry) ... ok
test_going_back (__main__.TestTimeOfDay.test_going_back) ... ok

----------------------------------------------------------------------
Ran 3 tests in 0.000s

OK
```

4.11.6 Baris demi baris

- `minutesSinceMidnight = wrap(hours * 60 + minutes);` — konstruktor menerima bilangan apa pun dan menormalisasinya, sehingga `new TimeOfDay(0, 1500)` adalah cara yang sah untuk menulis 01:00. `plusMinutes` memakai persis hal ini: `new TimeOfDay(0, minutesSinceMidnight + minutes)`.
- `private static int wrap(int minutes)` — pembantu yang *privat*: kelas membutuhkannya, tetapi ia bukan bagian dari apa yang ditawarkan kelas. Ekspresi `(minutes % 1440 + 1440) % 1440` adalah idiom Java dari Soal 4.2 untuk sisa pembagian yang tidak pernah negatif. Di Python cukup satu `% MINUTES_PER_DAY`.
- `getHours()` adalah `minutesSinceMidnight / 60` (pembagian bilangan bulat, yang di sini memang *diinginkan*) dan `getMinutes()` adalah `minutesSinceMidnight % 60`.
- `String.format("%02d:%02d", getHours(), getMinutes())` — `%d` mencetak bilangan bulat, `2` meminta sekurang-kurangnya dua karakter, dan `0` mengisi dengan nol alih-alih spasi. F-string Python `f"{h:02d}"` memakai bahasa kecil yang sama.
- `minutesUntil` memutar balik selisihnya, sehingga dari 22:30 ke 01:00 jawabannya 150, bukan $60 - 1350 = -1290$.

4.11.7 Penyelesaian keliru yang menggoda

Kode 4.44 `WrongTime.java`: menambah menit tanpa penyimpanan ke jam, tanpa putaran balik, dan tanpa pengisian nol.

```java
/** Menambah menit tanpa simpanan, tanpa putaran balik, dan tanpa pengisian nol. */
public class WrongTime {
    public static void main(String[] args) {
        int hours = 22, minutes = 30;
        System.out.println("no carry:   " + hours + ":" + (minutes + 45));
        int total = hours * 60 + minutes + 150;
        System.out.println("no wrap:    " + total / 60 + ":" + total % 60);
        System.out.println("no padding: " + 8 + ":" + 5);
    }
}
```

```
no carry:   22:75
no wrap:    25:0
no padding: 8:5
```

Setiap baris adalah bug nyata yang pernah ditemukan dalam sistem pemesanan sungguhan: `22:75` (tanpa penyimpanan ke jam), `25:0` (tanpa putaran balik dan tanpa pengisian nol), dan `8:5` (tanpa pengisian nol). Ketiganya lenyap begitu waktu menjadi satu bilangan menit yang selalu dijaga berada di dalam rentangnya. **Kode 4.45** memperlihatkan tiga hasil yang sama dalam Python.

Kode 4.45 `wrong_time.py`: tiga kekeliruan yang sama dalam Python.

```
"""Menambah menit tanpa simpanan, tanpa putaran balik, dan tanpa pengisian nol."""
hours, minutes = 22, 30
print(f"no carry:   {hours}:{minutes + 45}")
total = hours * 60 + minutes + 150
print(f"no wrap:    {total // 60}:{total % 60}")
print(f"no padding: {8}:{5}")
```

```
no carry:   22:75
no wrap:    25:0
no padding: 8:5
```

UJI DIRI. Profitina Laundry buka dari pukul 07:00 sampai 21:00. Tulislah metode `isOpenAt(TimeOfDay t)` (statis, atau di dalam kelas kecil pilihan Anda). Manakah di antara 06:59, 07:00, 20:59, dan 21:00 yang termasuk "buka"?

Jawaban. Buka berarti `!t.isBefore(new TimeOfDay(7, 0)) && t.isBefore(new TimeOfDay(21, 0))`: tidak sebelum jam buka, dan sebelum jam tutup. Jadi 06:59 tutup, 07:00 buka, 20:59 buka, dan 21:00 tutup. Menuliskan kedua batas dengan cara ini — menit pembukaan disertakan, menit penutupan tidak — adalah kebiasaan umum untuk rentang waktu.

4.12 Soal 4.10 — Nota Laundry

4.12.1 Soalnya

SOAL 4.10. Tulislah kelas `LaundryReceipt` untuk Profitina Laundry. Sebuah nota dibuat untuk nama seorang pelanggan dan menyatakan apakah pesanannya *ekspres*. Metode `addService(service, kg)` menambahkan satu baris: layanannya `"wash"` (cuci, Rp7.000 per kilogram), `"iron"` (setrika, Rp5.000 per kilogram), atau `"wash+iron"` (cuci dan setrika, Rp10.000 per kilogram); berat ditagih per kilogram *yang dimulai*, sehingga 3.2 kg ditagih sebagai 4 kg; pesanan ekspres 50 persen lebih mahal. Layanan yang tidak dikenal atau berat yang tidak positif ditolak: metodenya mengembalikan `false` dan tidak menambahkan apa pun. Nota menyimpan total berjalan sebagai `Rupiah` (Soal 4.8), dan `toString`-nya mencetak baris-baris dan totalnya dalam kolom-kolom yang rata.

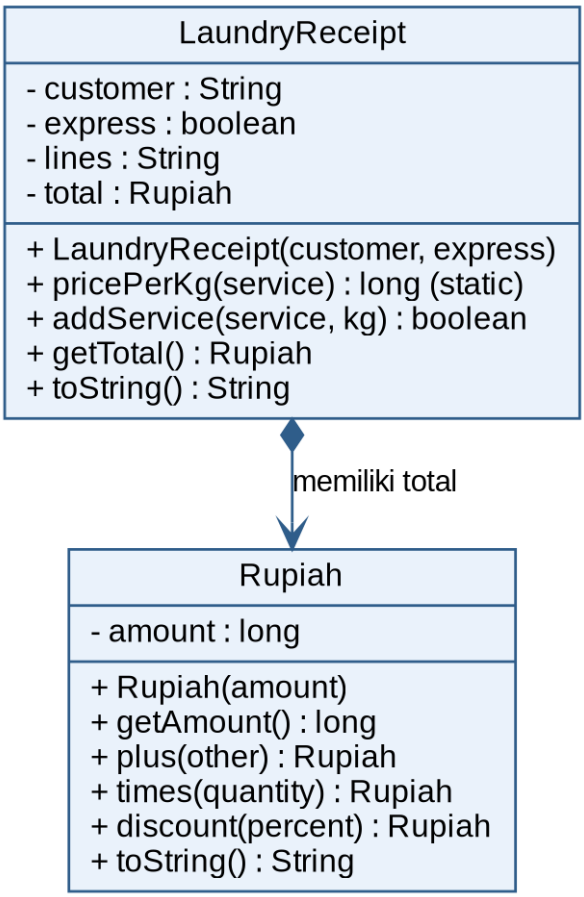
4.12.2 Memahaminya dengan tangan

Sari membawa 3.2 kg untuk dicuci dan 2.0 kg untuk disetrika, bukan ekspres. Cuci: 3.2 kg ditagih sebagai 4 kg, dan $4 \cdot 7000 = 28000$. Setrika: 2.0 kg tepat 2 kilogram yang dimulai, dan $2 \cdot 5000 = 10000$. Total 38000. Budi membawa 4.5 kg untuk dicuci dan disetrika, ekspres: 5 kg dengan harga Rp10.000 adalah 50000, dan setengahnya lagi 25000, jadi 75000.

"Per kilogram yang dimulai" adalah *fungsi atap* (*ceiling*) dalam matematika: bilangan bulat terkecil yang tidak kurang dari beratnya, ditulis $\lceil 3.2 \rceil = 4$ dan $\lceil 2.0 \rceil = 2$. `Math.ceil` di Java dan `math.ceil` di Python menghitungnya.

4.12.3 Kelas, field, dan metode

Gambar 4.5 memperlihatkan rancangannya: sebuah nota *memiliki* total bertipe `Rupiah`. Memakai objek dari satu kelas sebagai *field* kelas lain disebut *komposisi* (*composition*), dan cara ini memungkinkan `LaundryReceipt` memakai ulang, alih-alih mengulang, semua yang diketahui Soal 4.8 tentang uang.



Gambar 4.5 Diagram kelas Soal 4.10: sebuah `LaundryReceipt` memiliki total bertipe `Rupiah`; belah ketupat terisi menandai bahwa total itu milik nota.

4.12.4 Penyelesaian Java

Kode 4.46 adalah notanya dan **Kode 4.47** driver-nya.

Kode 4.46 `LaundryReceipt.java`: nota yang menetapkan harga layanan dengan `switch` dan menjumlahkannya dalam `Rupiah`.

```
import java.util.Locale;

/**
 * Soal 4.10: nota Profitina Laundry. Setiap baris layanan ditagih
 * per kilogram yang dimulai (3.2 kg ditagih sebagai 4 kg); layanan ekspres
 * 50 persen lebih mahal. Nota menyimpan barisnya sebagai teks dan totalnya sebagai Rupiah.
 */
public class LaundryReceipt {
    private String customer;
    private boolean express;
    private String lines;
    private Rupiah total;

    public LaundryReceipt(String customer, boolean express) {
        this.customer = customer;
        this.express = express;
        this.lines = "";
        this.total = new Rupiah(0);
    }

    // Harga per kilogram suatu layanan, atau 0 untuk layanan yang tidak kami sediakan.
    public static long pricePerKg(String service) {
        switch (service) {
            case "wash":
                return 7000;
            case "iron":
                return 5000;
            case "wash+iron":
                return 10000;
            default:
                return 0;
        }
    }

    // Menambah satu baris layanan; mengembalikan false (tanpa menambah apa pun) untuk
layanan tak dikenal.
    public boolean addService(String service, double kg) {
        long perKg = pricePerKg(service);
        if (perKg == 0 || kg <= 0) {
            return false;
        }
        int chargedKg = (int) Math.ceil(kg);
        Rupiah price = new Rupiah(perKg).times(chargedKg);
        if (express) {
            price = price.plus(new Rupiah(price.getAmount() / 2));
        }
        total = total.plus(price);
```

Kode 4.46 (Lanjutan) *LaundryReceipt.java: nota yang menetapkan harga layanan dengan switch dan menjumlahkannya dalam Rupiah.*

```
        // Locale.ROOT mencetak 3.2, bukan 3,2, apa pun bahasa komputernya.
        lines = lines + String.format(Locale.ROOT, "%-10s %4.1f kg -> %d kg  %12s%n",
                service, kg, chargedKg, price);
        return true;
    }

    public Rupiah getTotal() {
        return total;
    }

    @Override
    public String toString() {
        String header = "Profitina Laundry - " + customer + (express ? " (express)" : "");
        return String.format(Locale.ROOT, "%s%n%s%-28s%12s", header, lines, "TOTAL",
total);
    }
}
```

Kode 4.47 `LaundryReceiptTest.java`: nota Sari dan Budi serta penolakan-penolakannya.

```
/** Driver uji sederhana untuk LaundryReceipt: jalankan dengan "java LaundryReceiptTest".
*/
public class LaundryReceiptTest {
    private static int passed = 0;
    private static int failed = 0;

    private static void check(String label, Object expected, Object actual) {
        if (expected.equals(actual)) {
            passed++;
            System.out.println("PASS " + label);
        } else {
            failed++;
            System.out.println("FAIL " + label + ": expected " + expected + ", got " +
actual);
        }
    }

    public static void main(String[] args) {
        LaundryReceipt sari = new LaundryReceipt("Sari", false);
        sari.addService("wash", 3.2);
        sari.addService("iron", 2.0);
        System.out.println(sari);

        LaundryReceipt budi = new LaundryReceipt("Budi", true);
        budi.addService("wash+iron", 4.5);
        System.out.println(budi);

        check("3.2 kg charged as 4 kg", 38000L, sari.getTotal().getAmount());
        check("express adds half", 75000L, budi.getTotal().getAmount());
        check("unknown service refused", false, sari.addService("dry clean", 1.0));
        check("refusal leaves total", 38000L, sari.getTotal().getAmount());
        LaundryReceipt exact = new LaundryReceipt("Test", false);
        exact.addService("iron", 2.0);
        check("exactly 2.0 kg stays 2 kg", 10000L, exact.getTotal().getAmount());
        check("zero weight refused", false, sari.addService("wash", 0));
        System.out.println(passed + " passed, " + failed + " failed");
    }
}
```

Kompilasi dengan `javac Rupiah.java LaundryReceipt.java LaundryReceiptTest.java` (atau cukup `javac *.java`) lalu jalankan:

```
Profitina Laundry - Sari
wash        3.2 kg -> 4 kg      Rp28.000
iron        2.0 kg -> 2 kg      Rp10.000
TOTAL                           Rp38.000
Profitina Laundry - Budi (express)
wash+iron   4.5 kg -> 5 kg      Rp75.000
TOTAL                           Rp75.000
PASS 3.2 kg charged as 4 kg
PASS express adds half
PASS unknown service refused
PASS refusal leaves total
PASS exactly 2.0 kg stays 2 kg
PASS zero weight refused
6 passed, 0 failed
```

4.12.5 Penyelesaian Python

Kode 4.48 mencetak nota-nota yang persis sama.

Kode 4.48 `laundry_receipt.py`: Soal 4.10 dalam Python, yang mengimpor `Rupiah` dari `rupiah.py`.

```
"""Soal 4.10: nota Profitina Laundry. Setiap baris layanan ditagih
per kilogram yang dimulai (3.2 kg ditagih sebagai 4 kg); layanan ekspres
50 persen lebih mahal. Nota menyimpan barisnya sebagai teks dan totalnya sebagai Rupiah.
"""
import math
import unittest

from rupiah import Rupiah


def price_per_kg(service):
    # Harga per kilogram suatu layanan, atau 0 untuk layanan yang tidak kami sediakan.
    match service:
        case "wash":
            return 7000
        case "iron":
            return 5000
        case "wash+iron":
            return 10000
        case _:
            return 0


class LaundryReceipt:
    def __init__(self, customer, express):
        self._customer = customer
        self._express = express
        self._lines = ""
        self._total = Rupiah(0)

    def add_service(self, service, kg):
        # Menambah satu baris layanan; mengembalikan False (tanpa menambah apa pun) untuk
layanan tak dikenal.
        per_kg = price_per_kg(service)
        if per_kg == 0 or kg <= 0:
            return False
        charged_kg = math.ceil(kg)
        price = Rupiah(per_kg).times(charged_kg)
        if self._express:
            price = price.plus(Rupiah(price.get_amount() // 2))
        self._total = self._total.plus(price)
        self._lines += f"{service:<10} {kg:4.1f} kg -> {charged_kg} kg  {str(price):>12}\n"
        return True

    def get_total(self):
        return self._total
```

Kode 4.48 (Lanjutan) *laundry_receipt.py: Soal 4.10 dalam Python, yang mengimpor Rupiah dari rupiah.py.*

```
    def __str__(self):
        header = f"Profitina Laundry - {self._customer}" + (" (express)" if self._express
else "")
        return f"{header}\n{self._lines}{'TOTAL':<28}{str(self._total):>12}"


class TestLaundryReceipt(unittest.TestCase):
    def test_started_kilograms(self):
        sari = LaundryReceipt("Sari", False)
        sari.add_service("wash", 3.2)
        sari.add_service("iron", 2.0)
        self.assertEqual(sari.get_total().get_amount(), 38000)

    def test_express(self):
        budi = LaundryReceipt("Budi", True)
        budi.add_service("wash+iron", 4.5)
        self.assertEqual(budi.get_total().get_amount(), 75000)

    def test_refusals(self):
        sari = LaundryReceipt("Sari", False)
        self.assertFalse(sari.add_service("dry clean", 1.0))
        self.assertFalse(sari.add_service("wash", 0))
        self.assertEqual(sari.get_total().get_amount(), 0)


if __name__ == "__main__":
    sari = LaundryReceipt("Sari", False)
    sari.add_service("wash", 3.2)
    sari.add_service("iron", 2.0)
    print(sari)
    budi = LaundryReceipt("Budi", True)
    budi.add_service("wash+iron", 4.5)
    print(budi)
    unittest.main(verbosity=2)
```

```
Profitina Laundry - Sari
wash        3.2 kg -> 4 kg      Rp28.000
iron        2.0 kg -> 2 kg      Rp10.000
TOTAL                           Rp38.000
Profitina Laundry - Budi (express)
wash+iron   4.5 kg -> 5 kg      Rp75.000
TOTAL                           Rp75.000
test_express (__main__.TestLaundryReceipt.test_express) ... ok
test_refusals (__main__.TestLaundryReceipt.test_refusals) ... ok
test_started_kilograms (__main__.TestLaundryReceipt.test_started_kilograms) ... ok

----------------------------------------------------------------------
Ran 3 tests in 0.000s

OK
```

4.12.6 Baris demi baris

- `switch (service) { case "wash": return 7000; … default: return 0; }` — `switch` atas string membandingkan dengan `equals`, bukan `==`, sehingga aman untuk teks yang diketik. Setiap `case` langsung mengembalikan nilai, jadi `break` tidak diperlukan. Python 3.10 dan sesudahnya memiliki pernyataan `match` yang serupa, dengan `case _:` sebagai cabang bawaan.
- `if (perKg == 0 || kg <= 0) { return false; }` — tolak sejak awal, sebelum ada yang diubah. `||` berarti “atau”.
- `int chargedKg = (int) Math.ceil(kg);` — `Math.ceil(3.2)` adalah `double` `4.0`, dan *cast* menjadikannya `int` 4.

- `price = price.plus(new Rupiah(price.getAmount() / 2));` — biaya tambahan ekspres, dalam rupiah utuh.
- `String.format(Locale.ROOT, "%-10s %4.1f kg -> %d kg  %12s%n", …)` — `%-10s` adalah string rata kiri dalam 10 karakter, `%4.1f` bilangan dengan satu desimal dalam 4 karakter, `%12s` string rata kanan dalam 12 karakter, dan `%n` pergantian baris. `Locale.ROOT` penting di Indonesia: tanpanya, komputer yang disetel ke bahasa Indonesia mencetak `3,2` dengan *koma* desimal [6]; lihat di bawah.
- Di Python, `f"{service:<10} {kg:4.1f} kg"` memakai `<` untuk rata kiri dan `>` untuk rata kanan; f-string Python mengabaikan pengaturan bahasa komputer, sehingga selalu mencetak `3.2`. `from rupiah import Rupiah` berhasil karena kedua berkas berada di folder yang sama.

4.12.7 Penyelesaian keliru yang menggoda

Kode 4.49 mencoba ketiga cara mengubah 3.2 menjadi bilangan bulat, dan mencetak dengan *locale* Indonesia.

Kode 4.49 `WrongReceipt.java`: memotong atau membulatkan berat, dan memformat dengan *locale* Indonesia.

```
import java.util.Locale;

/** Menagih dengan kilogram yang dipotong, dan memformat dengan locale komputer sendiri. */
public class WrongReceipt {
    public static void main(String[] args) {
        double kg = 3.2;
        System.out.println("(int) kg:        " + (int) kg + " kg -> Rp" + (int) kg * 7000);
        System.out.println("Math.round(kg):  " + Math.round(kg) + " kg");
        System.out.println("Math.ceil(kg):   " + (int) Math.ceil(kg) + " kg");
        Locale indonesia = Locale.forLanguageTag("id-ID");
        System.out.println("Indonesian locale: " + String.format(indonesia, "%4.1f kg",
kg));
    }
}
```

```
(int) kg:        3 kg -> Rp21000
Math.round(kg):  3 kg
Math.ceil(kg):   4 kg
Indonesian locale:  3,2 kg
```

Cast `(int) 3.2` sekadar memotong pecahannya dan menagih 3 kg — Rp21.000 alih-alih Rp28.000, kerugian Rp7.000 untuk satu kantong. `Math.round` juga memberi 3; hanya `Math.ceil` yang memberi 4 seperti dituntut aturannya. Baris terakhir memperlihatkan jebakan *locale*: format yang sama mencetak `3,2` pada komputer yang disetel ke bahasa Indonesia. **Kode 4.50** menambahkan sebuah kejutan dari Python.

Kode 4.50 `wrong_receipt.py`: pemotongan, pembulatan, dan pembulatan setengah ke genap milik Python.

```
"""Menagih dengan kilogram yang dipotong atau dibulatkan, bukan kilogram yang dimulai."""
import math

kg = 3.2
print(f"int(kg):       {int(kg)} kg -> Rp{int(kg) * 7000}")
print(f"round(kg):     {round(kg)} kg")
print(f"round(2.5):    {round(2.5)} kg")
print(f"math.ceil(kg): {math.ceil(kg)} kg")
```

```
int(kg):       3 kg -> Rp21000
round(kg):     3 kg
round(2.5):    2 kg
math.ceil(kg): 4 kg
```

`round(2.5)` di Python bernilai 2 — aturan setengah ke genap yang kita temui di Soal 4.2 — sehingga laundry yang "membulatkan" 2.5 kg akan menagih 2 kg. Pelajarannya: bila aturan usaha berbunyi "per kilogram yang dimulai", pakailah fungsi yang berarti persis itu, yaitu fungsi atap.

> **UJI DIRI.** Tambahkan tagihan minimum: setiap pesanan yang totalnya di bawah Rp20.000 ditagih Rp20.000. Di mana aturan itu seharusnya diletakkan, dan berapa totalnya untuk satu baris setrika 1.5 kg yang bukan ekspres?
>
> *Jawaban.* Aturan itu menyangkut seluruh nota, bukan satu baris, jadi tempatnya di `getTotal()`: `if (total.getAmount() < 20000) { return new Rupiah(20000); } return total;`. Untuk setrika 1.5 kg, 2 kilogram yang dimulai berharga $2 \cdot 5000 = 10000$, di bawah minimum, jadi totalnya Rp20.000. (`toString` kemudian harus mencetak `getTotal()`, bukan *field* `total`.)

4.13 Soal 4.11 — Merapikan Nama

4.13.1 Soalnya

> **SOAL 4.11.** Di meja kasir Profitina Laundry, nama diketik dengan tergesa-gesa: `"  siti NUR aisyah "`. Tulislah kelas `PersonName` yang menyimpan teks mentahnya dan menyediakan tiga metode: `wordCount()`, banyaknya kata; `initials()`, huruf pertama setiap kata dalam huruf kapital, masing-masing diikuti titik (`"S.N.A."`); dan `tidy()`, nama dengan satu spasi di antara kata-kata, tanpa spasi di kedua ujungnya, dan setiap kata diawali huruf kapital (`"Siti Nur Aisyah"`). Pakailah hanya `length`, `charAt`, dan metode-metode karakter dari Bab 3.

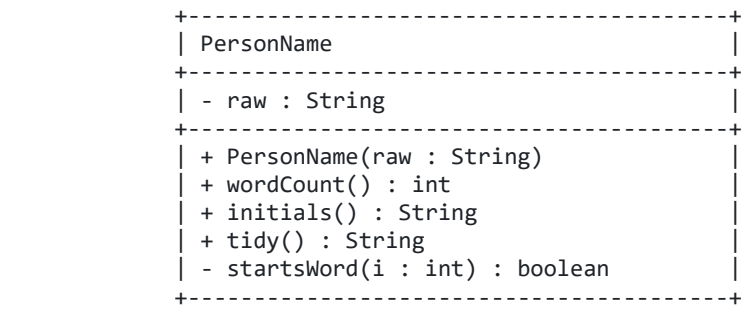
4.13.2 Memahaminya dengan tangan

Seluruh soal ini bertumpu pada satu definisi: di mana sebuah kata *dimulai*? Sebuah karakter memulai kata bila ia bukan spasi, dan ia adalah karakter paling pertama atau karakter sebelumnya adalah spasi. Mari kita tandai posisi-posisi `"  siti  NUR aisyah "` (karakter pertama berada di posisi 0):

```
posisi:    0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
karakter:  _ _ s i t i _ _ _ N  U  R  _  a  i  s  y  a  h  _
awal kata?     ^             ^           ^
```

(Di sini `_` menandai spasi.) Awal kata berada di posisi 2, 9, dan 13, jadi ada tiga kata, dan inisialnya adalah `s`, `N`, `a` dalam huruf kapital: `S.N.A.`. Untuk `tidy`, setiap karakter diperlakukan dengan salah satu dari tiga cara: awal kata dijadikan kapital dan, kecuali untuk kata pertama, didahului satu spasi; huruf lain dijadikan kecil; dan spasi dibuang (satu spasi itu dibuat ulang pada awal kata berikutnya). Contoh tepi: nama kosong tidak memiliki kata dan inisialnya `""`; nama yang terdiri atas tiga spasi tidak memiliki kata.

4.13.3 Kelas, field, dan metode



Definisi “memulai kata” dipakai tiga kali, jadi ia menjadi satu metode privat, `startsWord(i)`: aturan yang ditulis sekali tidak mungkin ditulis dalam tiga cara yang sedikit berbeda.

4.13.4 Penyelesaian Java

Kode 4.51 adalah kelasnya dan **Kode 4.52** driver-nya.

Kode 4.51 `PersonName.java`: awal kata ditemukan karakter demi karakter.

```java
/**
 * Soal 4.11: nama seseorang seperti yang diketik pelanggan, mungkin dengan
 * spasi berlebih dan huruf kapital acak. Sebuah kata dimulai pada karakter yang
 * bukan spasi dan merupakan karakter pertama atau terletak sesudah spasi.
 */
public class PersonName {
    private String raw;

    public PersonName(String raw) {
        this.raw = raw;
    }

    private boolean startsWord(int i) {
        return raw.charAt(i) != ' ' && (i == 0 || raw.charAt(i - 1) == ' ');
    }

    public int wordCount() {
        int count = 0;
        for (int i = 0; i < raw.length(); i++) {
            if (startsWord(i)) {
                count++;
            }
        }
        return count;
    }

    public String initials() {
        String result = "";
        for (int i = 0; i < raw.length(); i++) {
            if (startsWord(i)) {
                result = result + Character.toUpperCase(raw.charAt(i)) + ".";
            }
        }
        return result;
    }
```

Kode 4.51 (Lanjutan) *PersonName.java: awal kata ditemukan karakter demi karakter.*

```
    // "  siti   NUR aisyah " menjadi "Siti Nur Aisyah".
    public String tidy() {
        String result = "";
        for (int i = 0; i < raw.length(); i++) {
            char c = raw.charAt(i);
            if (startsWord(i)) {
                if (result.length() > 0) {
                    result = result + " ";
                }
                result = result + Character.toUpperCase(c);
            } else if (c != ' ') {
                result = result + Character.toLowerCase(c);
            }
        }
        return result;
    }
}
```

Kode 4.52 `PersonNameTest.java`: nama yang berantakan dan tiga kasus tepi.

```
/** Driver uji sederhana untuk PersonName: jalankan dengan "java PersonNameTest". */
public class PersonNameTest {
    private static int passed = 0;
    private static int failed = 0;

    private static void check(String label, Object expected, Object actual) {
        if (expected.equals(actual)) {
            passed++;
            System.out.println("PASS " + label);
        } else {
            failed++;
            System.out.println("FAIL " + label + ": expected " + expected + ", got " +
actual);
        }
    }

    public static void main(String[] args) {
        PersonName name = new PersonName("  siti   NUR aisyah ");
        System.out.println("[" + name.tidy() + "] " + name.initials() + " " +
name.wordCount() + " words");

        check("tidy", "Siti Nur Aisyah", name.tidy());
        check("initials", "S.N.A.", name.initials());
        check("word count", 3, name.wordCount());
        check("one word", "B.", new PersonName("budi").initials());
        check("empty name", "", new PersonName("").initials());
        check("only spaces", 0, new PersonName("   ").wordCount());
        System.out.println(passed + " passed, " + failed + " failed");
    }
}
```

```
[Siti Nur Aisyah] S.N.A. 3 words
PASS tidy
PASS initials
PASS word count
PASS one word
PASS empty name
PASS only spaces
6 passed, 0 failed
```

Kurung siku pada baris pertama ada untuk memperlihatkan spasi di depan dan di belakang: ternyata tidak ada.

4.13.5 Penyelesaian Python

Kode 4.53 memakai pengindeksan, `self._raw[i]`, di tempat Java memakai `charAt(i)`.

Kode 4.53 `person_name.py`: Soal 4.11 dalam Python.

```
"""Soal 4.11: nama seseorang seperti yang diketik pelanggan, mungkin dengan
spasi berlebih dan huruf kapital acak. Sebuah kata dimulai pada karakter yang
bukan spasi dan merupakan karakter pertama atau terletak sesudah spasi.
"""
import unittest


class PersonName:
    def __init__(self, raw):
        self._raw = raw

    def _starts_word(self, i):
        return self._raw[i] != " " and (i == 0 or self._raw[i - 1] == " ")

    def word_count(self):
        count = 0
        for i in range(len(self._raw)):
            if self._starts_word(i):
                count += 1
        return count

    def initials(self):
        result = ""
        for i in range(len(self._raw)):
            if self._starts_word(i):
                result = result + self._raw[i].upper() + "."
        return result

    def tidy(self):
        # "  siti   NUR aisyah " menjadi "Siti Nur Aisyah".
        result = ""
        for i in range(len(self._raw)):
            c = self._raw[i]
            if self._starts_word(i):
                if len(result) > 0:
                    result = result + " "
                result = result + c.upper()
            elif c != " ":
                result = result + c.lower()
        return result
```

Kode 4.53 (Lanjutan) *person_name.py: Soal 4.11 dalam Python.*

```
class TestPersonName(unittest.TestCase):
    def test_messy_name(self):
        name = PersonName("  siti   NUR aisyah ")
        self.assertEqual(name.tidy(), "Siti Nur Aisyah")
        self.assertEqual(name.initials(), "S.N.A.")
        self.assertEqual(name.word_count(), 3)

    def test_small_cases(self):
        self.assertEqual(PersonName("budi").initials(), "B.")
        self.assertEqual(PersonName("").initials(), "")
        self.assertEqual(PersonName("   ").word_count(), 0)


if __name__ == "__main__":
    name = PersonName("  siti   NUR aisyah ")
    print(f"[{name.tidy()}] {name.initials()} {name.word_count()} words")
    unittest.main(verbosity=2)
```

```
[Siti Nur Aisyah] S.N.A. 3 words
test_messy_name (__main__.TestPersonName.test_messy_name) ... ok
test_small_cases (__main__.TestPersonName.test_small_cases) ... ok

----------------------------------------------------------------------
Ran 2 tests in 0.000s

OK
```

4.13.6 Baris demi baris

- `return raw.charAt(i) != ' ' && (i == 0 || raw.charAt(i - 1) == ' ');` — definisinya, kata demi kata. Urutan di dalam tanda kurung penting: ketika `i` bernilai 0, `i == 0` sudah `true`, dan Java tidak mengevaluasi sisi kanan `||`, sehingga `raw.charAt(-1)`, yang akan membuat program macet, tidak pernah dicoba. Evaluasi *hubung-pendek* (*short-circuit*) ini adalah fitur yang dapat Anda andalkan di Java maupun di Python.
- `' '` dalam tanda petik tunggal adalah `char`, satu karakter; `" "` dalam tanda petik ganda adalah `String`. `charAt` mengembalikan `char`, jadi ia dibandingkan dengan `' '`. Python tidak memiliki tipe karakter tersendiri: `self._raw[i]` adalah string sepanjang satu, yang dibandingkan dengan `" "`.
- `Character.toUpperCase(c)` dan `Character.toLowerCase(c)` di Java, `c.upper()` dan `c.lower()` di Python, mengubah satu huruf dan membiarkan karakter lain apa adanya.
- `if (result.length() > 0) { result = result + " "; }` — sebuah spasi diletakkan *sebelum* setiap kata kecuali kata pertama, yang lebih sederhana daripada berusaha membuang spasi di belakang pada akhirnya.

4.13.7 Penyelesaian keliru yang menggoda

Kedua bahasa memiliki metode yang memecah string menjadi kata-kata, dan menggoda sekali memakainya dengan satu spasi sebagai pemisah. **Kode 4.54** memperlihatkan apa yang terjadi. (Perulangan `for (String word : …)` mengunjungi setiap potongan bergiliran; larik potongan seperti itu adalah pokok bahasan Bab 5.)

Kode 4.54 `WrongInitials.java`: `split(" ")` menghasilkan kata-kata kosong, dan `charAt(0)` dari kata kosong membuat program macet.

```java
/** split(" ") mengubah setiap spasi berlebih menjadi kata kosong, dan "".charAt(0) gagal.
*/
public class WrongInitials {
    public static void main(String[] args) {
        String raw = "  siti   NUR aisyah ";
        String initials = "";
        for (String word : raw.split(" ")) {
            System.out.print("[" + word + "] ");
        }
        System.out.println();
        for (String word : raw.split(" ")) {
            initials = initials + Character.toUpperCase(word.charAt(0)) + ".";
        }
        System.out.println(initials);
    }
}
```

```
[] [] [siti] [] [] [NUR] [aisyah]
Exception in thread "main" java.lang.StringIndexOutOfBoundsException: Index 0 out of bounds
for length 0
    at java.base/jdk.internal.util.Preconditions$1.apply(Preconditions.java:55)
    at java.base/jdk.internal.util.Preconditions$1.apply(Preconditions.java:52)
    at java.base/jdk.internal.util.Preconditions$4.apply(Preconditions.java:213)
    at java.base/jdk.internal.util.Preconditions$4.apply(Preconditions.java:210)
    at java.base/jdk.internal.util.Preconditions.outOfBounds(Preconditions.java:98)
    at
java.base/jdk.internal.util.Preconditions.outOfBoundsCheckIndex(Preconditions.java:106)
    at java.base/jdk.internal.util.Preconditions.checkIndex(Preconditions.java:302)
    at java.base/java.lang.String.checkIndex(String.java:4822)
    at java.base/java.lang.StringLatin1.charAt(StringLatin1.java:46)
    at java.base/java.lang.String.charAt(String.java:1544)
    at WrongInitials.main(WrongInitials.java:11)
```

Memecah pada *setiap* spasi tunggal menghasilkan potongan kosong di antara dua spasi yang bersebelahan dan sebelum spasi di depan. Potongan kosong pertama tidak memiliki karakter di posisi 0, dan program berhenti dengan `StringIndexOutOfBoundsException`. Baris terakhir pesan panjang itu, `at WrongInitials.main(WrongInitials.java:11)`, menyebutkan berkas dan barisnya: dalam *stack trace* seperti ini, bacalah lebih dulu baris terbawah yang menunjuk kode Anda sendiri. Python gagal dengan cara yang sama (**Kode 4.55**).

Kode 4.55 `wrong_initials.py`: kata-kata kosong yang sama dalam Python, dan sebuah `IndexError`.

```python
"""split(" ") mengubah setiap spasi berlebih menjadi kata kosong, dan ""[0] gagal."""
raw = "  siti   NUR aisyah "
print(raw.split(" "))
initials = ""
for word in raw.split(" "):
    initials = initials + word[0].upper() + "."
print(initials)
```

```
['', '', 'siti', '', '', 'NUR', 'aisyah', '']
Traceback (most recent call last):
  File "/home/reader/Chapter04/python/wrong_initials.py", line 6, in <module>
    initials = initials + word[0].upper() + "."
                          ~~~~^^^
IndexError: string index out of range
```

(Jalur pada baris `File` adalah di mana pun berkas itu berada di komputer Anda.) Daftar yang tercetak memperlihatkan biang keladinya: `''` adalah string kosong. Kedua bahasa *memang* memiliki jalan pintas yang benar — `split()` tanpa argumen di Python, dan

`trim().split("\\s+")` di Java — tetapi Anda sebaiknya memahami versi karakter demi karakter lebih dulu, karena itulah yang dikerjakan jalan-jalan pintas itu di dalamnya.

> **UJI DIRI.** Tambahkan metode `lastNameFirst()` yang mengembalikan nama yang sudah rapi dengan kata terakhir di depan, diikuti koma: `"Aisyah, Siti Nur"`. Apa yang dikembalikannya untuk `"budi"`, yang hanya satu kata?
>
> *Jawaban.* Ambil `String t = tidy();`, cari spasi terakhir dengan `int k = t.lastIndexOf(' ');` dan, bila `k` bernilai -1 (tidak ada spasi), kembalikan `t` tanpa perubahan; bila tidak, kembalikan `t.substring(k + 1) + ", " + t.substring(0, k)`. Untuk `"budi"` tidak ada spasi, jadi hasilnya `"Budi"`. Di Python, `t.rfind(" ")` memainkan peran `lastIndexOf`, dan irisan `t[k + 1:]` serta `t[:k]` memainkan peran `substring`.

4.14 Soal 4.12 — Aturan Kata Sandi

4.14.1 Soalnya

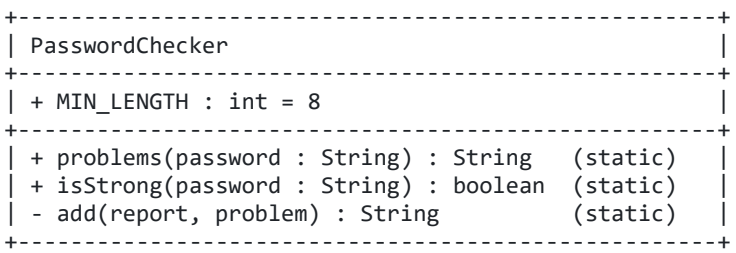
> **SOAL 4.12.** Sebelum `changePassword` dari Soal 4.3 menerima kata sandi baru, metode itu seharusnya memeriksanya. Tulislah kelas `PasswordChecker` dengan metode statis `problems(password)` yang mengembalikan teks berisi setiap aturan yang dilanggar kata sandi itu, dipisahkan dengan `"; "`, atau string kosong bila tidak ada yang dilanggar, dan metode statis `isStrong(password)` yang mengembalikan `true` tepat ketika tidak ada masalah. Aturannya: sekurang-kurangnya 8 karakter (`"too short"`), sekurang-kurangnya satu angka (`"no digit"`), sekurang-kurangnya satu huruf kapital (`"no capital letter"`), sekurang-kurangnya satu huruf kecil (`"no small letter"`).

4.14.2 Memahaminya dengan tangan

`sesame`: 6 karakter, terlalu pendek; tanpa angka; tanpa huruf kapital; ada huruf kecilnya. Jadi: `too short; no digit; no capital letter`. `1sesame`: 7 karakter, masih terlalu pendek; angka `1` dihitung; tanpa huruf kapital. Jadi: `too short; no capital letter`. `Sesame2026`: 10 karakter, huruf kapital `S`, huruf kecil, angka — kuat. Kasus-kasus tepi terletak di batas panjang: `Abcdefg1` tepat 8 karakter dan kuat; `Abcdef1` 7 karakter dan hanya terlalu pendek. Dan kata sandi kosong melanggar keempat aturan.

Perhatikan *bagaimana* kita memeriksa `1sesame` untuk angka: kita melihat *setiap* karakter, dan putusan "memiliki angka" sudah pasti pada karakter pertama, sedangkan putusan "tidak memiliki huruf kapital" baru dapat diberikan setelah karakter *terakhir*. Perulangan yang memutuskan "ada karakter yang bersifat P" boleh berhenti begitu menemukannya; perulangan yang memutuskan "tidak ada karakter yang bersifat P" harus memeriksa semuanya.

4.14.3 Kelas, field, dan metode



Rencana untuk `problems`: satu perulangan yang *menghitung* angka, huruf kapital, dan huruf kecil; lalu, *sesudah* perulangan, empat pernyataan `if` yang mengubah hitungan-hitungan itu menjadi laporan.

4.14.4 Penyelesaian Java

Kode 4.56 adalah pemeriksanya dan **Kode 4.57** driver-nya.

Kode 4.56 `PasswordChecker.java`: menghitung dalam satu perulangan, memutuskan sesudahnya.

```
/**
 * Soal 4.12: memeriksa kata sandi usulan terhadap empat aturan dan
 * menjelaskan setiap aturan yang dilanggarnya. Satu kali lintasan atas karakter
 * menghitung angka, huruf kapital, dan huruf kecil; putusan datang sesudah gelung.
 */
public class PasswordChecker {
    public static final int MIN_LENGTH = 8;

    // Menambahkan satu masalah lagi ke laporan, dengan "; " di antara masalah.
    private static String add(String report, String problem) {
        if (report.isEmpty()) {
            return problem;
        }
        return report + "; " + problem;
    }

    public static String problems(String password) {
        int digits = 0;
        int upper = 0;
        int lower = 0;
        for (int i = 0; i < password.length(); i++) {
            char c = password.charAt(i);
            if (Character.isDigit(c)) {
                digits++;
            } else if (Character.isUpperCase(c)) {
                upper++;
            } else if (Character.isLowerCase(c)) {
                lower++;
            }
        }
        String report = "";
        if (password.length() < MIN_LENGTH) {
            report = add(report, "too short");
        }
        if (digits == 0) {
            report = add(report, "no digit");
        }
        if (upper == 0) {
            report = add(report, "no capital letter");
        }
        if (lower == 0) {
            report = add(report, "no small letter");
        }
        return report;
    }
```

Kode 4.56 (Lanjutan) *PasswordChecker.java: menghitung dalam satu perulangan, memutuskan sesudahnya.*

```
    public static boolean isStrong(String password) {
        return problems(password).isEmpty();
    }
}
```

Kode 4.57 `PasswordCheckerTest.java`: angka di salah satu ujung, batas panjang, dan kata sandi kosong.

```java
/** Driver uji sederhana untuk PasswordChecker: jalankan dengan "java PasswordCheckerTest".
*/
public class PasswordCheckerTest {
    private static int passed = 0;
    private static int failed = 0;

    private static void check(String label, Object expected, Object actual) {
        if (expected.equals(actual)) {
            passed++;
            System.out.println("PASS " + label);
        } else {
            failed++;
            System.out.println("FAIL " + label + ": expected " + expected + ", got " +
actual);
        }
    }

    public static void main(String[] args) {
        System.out.println("sesame     -> " + PasswordChecker.problems("sesame"));
        System.out.println("1sesame    -> " + PasswordChecker.problems("1sesame"));
        System.out.println("Sesame2026 -> " + PasswordChecker.isStrong("Sesame2026"));

        check("strong", true, PasswordChecker.isStrong("Sesame2026"));
        check("digit only at the start still counts", "too short; no capital letter",
                PasswordChecker.problems("1sesame"));
        check("digit only at the end still counts", true,
PasswordChecker.isStrong("Laundry7"));
        check("empty password", "too short; no digit; no capital letter; no small letter",
                PasswordChecker.problems(""));
        check("exactly eight is long enough", true, PasswordChecker.isStrong("Abcdefg1"));
        check("seven is too short", "too short", PasswordChecker.problems("Abcdef1"));
        System.out.println(passed + " passed, " + failed + " failed");
    }
}
```

```
sesame     -> too short; no digit; no capital letter
1sesame    -> too short; no capital letter
Sesame2026 -> true
PASS strong
PASS digit only at the start still counts
PASS digit only at the end still counts
PASS empty password
PASS exactly eight is long enough
PASS seven is too short
6 passed, 0 failed
```

4.14.5 Penyelesaian Python

Dalam **Kode 4.58** kedua fungsi berada di tingkat teratas modul, cara Python menulis apa yang di Java disebut metode statis.

Kode 4.58 `password_checker.py`: Soal 4.12 dalam Python.

```
"""Soal 4.12: memeriksa kata sandi usulan terhadap empat aturan dan
menjelaskan setiap aturan yang dilanggarnya. Satu kali lintasan atas karakter
menghitung angka, huruf kapital, dan huruf kecil; putusan datang sesudah gelung.
"""
import unittest

MIN_LENGTH = 8


def _add(report, problem):
    # Menambahkan satu masalah lagi ke laporan, dengan "; " di antara masalah.
    if report == "":
        return problem
    return report + "; " + problem


def problems(password):
    digits = 0
    upper = 0
    lower = 0
    for c in password:
        if c.isdigit():
            digits += 1
        elif c.isupper():
            upper += 1
        elif c.islower():
            lower += 1
    report = ""
    if len(password) < MIN_LENGTH:
        report = _add(report, "too short")
    if digits == 0:
        report = _add(report, "no digit")
    if upper == 0:
        report = _add(report, "no capital letter")
    if lower == 0:
        report = _add(report, "no small letter")
    return report


def is_strong(password):
    return problems(password) == ""
```

Kode 4.58 (Lanjutan) *password_checker.py: Soal 4.12 dalam Python.*

```
class TestPasswordChecker(unittest.TestCase):
    def test_strong(self):
        self.assertTrue(is_strong("Sesame2026"))
        self.assertTrue(is_strong("Laundry7"))
        self.assertTrue(is_strong("Abcdefg1"))

    def test_reports(self):
        self.assertEqual(problems("1sesame"), "too short; no capital letter")
        self.assertEqual(problems(""), "too short; no digit; no capital letter; no small
letter")
        self.assertEqual(problems("Abcdef1"), "too short")


if __name__ == "__main__":
    print(f"sesame     -> {problems('sesame')}")
    print(f"1sesame    -> {problems('1sesame')}")
    print(f"Sesame2026 -> {is_strong('Sesame2026')}")
    unittest.main(verbosity=2)
```

```
sesame     -> too short; no digit; no capital letter
1sesame    -> too short; no capital letter
Sesame2026 -> True
test_reports (__main__.TestPasswordChecker.test_reports) ... ok
test_strong (__main__.TestPasswordChecker.test_strong) ... ok

----------------------------------------------------------------------
Ran 2 tests in 0.000s

OK
```

Java mencetak nilai Boolean sebagai `true` dan Python sebagai `True`; nilainya sama.

4.14.6 Baris demi baris

- `if (Character.isDigit(c)) { digits++; } else if (Character.isUpperCase(c)) { upper++; } else if (…)` — sebuah karakter paling banyak termasuk satu dari ketiga jenis itu, jadi rantai `else if` tepat, dan setiap pencacah bertambah dengan `++`, "tambah satu". Karakter yang tidak termasuk ketiganya, seperti `!`, sekadar tidak dihitung.
- `private static String add(String report, String problem)` — pemisah `"; "` dibutuhkan *di antara* masalah, bukan sebelum masalah pertama; pembantu ini memutuskannya di satu tempat, sehingga keempat pernyataan `if` tetap sederhana.
- `return problems(password).isEmpty();` — `isStrong` didefinisikan melalui `problems`, sehingga keduanya tidak mungkin berselisih. (Bandingkan dengan Soal 4.5: simpan, atau di sini hitung, setiap fakta satu kali saja.)
- `for c in password:` di Python mengunjungi karakter-karakternya secara langsung, tanpa indeks; `c.isdigit()`, `c.isupper()`, dan `c.islower()` adalah uji-uji karakternya.

Sepatah kata jujur tentang aturan-aturan ini. Aturan tersebut baik untuk berlatih perulangan, tetapi pedoman modern tentang kata sandi sungguhan, seperti NIST Special Publication 800-63B, justru menganjurkan *untuk tidak* memaksakan campuran jenis karakter, dan merekomendasikan kata sandi yang panjang yang diperiksa terhadap daftar kata sandi yang lazim dipakai [9]. Sistem sungguhan juga tidak akan pernah menyimpan atau mencetak kata sandi itu sendiri, hanya *hash* bergaram (*salted hash*) darinya.

4.14.7 Penyelesaian keliru yang menggoda

Kode 4.59 `WrongPassword.java`: perulangan yang mengembalikan putusan setelah karakter pertama.

```java
/** Gelung yang memutuskan setelah hanya melihat karakter pertama. */
public class WrongPassword {
    public static boolean hasDigit(String s) {
        for (int i = 0; i < s.length(); i++) {
            if (Character.isDigit(s.charAt(i))) {
                return true;
            } else {
                return false; // memutuskan terlalu dini
            }
        }
        return false;
    }

    public static void main(String[] args) {
        System.out.println("1sesame:  " + hasDigit("1sesame"));
        System.out.println("Laundry7: " + hasDigit("Laundry7"));
    }
}
```

```
1sesame:  true
Laundry7: false
```

`else { return false; }` di dalam perulangan mengakhiri metode setelah melihat karakter *pertama* saja. `1sesame` lolos karena angkanya kebetulan berada di depan; `Laundry7` gagal, padahal ia memiliki angka. Badan perulangan paling banyak berjalan sekali, jadi sebenarnya ia bukan perulangan sama sekali. *Debugger* memperlihatkannya seketika: melangkahlah melalui `hasDigit("Laundry7")` dan amati bahwa `i` tidak pernah mencapai 1. **Kode 4.60** memperlihatkan bug yang sama dalam Python.

Kode 4.60 `wrong_password.py`: pengembalian terlalu dini yang sama dalam Python.

```python
"""Gelung yang memutuskan setelah hanya melihat karakter pertama."""


def has_digit(s):
    for c in s:
        if c.isdigit():
            return True
        else:
            return False  # memutuskan terlalu dini
    return False


print(f"1sesame:  {has_digit('1sesame')}")
print(f"Laundry7: {has_digit('Laundry7')}")
```

```
1sesame:  True
Laundry7: False
```

Perbaikannya adalah memindahkan `return false` ke *sesudah* perulangan: "bila tidak ada satu karakter pun yang berupa angka, maka tidak ada angka".

> **UJI DIRI.** Tambahkan aturan kelima, `"no symbol"`: kata sandi harus memuat sekurang-kurangnya satu karakter yang bukan huruf dan bukan angka. Manakah di antara `Sesame2026` dan `Sesame-2026` yang kuat menurut lima aturan?
>
> *Jawaban.* Tambahkan pencacah `symbols` dan `else { symbols++; }` terakhir pada rantai di dalam perulangan (di Java, `Character.isLetterOrDigit(c)` juga bisa dipakai), dan sesudah perulangan `if (symbols == 0) { report = add(report, "no symbol"); }`. Maka `Sesame2026` melaporkan `no symbol`, dan `Sesame-2026` kuat.

4.15 Soal 4.13 — Menelusuri Perulangan dengan Tangan

4.15.1 Soalnya

> **SOAL 4.13.** Tanpa menjalankan program, telusurilah dua perulangan berikut dengan tabel yang memiliki satu baris per iterasi: (a) perulangan yang menghitung jumlah angka-angka dari $n = 4071$ dengan berulang kali mengambil angka terakhir `n % 10` lalu membuangnya dengan `n / 10`; (b) algoritma Euclid untuk faktor persekutuan terbesar dari $a = 84$ dan $b = 36$, yang mengulang "$r = a \bmod b$; $a = b$; $b = r$" selama $b \neq 0$ lalu mengembalikan a. Kemudian tulislah program yang keluaran jejaknya mereproduksi tabel-tabel Anda, dan ujilah kedua metodenya.

4.15.2 Memahaminya dengan tangan

Sebuah *tabel jejak* (*trace table*) memiliki satu kolom untuk setiap variabel yang berubah dan satu baris untuk setiap lintasan melalui perulangan, diisi *setelah* badan perulangan berjalan satu kali.

Tabel ini adalah versi kertas dari jendela pengamatan *debugger*, dan merupakan satu-satunya cara terbaik untuk memahami sebuah perulangan.

(a) Mulailah dengan `total = 0` dan `n = 4071`. Pada setiap lintasan, `digit = n % 10` adalah angka terakhir, `total` bertambah sebesar angka itu, dan `n / 10` (pembagian bilangan bulat) membuangnya.

Tabel 4.6 Jejak perulangan jumlah angka untuk $n = 4071$: empat lintasan, satu per angka, berakhir ketika n mencapai 0.

Langkah	`digit` $= n \bmod 10$	`total`	n Setelah `n / 10`
1	$4071 \bmod 10 = 1$	$0 + 1 = 1$	407
2	$407 \bmod 10 = 7$	$1 + 7 = 8$	40
3	$40 \bmod 10 = 0$	$8 + 0 = 8$	4
4	$4 \bmod 10 = 4$	$8 + 4 = 12$	0

Setelah langkah 4 kondisi `n > 0` bernilai salah dan perulangan berhenti: jumlah angkanya 12, dan memang $4 + 0 + 7 + 1 = 12$. Perulangan berjalan sekali per angka, jadi bilangan d angka membutuhkan d lintasan. Untuk $n = 0$ perulangan berjalan nol kali dan mengembalikan 0.

(b) Algoritma Euclid bertumpu pada satu fakta: setiap bilangan yang membagi habis a dan b juga membagi habis $a \bmod b$ (yang sama dengan a dikurangi suatu kelipatan b), dan sebaliknya, sehingga

$$\gcd(a, b) = \gcd(b,\ a \bmod b), \qquad \gcd(a, 0) = a \tag{4.8}$$

Setiap lintasan perulangan menerapkan Persamaan (4.8) satu kali [10]:

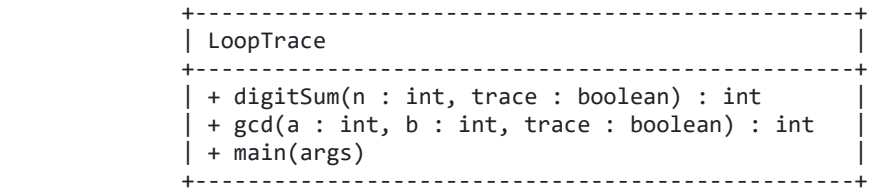
Tabel 4.7 Jejak algoritma Euclid untuk $a = 84$, $b = 36$: setelah dua lintasan $b = 0$ dan jawabannya $a = 12$.

a	b	$r = a \bmod b$	a Baru, b baru
84	36	$84 - 2 \cdot 36 = 12$	36, 12
36	12	$36 - 3 \cdot 12 = 0$	12, 0

Jadi $\gcd(84,36) = 12$; periksa: $84 = 7 \cdot 12$ dan $36 = 3 \cdot 12$, dan 7 serta 3 tidak memiliki faktor persekutuan. Bila kita memulai dengan bilangan yang lebih kecil lebih dulu, gcd(36,84), lintasan pertama memberi $r = 36 \bmod 84 = 36$ dan sekadar menukar kedua bilangan — satu lintasan terbuang, tetapi jawabannya sama.

4.15.3 Kelas, field, dan metode

Tidak ada yang perlu diingat di antara pemanggilan, jadi `LoopTrace` memuat dua metode statis, masing-masing dengan parameter Boolean `trace` yang menyalakan atau mematikan pencetakan tabel. Dengan begitu metode yang sama sekaligus *dijelaskan* (dengan `trace = true`) dan *diuji* (dengan `trace = false`).



4.15.4 Penyelesaian Java

Menjalankan **Kode 4.61** mencetak kedua tabel.

Kode 4.61 `LoopTrace.java`: dua perulangan yang mencetak tabel jejaknya sendiri.

```java
/**
 * Soal 4.13: dua gelung while yang ditelusuri dengan tangan di dalam teks.
 * Dengan trace = true setiap gelung mencetak satu baris per iterasi, sehingga
 * tabel yang tercetak dapat dibandingkan dengan tabel yang ditulis di kertas.
 */
public class LoopTrace {

    public static int digitSum(int n, boolean trace) {
        int total = 0;
        int step = 0;
        if (trace) {
            System.out.println("step  digit  total     n");
        }
        while (n > 0) {
            int digit = n % 10; // angka terakhir
            total = total + digit;
            n = n / 10;         // buang angka terakhir
            step++;
            if (trace) {
                System.out.printf("%4d %6d %6d %5d%n", step, digit, total, n);
            }
        }
        return total;
    }

    // Algoritma Euclid: gcd(a, b) = gcd(b, a % b), dan gcd(a, 0) = a.
    public static int gcd(int a, int b, boolean trace) {
        if (trace) {
            System.out.println("   a    b  a % b");
        }
        while (b != 0) {
            int r = a % b;
            if (trace) {
                System.out.printf("%4d %4d %6d%n", a, b, r);
            }
            a = b;
            b = r;
        }
        return a;
    }

    public static void main(String[] args) {
        System.out.println("digitSum(4071) = " + digitSum(4071, true));
        System.out.println("gcd(84, 36) = " + gcd(84, 36, true));
    }
}
```

```
step  digit  total     n
   1      1      1   407
   2      7      8    40
   3      0      8     4
   4      4     12     0
digitSum(4071) = 12
   a    b  a % b
  84   36     12
  36   12      0
gcd(84, 36) = 12
```

Baris-baris yang tercetak adalah **Tabel 4.6** dan **Tabel 4.7**. Perhatikan bahwa setiap tabel dicetak *sebelum* baris `digitSum(4071) = 12`: untuk membangun baris itu, Java harus memanggil `digitSum` lebih dulu, dan pemanggilan itu mencetak baris-barisnya di tengah jalan. **Kode 4.62** memeriksa kasus-kasus tepinya.

Kode 4.62 `LoopTraceTest.java`: nol, satu angka, nol di belakang, dan kedua urutan untuk Euclid.

```java
/** Driver uji sederhana untuk LoopTrace: jalankan dengan "java LoopTraceTest". */
public class LoopTraceTest {
    private static int passed = 0;
    private static int failed = 0;

    private static void check(String label, Object expected, Object actual) {
        if (expected.equals(actual)) {
            passed++;
            System.out.println("PASS " + label);
        } else {
            failed++;
            System.out.println("FAIL " + label + ": expected " + expected + ", got " +
actual);
        }
    }

    public static void main(String[] args) {
        check("digitSum(4071)", 12, LoopTrace.digitSum(4071, false));
        check("digitSum(9)", 9, LoopTrace.digitSum(9, false));
        check("digitSum(0) runs zero times", 0, LoopTrace.digitSum(0, false));
        check("digitSum(1000)", 1, LoopTrace.digitSum(1000, false));
        check("gcd(84, 36)", 12, LoopTrace.gcd(84, 36, false));
        check("gcd(36, 84) swaps first", 12, LoopTrace.gcd(36, 84, false));
        check("gcd(17, 5)", 1, LoopTrace.gcd(17, 5, false));
        check("gcd(5, 0)", 5, LoopTrace.gcd(5, 0, false));
        System.out.println(passed + " passed, " + failed + " failed");
    }
}
```

```
PASS digitSum(4071)
PASS digitSum(9)
PASS digitSum(0) runs zero times
PASS digitSum(1000)
PASS gcd(84, 36)
PASS gcd(36, 84) swaps first
PASS gcd(17, 5)
PASS gcd(5, 0)
8 passed, 0 failed
```

4.15.5 Penyelesaian Python

Kode 4.63 mencetak kedua tabel yang sama.

Kode 4.63 `loop_trace.py`: Soal 4.13 dalam Python, dengan `//` untuk pembagian bilangan bulat.

```
"""Soal 4.13: dua gelung while yang ditelusuri dengan tangan di dalam teks.
Dengan trace=True setiap gelung mencetak satu baris per iterasi, sehingga
tabel yang tercetak dapat dibandingkan dengan tabel yang ditulis di kertas.
"""
import unittest


def digit_sum(n, trace):
    total = 0
    step = 0
    if trace:
        print("step  digit  total     n")
    while n > 0:
        digit = n % 10  # angka terakhir
        total = total + digit
        n = n // 10     # buang angka terakhir (// menjaga n tetap bilangan bulat)
        step += 1
        if trace:
            print(f"{step:4d} {digit:6d} {total:6d} {n:5d}")
    return total


def gcd(a, b, trace):
    # Algoritma Euclid: gcd(a, b) = gcd(b, a % b), dan gcd(a, 0) = a.
    if trace:
        print("   a    b  a % b")
    while b != 0:
        r = a % b
        if trace:
            print(f"{a:4d} {b:4d} {r:6d}")
        a = b
        b = r
    return a


class TestLoopTrace(unittest.TestCase):
    def test_digit_sum(self):
        self.assertEqual(digit_sum(4071, False), 12)
        self.assertEqual(digit_sum(9, False), 9)
        self.assertEqual(digit_sum(0, False), 0)
        self.assertEqual(digit_sum(1000, False), 1)
```

Kode 4.63 (Lanjutan) *loop_trace.py: Soal 4.13 dalam Python, dengan // untuk pembagian bilangan bulat.*

```
    def test_gcd(self):
        self.assertEqual(gcd(84, 36, False), 12)
        self.assertEqual(gcd(36, 84, False), 12)
        self.assertEqual(gcd(17, 5, False), 1)
        self.assertEqual(gcd(5, 0, False), 5)


if __name__ == "__main__":
    print(f"digit_sum(4071) = {digit_sum(4071, True)}")
    print(f"gcd(84, 36) = {gcd(84, 36, True)}")
    unittest.main(verbosity=2)
```

```
step  digit  total     n
   1      1      1   407
   2      7      8    40
   3      0      8     4
   4      4     12     0
digit_sum(4071) = 12
   a    b  a % b
  84   36     12
  36   12      0
gcd(84, 36) = 12
test_digit_sum (__main__.TestLoopTrace.test_digit_sum) ... ok
test_gcd (__main__.TestLoopTrace.test_gcd) ... ok

----------------------------------------------------------------------
Ran 2 tests in 0.000s

OK
```

4.15.6 Baris demi baris

- `while (n > 0)` — kondisinya diperiksa *sebelum* setiap lintasan, sehingga untuk `n = 0` badannya tidak pernah berjalan; itulah sebabnya `digitSum(0)` dengan benar bernilai 0.
- `n = n / 10;` — pembagian bilangan bulat di Java karena kedua operannya `int`. Di Python langkah yang sama *harus* ditulis `n // 10`; lihat di bawah.
- `System.out.printf("%4d %6d %6d %5d%n", …)` — `printf` adalah `String.format` dan `print` sekaligus; lebar-lebar itu menjajarkan bilangan di bawah judul kolomnya.
- Dalam `gcd`, ketiga penugasan `r = a % b; a = b; b = r;` harus terjadi dengan urutan ini: variabel pembantu `r` menjaga sisa pembagian tetap aman selagi `a` dan `b` bergeser.
- Kita menamai jumlah berjalan `total`, bukan `sum`, di kedua bahasa: di Python `sum` adalah nama fungsi bawaan, dan memakainya sebagai variabel menyembunyikan fungsi itu.

4.15.7 Penyelesaian keliru yang menggoda

Kode 4.64 mengubah satu karakter saja pada kondisinya.

Kode 4.64 `WrongLoop.java`: kondisi `n > 9` berhenti satu lintasan terlalu awal.

```java
/** Kondisi gelung yang meleset satu: angka terakhir yang tersisa tidak pernah dijumlahkan.
*/
public class WrongLoop {
    public static void main(String[] args) {
        int n = 4071;
        int total = 0;
        while (n > 9) {
            total = total + n % 10;
            n = n / 10;
        }
        System.out.println("digit sum of 4071: " + total);
    }
}
```

```
digit sum of 4071: 8
```

Kondisi `n > 9` terasa benar — "selama masih lebih dari satu angka" — tetapi ia berhenti ketika $n = 4$, sebelum angka terakhir dijumlahkan: jawabannya 8, bukan 12. Tabel jejak langsung menangkap kesalahan *meleset satu* (*off-by-one*) seperti ini: tabelnya hanya tiga baris, bukan empat. **Kode 4.65** mengulanginya dalam Python dan menambahkan jebakan kedua yang khas Python.

Kode 4.65 `wrong_loop.py`: kondisi yang meleset satu, dan `/` padahal yang dimaksud `//`.

```
"""Dua jumlah angka yang rusak: kondisi meleset satu, dan / padahal yang dimaksud //."""
n = 4071
total = 0
while n > 9:
    total = total + n % 10
    n = n // 10
print(f"n > 9 instead of n > 0: {total}")

n = 4071
total = 0
steps = 0
while n > 0:
    total = total + n % 10
    n = n / 10  # pembagian sejati: n tidak pernah cepat menjadi tepat 0
    steps += 1
print(f"/ instead of //: total {total} after {steps} steps")
```

```
n > 9 instead of n > 0: 8
/ instead of //: total 13.333333333333357 after 328 steps
```

Dengan `/`, Python membagi secara eksak: $4071/10 = 407.1$, lalu 40.71, dan seterusnya. Bilangan itu tidak pernah tepat menjadi 0 dalam beberapa langkah; ia hanya mengecil sampai terlalu kecil untuk dinyatakan oleh `double`, yang memakan 328 lintasan, sementara "angka-angka" `n % 10` berupa pecahan yang jumlahnya tidak bermakna. Inilah bayangan cermin jebakan pembagian bilangan bulat Java dari Soal 4.1: di Java, `/` di antara bilangan bulat kadang mengejutkan; di Python, `/` di antara bilangan bulat *tidak pernah* merupakan pembagian bilangan bulat.

> **UJI DIRI.** Telusurilah `gcd(1071, 462)` dengan tangan. Berapa lintasan yang dijalani perulangannya, dan apa hasilnya?
>
> *Jawaban.* $1071 \bmod 462 = 1071 - 2 \cdot 462 = 147$, sehingga (a, b) menjadi $(462,147)$; $462 \bmod 147 = 462 - 3 \cdot 147 = 21$, sehingga $(147,21)$; $147 \bmod 21 = 0$, sehingga $(21,0)$. Tiga lintasan, dan $\gcd(1071,462) = 21$. Periksa: $1071 = 51 \cdot 21$ dan $462 = 22 \cdot 21$, dan 51 serta 22 tidak memiliki faktor persekutuan.

4.16 Soal 4.14 — Tahun Kabisat dan Hari Berikutnya

4.16.1 Soalnya

> **SOAL 4.14.** Tulislah kelas `SimpleDate` untuk tanggal dalam kalender Gregorian dengan *field* tahun, bulan, dan hari. Kelas ini memiliki metode statis `isLeapYear(year)`, metode statis `daysInMonth(year, month)`, metode `isValid()` yang menyatakan apakah tanggal itu ada, metode `nextDay()` yang mengembalikan `SimpleDate` baru untuk hari berikutnya, dan `toString` dalam format internasional `YYYY-MM-DD`, seperti `2024-02-29`. Hitung pula, dengan perulangan, banyaknya tahun kabisat dari 1901 sampai 2100.

4.16.2 Memahaminya dengan tangan

Aturan Gregorian terdiri atas tiga bagian [11]: tahun yang habis dibagi 4 adalah tahun kabisat, *kecuali* tahun yang habis dibagi 100, yang bukan kabisat, *kecuali lagi* tahun yang habis dibagi 400, yang kabisat. Sebagai satu ekspresi Boolean:

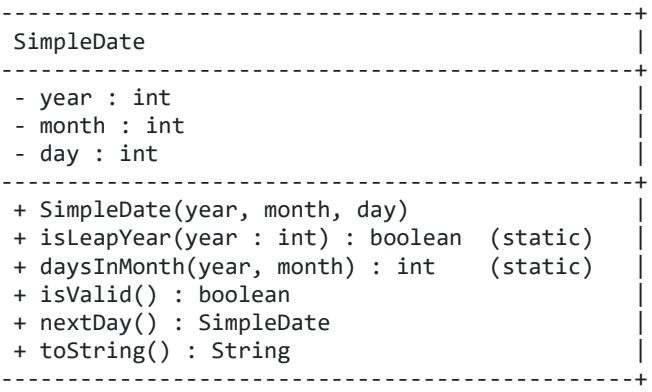
$$\text{kabisat}(y) = (4 \mid y \ \wedge \ 100 \nmid y) \ \vee \ 400 \mid y \tag{4.9}$$

dengan $4 \mid y$ dibaca "4 membagi habis y", yaitu $y \bmod 4 = 0$; $\nmid$ berarti "tidak membagi habis"; $\wedge$ adalah "dan" dan $\vee$ adalah "atau". Jadi 2024 tahun kabisat ($4 \mid 2024$, $100 \nmid 2024$), 2023 bukan, 1900 bukan ($100 \mid 1900$ tetapi $400 \nmid 1900$), dan 2000 kabisat ($400 \mid 2000$).

Menghitung dari 1901 sampai 2100: kelipatan 4 adalah 1904,1908, ... ,2100, yaitu $\frac{2100-1904}{4} + 1 = 49 + 1 = 50$ tahun. Di antaranya, tahun abad 2000 dan 2100 perlu ditengok kedua kalinya: 2000 habis dibagi 400 dan tetap kabisat, tetapi 2100 tidak. Jadi jawabannya $50 - 1 = 49$.

Untuk `nextDay`, ada tiga kasus, diperiksa berurutan: bila hari itu bukan hari terakhir bulannya, tambahkan satu pada harinya; bila tidak, dan bulannya bukan Desember, pindah ke tanggal 1 bulan berikutnya; bila tidak, pindah ke 1 Januari tahun berikutnya. Jadi 28 Februari 2024 diikuti 29 Februari 2024, 28 Februari 2023 diikuti 1 Maret 2023, dan 31 Desember 2026 diikuti 1 Januari 2027.

4.16.3 Kelas, field, dan metode



`isLeapYear` dan `daysInMonth` bersifat statis karena keduanya adalah fakta tentang kalender, bukan tentang satu tanggal tertentu.

4.16.4 Penyelesaian Java

Kode 4.66 adalah kelasnya; **Kode 4.67** lebih dulu menghitung tahun-tahun kabisat dengan perulangan `for` lalu memeriksa contoh-contoh tangan.

Kode 4.66 `SimpleDate.java`: tahun kabisat menurut Persamaan (4.9), panjang bulan dengan `switch`, dan hari berikutnya dalam tiga kasus.

```java
/**
 * Soal 4.14: tanggal dalam kalender Gregorian. Suatu tahun adalah
 * tahun kabisat jika habis dibagi 4, kecuali tahun abad, yang harus
 * habis dibagi 400: 2024 dan 2000 tahun kabisat, 1900 dan 2100 bukan.
 */
public class SimpleDate {
    private int year;
    private int month;
    private int day;

    public SimpleDate(int year, int month, int day) {
        this.year = year;
        this.month = month;
        this.day = day;
    }

    public static boolean isLeapYear(int year) {
        return (year % 4 == 0 && year % 100 != 0) || year % 400 == 0;
    }

    public static int daysInMonth(int year, int month) {
        switch (month) {
            case 2:
                return isLeapYear(year) ? 29 : 28;
            case 4:
            case 6:
            case 9:
            case 11:
                return 30;
            default:
                return 31;
        }
    }

    public boolean isValid() {
        return month >= 1 && month <= 12 && day >= 1 && day <= daysInMonth(year, month);
    }
```

Kode 4.66 (Lanjutan) *SimpleDate.java: tahun kabisat menurut Persamaan (4.9), panjang bulan dengan switch, dan hari berikutnya dalam tiga kasus.*

```java
    public SimpleDate nextDay() {
        if (day < daysInMonth(year, month)) {
            return new SimpleDate(year, month, day + 1);
        }
        if (month < 12) {
            return new SimpleDate(year, month + 1, 1);
        }
        return new SimpleDate(year + 1, 1, 1);
    }

    @Override
    public String toString() {
        return String.format("%04d-%02d-%02d", year, month, day);
    }
}
```

Kode 4.67 `SimpleDateTest.java`: keempat jenis tahun, akhir bulan, akhir tahun, dan tanggal yang tidak sah.

```java
/** Driver uji sederhana untuk SimpleDate: jalankan dengan "java SimpleDateTest". */
public class SimpleDateTest {
    private static int passed = 0;
    private static int failed = 0;

    private static void check(String label, Object expected, Object actual) {
        if (expected.equals(actual)) {
            passed++;
            System.out.println("PASS " + label);
        } else {
            failed++;
            System.out.println("FAIL " + label + ": expected " + expected + ", got " +
actual);
        }
    }

    public static void main(String[] args) {
        int leapYears = 0;
        for (int year = 1901; year <= 2100; year++) {
            if (SimpleDate.isLeapYear(year)) {
                leapYears++;
            }
        }
        System.out.println("Leap years from 1901 to 2100: " + leapYears);

        check("2024 is leap", true, SimpleDate.isLeapYear(2024));
        check("2023 is not", false, SimpleDate.isLeapYear(2023));
        check("1900 is not", false, SimpleDate.isLeapYear(1900));
        check("2000 is", true, SimpleDate.isLeapYear(2000));
        check("April has 30 days", 30, SimpleDate.daysInMonth(2026, 4));
        check("into 29 February", "2024-02-29", new SimpleDate(2024, 2,
28).nextDay().toString());
        check("past 28 February", "2023-03-01", new SimpleDate(2023, 2,
28).nextDay().toString());
        check("new year", "2027-01-01", new SimpleDate(2026, 12, 31).nextDay().toString());
        check("29 Feb 2023 invalid", false, new SimpleDate(2023, 2, 29).isValid());
        check("month 13 invalid", false, new SimpleDate(2026, 13, 1).isValid());
        System.out.println(passed + " passed, " + failed + " failed");
    }
}
```

```
Leap years from 1901 to 2100: 49
PASS 2024 is leap
PASS 2023 is not
PASS 1900 is not
PASS 2000 is
PASS April has 30 days
PASS into 29 February
PASS past 28 February
PASS new year
PASS 29 Feb 2023 invalid
PASS month 13 invalid
10 passed, 0 failed
```

4.16.5 Penyelesaian Python

Kode 4.68 menandai `is_leap_year` dan `days_in_month` dengan `@staticmethod`, cara Python menyatakan bahwa keduanya milik kelas.

Kode 4.68 `simple_date.py`: Soal 4.14 dalam Python, dengan `match` sebagai ganti `switch`.

```python
"""Soal 4.14: tanggal dalam kalender Gregorian. Suatu tahun adalah
tahun kabisat jika habis dibagi 4, kecuali tahun abad, yang harus
habis dibagi 400: 2024 dan 2000 tahun kabisat, 1900 dan 2100 bukan.
"""
import unittest


class SimpleDate:
    def __init__(self, year, month, day):
        self._year = year
        self._month = month
        self._day = day

    @staticmethod
    def is_leap_year(year):
        return (year % 4 == 0 and year % 100 != 0) or year % 400 == 0

    @staticmethod
    def days_in_month(year, month):
        match month:
            case 2:
                return 29 if SimpleDate.is_leap_year(year) else 28
            case 4 | 6 | 9 | 11:
                return 30
            case _:
                return 31

    def is_valid(self):
        return (1 <= self._month <= 12
                and 1 <= self._day <= SimpleDate.days_in_month(self._year, self._month))

    def next_day(self):
        if self._day < SimpleDate.days_in_month(self._year, self._month):
            return SimpleDate(self._year, self._month, self._day + 1)
        if self._month < 12:
            return SimpleDate(self._year, self._month + 1, 1)
        return SimpleDate(self._year + 1, 1, 1)

    def __str__(self):
        return f"{self._year:04d}-{self._month:02d}-{self._day:02d}"
```

Kode 4.68 (Lanjutan) *simple_date.py: Soal 4.14 dalam Python, dengan match sebagai ganti switch.*

```
class TestSimpleDate(unittest.TestCase):
    def test_leap_years(self):
        self.assertTrue(SimpleDate.is_leap_year(2024))
        self.assertFalse(SimpleDate.is_leap_year(2023))
        self.assertFalse(SimpleDate.is_leap_year(1900))
        self.assertTrue(SimpleDate.is_leap_year(2000))

    def test_next_day(self):
        self.assertEqual(str(SimpleDate(2024, 2, 28).next_day()), "2024-02-29")
        self.assertEqual(str(SimpleDate(2023, 2, 28).next_day()), "2023-03-01")
        self.assertEqual(str(SimpleDate(2026, 12, 31).next_day()), "2027-01-01")

    def test_validity(self):
        self.assertEqual(SimpleDate.days_in_month(2026, 4), 30)
        self.assertFalse(SimpleDate(2023, 2, 29).is_valid())
        self.assertFalse(SimpleDate(2026, 13, 1).is_valid())


if __name__ == "__main__":
    leap_years = 0
    for year in range(1901, 2101):
        if SimpleDate.is_leap_year(year):
            leap_years += 1
    print(f"Leap years from 1901 to 2100: {leap_years}")
    unittest.main(verbosity=2)
```

```
Leap years from 1901 to 2100: 49
test_leap_years (__main__.TestSimpleDate.test_leap_years) ... ok
test_next_day (__main__.TestSimpleDate.test_next_day) ... ok
test_validity (__main__.TestSimpleDate.test_validity) ... ok

----------------------------------------------------------------------
Ran 3 tests in 0.000s

OK
```

4.16.6 Baris demi baris

- `return (year % 4 == 0 && year % 100 != 0) || year % 400 == 0;` — Persamaan (4.9) dalam Java. Di Java, `&&` mengikat lebih kuat daripada `||`, jadi tanda kurungnya tidak mutlak diperlukan, tetapi tanda kurung itu memperlihatkan struktur aturannya kepada pembaca.
- `case 4: case 6: case 9: case 11: return 30;` — beberapa label `case` boleh berbagi satu badan. Python menulis `case 4 | 6 | 9 | 11:`.
- `return isLeapYear(year) ? 29 : 28;` — operator kondisional lagi; bentuk Python-nya `29 if … else 28`.
- `month >= 1 && month <= 12 && day >= 1 && day <= daysInMonth(year, month)` — urutannya penting: `daysInMonth` baru dirujuk setelah bulannya diketahui berada di antara 1 dan 12. Python dapat menulis `1 <= self._month <= 12`, sebuah *perbandingan berantai* yang tidak dimiliki Java.
- `String.format("%04d-%02d-%02d", …)` — sekurang-kurangnya empat angka untuk tahun dan dua untuk bulan dan hari, diisi nol di depan. Tanggal yang ditulis dengan cara ini terurut dengan benar sebagai teks biasa, salah satu alasan standar internasional memakainya.

4.16.7 Penyelesaian keliru yang menggoda

Kode 4.69 hanya mempertahankan bagian pertama aturannya.

Kode 4.69 `WrongLeapYear.java`: “habis dibagi 4” saja.

```java
/** "Habis dibagi 4" saja adalah aturan Julian, bukan aturan Gregorian. */
public class WrongLeapYear {
    public static void main(String[] args) {
        int count = 0;
        for (int year = 1901; year <= 2100; year++) {
            if (year % 4 == 0) {
                count++;
            }
        }
        System.out.println("1900 leap? " + (1900 % 4 == 0));
        System.out.println("Leap years from 1901 to 2100: " + count);
    }
}
```

```
1900 leap? true
Leap years from 1901 to 2100: 50
```

"Setiap empat tahun" adalah aturan Julian yang lama, dan itulah yang diingat kebanyakan orang. Aturan itu benar untuk setiap tahun dari 1901 sampai 2099, dan justru karena itulah bug ini selamat dari pengujian sekilas, tetapi salah untuk 1900 dan 2100. Daftar uji yang memuat 1900, 2000, dan 2100 — ketiga jenis tahun abad — langsung menangkapnya. **Kode 4.70** adalah versi Python-nya.

Kode 4.70 `wrong_leap_year.py`: aturan Julian yang sama dalam Python.

```python
"""Habis dibagi 4" saja adalah aturan Julian, bukan aturan Gregorian."""
count = 0
for year in range(1901, 2101):
    if year % 4 == 0:
        count += 1
print(f"1900 leap? {1900 % 4 == 0}")
print(f"Leap years from 1901 to 2100: {count}")
```

```
1900 leap? True
Leap years from 1901 to 2100: 50
```

> **UJI DIRI.** Tulislah metode `dayOfYear()` yang mengembalikan nomor hari di dalam tahunnya, 1 untuk 1 Januari. Pakailah perulangan `for` atas bulan-bulan sebelum bulan tanggal itu. Berapa nomor hari dalam tahun untuk 2024-03-01 dan untuk 2023-12-31?
>
> *Jawaban.* `int n = day; for (int m = 1; m < month; m++) { n = n + daysInMonth(year, m); } return n;`. Untuk 2024-03-01: $31 + 29 + 1 = 61$. Untuk 2023-12-31: kesebelas bulan sebelum Desember berjumlah $31 + 28 + 31 + 30 + 31 + 30 + 31 + 31 + 30 + 31 + 30 = 334$ hari, dan $334 + 31 = 365$, seperti yang diharapkan pada tahun biasa.

4.17 Soal-Soal Berdampingan

Empat belas soal bisa saling mengabur. **Tabel 4.8** menjajarkan semuanya bersama satu gagasan yang paling baik diajarkan masing-masing dan jebakan yang dihindarinya. Bila Anda hanya mengingat kolom paling kanan tabel ini, Anda akan terhindar dari sebagian besar bug yang ditulis pemula pada tahun pertamanya.

Tabel 4.8 Keempat belas soal dalam bab ini, gagasan rancangan yang diajarkan masing-masing, dan jebakan yang diperagakan penyelesaian kelirunya.

Soal	Gagasan rancangan	Jebakan yang diperlihatkan penyelesaian keliru
4.1 Kalkulator pecahan	satu rumus per metode	pembagian bilangan bulat; menjumlahkan pembilang dan penyebut
4.2 Sudut jarum jam	normalisasi dulu, baru bulatkan	`%` negatif di Java; setengah ke atas lawan setengah ke genap
4.3 Akun komputer	*field* privat, satu objek per akun	`==` pada string
4.4 Bilangan kompleks	operasi mengembalikan objek baru	melupakan $i^2 = -1$
4.5 Berat	simpan sekali, turunkan sisanya	*field* kedua yang menjadi basi
4.6 Kelas pecahan	imutabilitas, perangkaian	mengubah `this` di dalam `getSum`
4.7 Pemain dan musuh	setiap objek menjaga *field*-nya sendiri	menulis *field* objek lain
4.8 Rupiah	uang yang tepat dalam `long`	mengelompokkan dari kiri; uang dalam `double`
4.9 Waktu dalam sehari	satu bilangan internal, tersembunyi	tanpa penyimpanan ke jam, tanpa putaran balik, tanpa pengisian nol
4.10 Nota laundry	komposisi, `switch`	memotong berat; *locale*
4.11 Merapikan nama	satu aturan privat dipakai tiga kali	`split(" ")` dan kata-kata kosong
4.12 Aturan kata sandi	hitung dalam perulangan, putuskan sesudahnya	mengembalikan nilai di dalam perulangan
4.13 Menelusuri perulangan	tabel jejak	`n > 9`; `/` alih-alih `//`
4.14 Tahun kabisat	aturan sebagai satu ekspresi Boolean	"habis dibagi 4" saja

Dua pola menonjol. Pertama, setiap kelas yang menyatakan sebuah *nilai* — pecahan, bilangan kompleks, jumlah uang, waktu, tanggal — dibuat *immutable*, dengan operasi-operasi yang mengembalikan objek baru. Kedua, hampir setiap jebakan tidak terlihat pada contoh-contoh yang "manis" dan baru tampak di tepi: menit yang ganjil, bilangan negatif, spasi kedua, tahun abad. Menguji tepi-tepi bukanlah tambahan; itulah *inti* pengujian.

4.18 Profitina dalam Praktik

Profitina (app.profitina.com) mengukur seberapa siap situs web sebuah merek untuk dikutip oleh mesin jawaban berbasis AI, dan membantu memperbaiki apa yang ditemukannya. Soal-soal dalam bab ini kecil, tetapi setiap gagasannya muncul dalam ukuran penuh di produk semacam itu.

Kelas-kelas kecil, masing-masing diuji tersendiri. Pemindai Profitina dibangun dari kelas-kelas sinyal: setiap sinyal — misalnya, apakah sebuah halaman membawa data terstruktur — adalah kelas kecil tersendiri dengan satu tugas, yang dinilai dan dijelaskan secara mandiri, persis seperti `Rupiah`, `TimeOfDay`, dan `PasswordChecker` masing-masing memiliki satu tugas di sini. Kelas-kelas kecil mudah diuji di tepi-tepinya, dan proses rilis Profitina menjadikannya aturan: seluruh rangkaian uji otomatis, jauh lebih dari seribu uji, harus hijau sebelum versi apa pun dirilis. Baris-baris `PASS`/`FAIL` pada driver-driver bab ini adalah gagasan yang sama dalam bentuk mini.

Uang dan bilangan yang tepat. Koin utilitas PROFIT milik Profitina dinilai terhadap rupiah, dan sebuah koin dengan enam angka desimal, seperti jumlah rupiah dalam Soal 4.8, paling baik ditangani sebagai bilangan bulat dari satuan terkecilnya, bukan sebagai nilai titik-mengambang.

Di mana pun sebuah jumlah uang ditampilkan kepada pengguna Indonesia, angkanya dikelompokkan dengan titik, seperti yang dilakukan `Rupiah.toString`.

Kata sandi tidak pernah dicetak. Soal 4.3 harus menyediakan `printPassword` karena soal memintanya. Produk sungguhan melakukan yang sebaliknya. Pengaturan akun swalayan Profitina, yang dibangun untuk menghormati undang-undang pelindungan data pribadi Indonesia (UU PDP 27/2022) [12], memungkinkan pengguna mengganti kata sandinya, tetapi kata sandi itu sendiri hanya disimpan sebagai *hash* Argon2id, tidak pernah ditampilkan, dan penggantian kata sandi mensyaratkan bukti kata sandi lama — *uji diri* Soal 4.3, yang ditanggapi dengan sungguh-sungguh.

Waktu dan tanggal. Dasbor historis Profitina menampilkan tren skor mingguan per merek dan memindai ulang secara otomatis menurut jadwal. “Tujuh hari setelah pemindaian ini”, “apakah pemindaian ini lebih lama daripada yang terakhir”, “tanggal ini jatuh di minggu yang mana”: itulah pertanyaan-pertanyaan yang dijawab `TimeOfDay` dan `SimpleDate` dalam bentuk mini, dan jebakan-jebakan Soal 4.9 dan 4.14 — putaran balik, akhir bulan, tahun kabisat — adalah jebakan yang harus ditangani dengan benar oleh setiap penjadwal.

Semua ini tetaplah nasihat: sebuah skor memberi tahu pemilik merek ke mana harus melihat, dan pemilik itulah yang memutuskan apa yang diubah. Latihan yang dapat dijalankan dalam buku ini tetap bersama Profitina Laundry, yang notanya pada Soal 4.10 memadukan, dalam empat puluh baris, uang dari Soal 4.8, percabangan dari Bab 3, dan gagasan satu objek yang memuat objek lain.

4.19 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya

Membagi dua bilangan bulat dan mengharapkan pecahan. Di Java, `1 / 2` bernilai `0`. Tulislah `1 / 2.0`, atau ubah salah satu operan menjadi `double`, setiap kali Anda menginginkan hasil pecahan; di Python, ingatlah bahwa `/` selalu menghasilkan bilangan titik-mengambang dan `//` adalah pembagian bilangan bulat (Soal 4.1, 4.2, 4.13).

Memercayai `%` dengan bilangan negatif di Java. `-105 % 360` bernilai `-105` di Java [7] dan `255` di Python [5]. Di Java, normalisasikan dengan `(x % m + m) % m` (Soal 4.2 dan 4.9).

Membandingkan string dengan `==`. Pakailah `equals` di Java; di Python, `==` membandingkan isi dan `is` membandingkan identitas (Soal 4.3).

Membulatkan dengan aturan yang salah. Tentukan aturan mana yang dimaksud soal: setengah ke atas (`Math.round`), setengah ke genap (`Math.rint`, `round` di Python), ke arah nol (*cast*), atau ke atas (`Math.ceil`). Aturan-aturan itu berbeda tepat pada kasus-kasus yang seharusnya dimuat dalam uji (Soal 4.2 dan 4.10).

Membandingkan nilai `double` dengan `==`. Hasil titik-mengambang dari perhitungan membawa galat kecil; bandingkan jaraknya dengan toleransi, atau pakailah bilangan bulat untuk uang (Soal 4.5 dan 4.8).

Menyimpan fakta yang sama dua kali. *Field* kedua yang dapat diturunkan dari *field* pertama suatu hari akan berselisih dengannya; hitunglah saja (Soal 4.5).

Mengubah `this` di dalam metode yang seharusnya mengembalikan hasil. Metode bernama `getSum` tidak boleh memodifikasi objeknya. Kembalikan objek baru (Soal 4.4 dan 4.6).

Merogoh ke dalam objek lain. Biarkan setiap objek hanya mengubah *field*-nya sendiri, dan biarkan pihak lain memintanya melalui metode (Soal 4.7).

Memutuskan di dalam perulangan apa yang baru dapat diputuskan sesudahnya. "Tidak ada karakter yang berupa angka" baru diketahui setelah karakter terakhir (Soal 4.12).

Kondisi perulangan yang meleset satu. Telusurilah lintasan pertama dan terakhir setiap perulangan di atas kertas; bila tabelnya kekurangan atau kelebihan satu baris, demikian pula perulangannya (Soal 4.13).

Melupakan `this.` di dalam konstruktor. `realName = realName;` memberikan parameter kepada dirinya sendiri dan membiarkan *field*-nya kosong (Soal 4.3).

Hanya menguji contoh-contoh dalam pernyataan soal. Tambahkan nol, bilangan negatif, batas-batas, string kosong, menit yang ganjil, dan tahun abad. Setiap penyelesaian keliru yang menggoda dalam bab ini lolos pada sekurang-kurangnya satu contoh yang "manis" [13].

4.20 Ringkasan Bab

Bab ini mengukuhkan Bab 1 sampai 3 dengan menyelesaikan soal, bukan dengan memperkenalkan fitur baru.

Sebuah metode. Subbab 4.2 menguraikan delapan langkah: nyatakan ulang soalnya; kerjakan contoh dengan tangan; temukan kata benda dan kata kerjanya; gambarlah kotak kelasnya; tulis ujinya lebih dulu; tulis kodenya sepotong demi sepotong; jalankan, bandingkan, dan debug; lalu tengok kembali untuk kasus tepi, penyelesaian keliru yang menggoda, dan varian. Driver uji Java sederhana dengan pembantu `check` dan `unittest` milik Python mengubah contoh-contoh tangan menjadi pemeriksaan otomatis; driver-driver itu pindah ke JUnit 6 dengan mengganti setiap `check` dengan `assertEquals`.

Tujuh soal klasik. Soal 4.1 menjumlahkan dan mengalikan pecahan dengan bilangan bulat, dan memperlihatkan pembagian bilangan bulat sebagai jebakan pertama. Soal 4.2 menghitung sudut jarum jam dengan Persamaan (4.3) sampai Persamaan (4.5), dan memperlihatkan bahwa "dibulatkan" di sana berarti pembulatan setengah ke genap, serta bahwa `%` di Java mempertahankan tanda operan kirinya. Soal 4.3 membangun objek sungguhan yang pertama, dengan *field* privat dan `equals`. Soal 4.4 membuat operasi-operasi mengembalikan objek baru. Soal 4.5 menyimpan satu fakta dan menurunkan yang lain. Soal 4.6 mengubah Soal 4.1 menjadi kelas *immutable* yang hasil-hasilnya dapat dirangkaikan. Soal 4.7 membuat dua kelas saling mengirim pesan dan membiarkan masing-masing menjaga *field*-nya sendiri.

Tujuh soal baru. Jumlah rupiah sebagai bilangan bulat yang tepat dengan angka-angka dikelompokkan dari kanan; waktu dalam sehari yang disimpan sebagai menit sejak tengah malam; nota laundry yang memuat total `Rupiah` dan menagih per kilogram yang dimulai; nama yang dirapikan karakter demi karakter; aturan kata sandi yang dihitung dalam satu perulangan dan diputuskan sesudahnya; perulangan yang ditelusuri dengan tangan dalam tabel, termasuk algoritma Euclid Persamaan (4.8); serta aturan tahun kabisat Gregorian Persamaan (4.9) dengan tanggal yang mengenal hari berikutnya.

Menatap ke depan. Bab 5 membawa larik dan *list* — sehingga sebuah nota dapat memuat berapa pun baris sebagai objek, bukan sebagai teks — bersama masukan dari papan ketik, eksepsi, dan berkas. `StringIndexOutOfBoundsException` dari Soal 4.11 kemudian akan menjadi sesuatu yang dapat Anda tangkap dan tangani, bukan hanya dihindari.

4.21 Soal Latihan

4.1 Perluas `Q16Fraction` dengan `getDifference(other)` dan `getQuotient(other)`, yang masing-masing mengembalikan pecahan baru. Apa yang dicetak `f2.getDifference(f1)` dan `f2.getQuotient(f1)` untuk $f_1 = \frac{1}{2}$ dan $f_2 = \frac{3}{7}$ (tanpa penyederhanaan)?

4.2 Dengan memakai Persamaan (4.3) sampai Persamaan (4.5), hitunglah dengan tangan sudut berlawanan arah jarum jam pada pukul 7:20, 12:30, dan 0:01.

4.3 Pada pukul berapa di antara 3:00 dan 3:59 kedua jarum jam berimpit tepat? Berikan jawabannya sampai detik terdekat, dan sebutkan apa yang dikembalikan `Q12Time.angle` pada pukul 3:16.

4.4 Tulislah metode `equalsFraction(Q16Fraction other)` yang mengembalikan `true` bila dua pecahan bernilai sama, seperti $\frac{1}{2}$ dan $\frac{2}{4}$, tanpa menyederhanakan keduanya dan tanpa memakai `double`.

4.5 Telusurilah `Rupiah.toString` untuk jumlah -1000: daftarlah karakter yang dibaca, hitungan, dan hasil setelah setiap lintasan, serta string yang dikembalikan.

4.6 Berikan tiga kasus uji untuk `TimeOfDay.plusMinutes` yang masing-masing menangkap satu bug yang berbeda dari ketiga bug dalam **Kode 4.44**, dan nyatakan hasil yang diharapkan untuk masing-masing.

4.7 Untuk pesanan ekspres Budi pada Soal 4.10, mesin cuci rusak dan 1.2 kg dari 4.5 kg dikembalikan tanpa dicuci. Hitunglah total baru untuk 3.3 kg `wash+iron`, ekspres.

4.8 Tulislah tabel jejak `digitSum(90210)`. Berapa lintasan yang dijalani perulangannya, dan mengapa angka nol di tengah tidak mengakhiri perulangan?

4.9 Dengan memakai Persamaan (4.9), tentukan untuk masing-masing dari 1600, 1700, 2028, dan 2400 apakah tahun itu kabisat. Lalu hitunglah dengan tangan berapa hari terbentang dari 2024-02-27 sampai 2024-03-02, dan periksa jawaban Anda dengan menghitung langkah-langkah `nextDay`.

4.10 Seorang teman sekelas menulis `isStrong` sebagai `return password.length() >= 8 && hasDigit(password);`, dengan memakai `hasDigit` dari **Kode 4.59**. Berikan satu kata sandi kuat yang ditolak versi ini dan satu kata sandi lemah yang diterimanya, menurut keempat aturan Soal 4.12.

4.11 Rancanglah (gambarlah kotak kelasnya, dan daftarlah uji-ujinya) kelas `LaundryMachine` dengan nama, kapasitas dalam kilogram, dan muatan saat ini. Kelas ini menyediakan `load(kg)`, yang menolak muatan yang akan melampaui kapasitas, `unload()`, dan `freeCapacity()`. Gagasan-gagasan rancangan mana dari bab ini yang Anda pakai?

4.12 Terapkan delapan langkah Subbab 4.2 pada soal berikut, dengan menuliskan setiap langkah: "Diberikan sebuah bilangan bulat detik, cetaklah sebagai jam, menit, dan detik dalam bentuk `H:MM:SS`."

4.22 Penyelesaian Soal Latihan

4.1 Selisih: $\frac{e_1 d_2 - d_1 e_2}{d_1 d_2}$, sehingga `getDifference` mengembalikan `new Q16Fraction(numerator * other.getDenominator() - denominator * other.getNumerator(), denominator *`

`other.getDenominator()`. Untuk $\frac{3}{7} - \frac{1}{2}$: $3 \cdot 2 - 7 \cdot 1 = -1$ dan $7 \cdot 2 = 14$, jadi `-1/14`. Hasil bagi: membagi dengan $\frac{e}{d}$ sama dengan mengalikan dengan $\frac{d}{e}$, sehingga `getQuotient` mengembalikan `new Q16Fraction(numerator * other.getDenominator(), denominator * other.getNumerator())`. Untuk $\frac{3}{7} \div \frac{1}{2} = \frac{3 \cdot 2}{7 \cdot 1}$ tercetak `6/7`. (Pembagian dengan pecahan yang pembilangnya 0 tidak terdefinisi; eksepsi di Bab 5 adalah cara yang tepat untuk menolaknya.)

4.2 7:20: $\theta_{\text{jam}} = 30 \cdot \left(7 + \frac{20}{60}\right) = 210 + 10 = 220$, $\theta_{\text{menit}} = 120$, selisih 100, hasil 100. 12:30: $30 \cdot (0 + 0.5) = 15$, jarum menit 180, selisih -165, modulo 360 menjadi 195. 0:01: $30 \cdot \frac{1}{60} = 0.5$, jarum menit 6, selisih -5.5, modulo 360 menjadi 354.5, yang dibulatkan setengah ke genap menjadi 354.

4.3 Kedua jarum berimpit bila $30 \cdot \left(3 + \frac{m}{60}\right) = 6m$, yaitu $90 + 0.5m = 6m$, sehingga $5.5m = 90$ dan $m = \frac{90}{5.5} = 16.\overline{36}$ menit, yaitu 16 menit dan $0.\overline{36} \cdot 60 \approx 21.8$ detik: sekitar 3:16:22. Pada pukul 3:16 selisihnya $90 + 8 - 96 = 2$, sehingga `angle(3, 16)` mengembalikan 2; pada pukul 3:17 selisihnya $90 + 8.5 - 102 = -3.5$, yang dinormalisasi menjadi 356.5 dan dibulatkan (setengah ke genap) menjadi 356. Kedua jarum saling melewati di antara kedua menit itu.

4.4 Dua pecahan $\frac{a}{b}$ dan $\frac{c}{d}$ dengan penyebut positif sama nilainya tepat ketika $ad = bc$ (kalikan kedua ruas $\frac{a}{b} = \frac{c}{d}$ dengan bd). Jadi `return numerator * other.getDenominator() == denominator * other.getNumerator();`. Untuk $\frac{1}{2}$ dan $\frac{2}{4}$: $1 \cdot 4 = 4 = 2 \cdot 2$, jadi `true`. Memakai `double` berisiko galat pembulatan, seperti pada Soal 4.5.

4.5 `digits` adalah `"1000"` (nilai mutlaknya). Membaca dari kanan: `0`, hitungan 1, hasil `0`; `0`, hitungan 2, `00`; `0`, hitungan 3, dan karena `i = 1 > 0` sebuah titik ditambahkan: `.000`; `1`, hitungan 4, `1.000`. Karena jumlahnya negatif, metode mengembalikan `-Rp1.000`.

4.6 (a) `new TimeOfDay(22, 30).plusMinutes(45)` harus memberi `23:15`; bug tanpa penyimpanan ke jam memberi `22:75`. (b) `new TimeOfDay(22, 30).plusMinutes(150)` harus memberi `01:00`; bug tanpa putaran balik memberi `25:0`. (c) `new TimeOfDay(8, 0).plusMinutes(5)` harus memberi `08:05`; bug tanpa pengisian nol memberi `8:5`. Satu uji per bug adalah yang ideal: ketika sebuah uji gagal, namanya memberi tahu Anda apa yang rusak.

4.7 [3.3] = 4 kg dengan harga Rp10.000 adalah 40000; ekspres menambahkan 20000; totalnya 60000, tercetak `Rp60.000`. (Sebelumnya, dengan 5 kg, totalnya Rp75.000.)

4.8 Lintasan-lintasannya: angka 0, total 0, $n = 9021$; angka 1, total 1, $n = 902$; angka 2, total 3, $n = 90$; angka 0, total 3, $n = 9$; angka 9, total 12, $n = 0$. Lima lintasan, jumlah 12. Kondisi perulangan menguji n, bukan angkanya: *angka* nol membiarkan n tetap positif, sehingga perulangan berlanjut; hanya $n = 0$ yang mengakhirinya.

4.9 1600: habis dibagi 400, kabisat. 1700: habis dibagi 100 tetapi tidak oleh 400, bukan kabisat. 2028: habis dibagi 4, tidak oleh 100, kabisat. 2400: habis dibagi 400, kabisat. Dari 2024-02-27: `nextDay` memberi 02-28, 02-29 (2024 tahun kabisat), 03-01, 03-02 — empat langkah, jadi empat hari.

4.10 Versi teman sekelas itu hanya memeriksa panjang dan sebuah angka, dan `hasDigit`-nya hanya melihat karakter pertama. Kata sandi kuat yang ditolaknya: `Laundry7` — angkanya berada di akhir, sehingga `hasDigit` mengembalikan `false`. Kata sandi lemah yang diterimanya:

`1sesamesesame` — cukup panjang dan diawali angka, tetapi tanpa huruf kapital, yang tidak pernah diperiksa versi itu.

4.11 Kotak kelas: *field* `- name : String`, `- capacityKg : double`, `- loadKg : double`; metode `+ LaundryMachine(name, capacityKg)`, `+ load(kg : double) : boolean`, `+ unload() : void`, `+ freeCapacity() : double`. `load` mengembalikan `false` dan tidak mengubah apa pun bila `kg <= 0` atau `loadKg + kg > capacityKg`; `unload` mengatur `loadKg` menjadi 0; `freeCapacity` mengembalikan `capacityKg - loadKg` — diturunkan, bukan disimpan (Soal 4.5). Uji: mesin 7 kg menerima 3 kg lalu 4 kg (tepat penuh, batas), menolak tambahan 0.1 kg, melaporkan kapasitas bebas 0, dan setelah `unload` melaporkan 7. Gagasan yang dipakai: *field* privat yang dijaga metode (Soal 4.3 dan 4.7), menolak masukan buruk sejak awal (Soal 4.10), simpan sekali dan turunkan (Soal 4.5), serta menguji di batas (Soal 4.12).

4.12 Langkah 1: masukannya bilangan bulat $s \geq 0$ dalam detik; keluarannya teks `H:MM:SS`. Langkah 2: $3725 = 1 \cdot 3600 + 2 \cdot 60 + 5$, jadi `1:02:05`; 59 memberi `0:00:59`; tepi 0 memberi `0:00:00`. Langkah 3: kata bendanya adalah sebuah *durasi* dengan jam, menit, detik; kata kerjanya *memformat*. Langkah 4: kelas `Duration` dengan satu *field* `- seconds : int` (satu bilangan internal, seperti pada Soal 4.9) dan metode `getHours()` $= s/3600$, `getMinutes()` $= (s \bmod 3600)/60$, `getSeconds()` $= s \bmod 60$, serta `toString()`. Langkah 5: uji untuk 3725, 59, 0, dan 86399 (`23:59:59`). Langkah 6: `toString` mengembalikan `String.format("%d:%02d:%02d", getHours(), getMinutes(), getSeconds())`. Langkah 7: jalankan ujinya; `%02d` yang terlupa langsung tampak sebagai `1:2:5`. Langkah 8: penyelesaian keliru yang menggoda menghitung menit sebagai `s / 60`, yang memberi 62 untuk 3725; sebuah varian akan mencetak hari juga untuk durasi di atas 24 jam.

Daftar Pustaka

[1] G. Pólya, *How to Solve It: A New Aspect of Mathematical Method*, 2nd ed. Princeton, NJ, USA: Princeton University Press, 1957.

[2] The JUnit Team, *JUnit User Guide*, version 6.1.3, Aug. 2026. Accessed: Sep. 24, 2026. [Online]. Available: https://docs.junit.org/current/

[3] Python Software Foundation, “unittest — Unit testing framework,” *The Python Standard Library*, Python 3.14 documentation, 2025. [Online]. Available: https://docs.python.org/3/library/unittest.html

[4] *IEEE Standard for Floating-Point Arithmetic*, IEEE Std 754-2019, 2019.

[5] Python Software Foundation, “Built-in Functions” and “Expressions,” *The Python Standard Library* and *The Python Language Reference*, Python 3.14 documentation, 2025. [Online]. Available: https://docs.python.org/3/

[6] Oracle, *Java Platform, Standard Edition 21 API Specification* (classes `java.lang.Math`, `java.util.Formatter`, `java.util.Locale`), 2023. [Online]. Available: https://docs.oracle.com/en/java/javase/21/docs/api/

[7] J. Gosling, B. Joy, G. Steele, G. Bracha, A. Buckley, D. Smith, and G. Bierman, *The Java Language Specification, Java SE 21 Edition*. Oracle, 2023, §15.17.3 (remainder operator) and §15.21.3 (reference equality).

[8] J. Bloch, *Effective Java*, 3rd ed. Boston, MA, USA: Addison-Wesley, 2018, Item 17: “Minimize mutability.”

[9] P. A. Grassi *et al.*, “Digital Identity Guidelines: Authentication and Lifecycle Management,” NIST Special Publication 800-63B, National Institute of Standards and Technology, Gaithersburg, MD, USA, 2017 (with updates).

[10] D. E. Knuth, *The Art of Computer Programming, Volume 2: Seminumerical Algorithms*, 3rd ed. Reading, MA, USA: Addison-Wesley, 1997, §4.5.2.

[11] E. M. Reingold and N. Dershowitz, *Calendrical Calculations: The Ultimate Edition*, 4th ed. Cambridge, UK: Cambridge University Press, 2018.

[12] Republik Indonesia, *Undang-Undang Nomor 27 Tahun 2022 tentang Pelindungan Data Pribadi*, 2022.

[13] C. S. Horstmann, *Core Java, Volume I: Fundamentals*, 13th ed. Hoboken, NJ, USA: Pearson, 2024.

Bab 5

Larik, List, Masukan, Eksepsi, dan Berkas

Pada akhir bab ini Anda akan sanggup menyimpan banyak nilai di bawah satu nama, baik dalam *array* (larik) yang ukurannya tetap maupun dalam *list* yang tumbuh dengan sendirinya; menggambar rupa sebuah larik di dalam memori dan menjelaskan mengapa indeks pertamanya 0 dan indeks terakhirnya satu kurang dari panjangnya; membaca apa yang diketikkan seseorang di papan ketik, di Java dengan `Scanner` dan di Python dengan `input()`; membaca *stack trace* dan *traceback* dengan tenang, baris demi baris, lalu menemukan baris yang gagal; menangkap sebuah *exception* (eksepsi), membiarkannya berjalan menuju pemanggil yang tahu apa yang harus dilakukan terhadapnya, membereskan sumber daya dengan `finally`, dan melemparkan eksepsi buatan Anda sendiri; membedakan eksepsi terperiksa dari eksepsi tak terperiksa serta menjelaskan mengapa Java membuat pembedaan itu sedangkan Python tidak; menulis kelas eksepsi Anda sendiri; serta membaca dan menulis berkas teks, termasuk berkas CSV, sambil menutup setiap berkas yang Anda buka sekalipun terjadi kesalahan. Bab ini ditutup dengan Profitina Laundry yang memuat pesanan-pesanannya dari sebuah berkas CSV, menolak setiap baris yang buruk disertai penjelasan yang tepat, lalu menyimpan pesanan-pesanan yang baik kembali ke disk, di Java dan di Python.

5.1 Di Mana Kita Berhenti

Sampai sejauh ini setiap objek yang kita buat hidup sendirian. Sebuah variabel menyimpan satu bilangan, satu string, satu lingkaran, satu pesanan cucian. Bab 3 memberi kita kelas, objek, string, tipe, percabangan, dan perulangan, dan Bab 4 mempekerjakan semua itu pada serangkaian soal yang dibahas tuntas. Program-program itu jujur dan lengkap, tetapi semuanya berbagi satu anggapan diam-diam: bahwa kita tahu sejak awal berapa persisnya banyak benda yang akan kita tangani, dan bahwa tidak akan pernah ada yang salah.

Binatu yang sesungguhnya tidak bekerja seperti itu. Pada suatu Senin pagi Profitina Laundry mungkin menerima empat pesanan, mungkin juga empat puluh. Seorang pelanggan mungkin mengetik “tiga” di tempat program mengharapkan bilangan 3. Berkas yang menyimpan pesanan minggu lalu mungkin telah dipindahkan, diganti namanya, atau disunting dengan tangan oleh seseorang yang membiarkan satu isian kosong. Program yang menyambut salah satu keadaan itu dengan setumpuk teks merah lalu berhenti mendadak belumlah selesai; ia baru sebuah purwarupa.

Bab ini menambal tiga celah sekaligus, dan ketiganya lebih erat bertalian daripada yang mungkin terlihat pada pandangan pertama.

Celah pertama adalah **banyak nilai**. Kita mempelajari dua wadah: *array* (larik), yang tertua dan paling sederhana, yaitu sederet slot bernomor yang ukurannya tetap; dan *list* (`ArrayList` di Java, `list` bawaan di Python), yang tumbuh ketika kita menambahkan isi ke dalamnya.

Celah kedua adalah **masukan**. Program yang hanya mencetak adalah pidato, bukan percakapan. Kita belajar membaca apa yang diketikkan seseorang, mula-mula dari papan ketik, kemudian dari sebuah berkas.

Celah ketiga adalah **kegagalan**. Begitu sebuah program membaca masukan, ia tidak lagi menguasai datanya sendiri, dan pasti akan ada yang salah. Java dan Python sama-sama menjawabnya dengan *exception* (eksepsi): objek yang menguraikan apa yang salah, yang berjalan dari tempat masalah itu terdeteksi menuju tempat seseorang dapat berbuat sesuatu terhadapnya. Eksepsi dan berkas diajarkan bersama di sini dengan alasan yang baik: hampir segala sesuatu yang menyentuh berkas dapat gagal, dan Java bahkan tidak mau mengompilasi kode pembaca berkas sebelum Anda menyatakan apa yang harus terjadi bila kegagalan itu datang.

Kita akan bekerja mengikuti urutan ketergantungan gagasan-gagasannya: larik, lalu list, lalu masukan, lalu eksepsi, lalu berkas. Setiap contoh adalah program yang lengkap, pertama dalam Java lalu dalam Python, beserta keluarannya yang sebenarnya.

5.2 Banyak Nilai di Bawah Satu Nama: Larik

5.2.1 Rak kotak surat

Sebenarnya kita sudah pernah bertemu larik, meskipun belum mengamatinya dengan saksama. Setiap program Java yang telah kita tulis dimulai seperti ini:

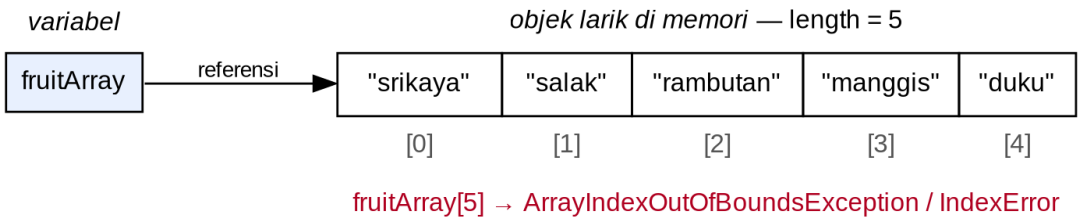
```
public class Hello {
    public static void main(String[] args) {
        ...
```

`String[] args` dalam deklarasi `main` adalah sebuah larik. Kurung siku sesudah `String` berarti "bukan satu `String`, melainkan sederet `String`".

Gambaran yang baik untuk sebuah larik adalah rak kotak surat di ruang pos sebuah kantor: satu perabot dengan cacah slot yang tetap, setiap slot diberi label nomor, dan setiap slot dapat menampung satu barang. Anda menentukan berapa slot yang dimiliki rak itu ketika Anda membuatnya, dan sesudah itu cacahnya tidak pernah berubah. Anda dapat menaruh sesuatu di slot 3, melihat apa yang ada di slot 0, atau mengganti isi slot 4, tetapi Anda tidak dapat menambahkan slot keenam pada rak yang dibuat dengan lima slot. Gagasannya adalah bahwa kita membuat larik dengan sudah mengetahui sebelumnya seberapa besar larik itu perlu.

> **DEFINISI.** *Array* (larik) adalah barisan slot yang panjangnya tetap, yang disebut *elemen*, semuanya bertipe sama, tersimpan berurutan satu sesudah yang lain, dan masing-masing dikenali dengan sebuah *indeks* bilangan bulat. Di Java dan di Python indeks pertamanya adalah 0, sehingga larik sepanjang n memiliki indeks $0,1, \ldots, n-1$.

Gambar 5.1 memperlihatkan sebuah larik berisi lima string sebagaimana komputer menyimpannya. Di sebelah kiri terdapat variabel `fruitArray`. Variabel itu tidak memuat kelima buah itu sendiri; ia memuat sebuah *reference* (referensi), yaitu alamat tempat di memori tempat objek larik itu berada. Objek lariknya adalah deretan lima slot di sebelah kanan, beserta catatan tentang panjangnya. Di bawah setiap slot tertera indeksnya.



Gambar 5.1 Sebuah larik berisi lima string di dalam memori. Variabel menyimpan referensi ke objek larik; slot-slotnya bernomor 0 sampai 4, dan indeks 5 tidak ada.

Mengapa mulai dari 0 dan bukan dari 1? Karena indeks sesungguhnya adalah sebuah jarak. Slot-slot larik terletak berdampingan di memori, masing-masing memakan cacah bita yang sama. Jika larik dimulai pada alamat memori b dan setiap slot selebar s bita, maka slot i dimulai pada

$$\text{alamat}(i) = b + i \cdot s \tag{5.1}$$

Slot pertama berjarak nol dari awal, sehingga indeksnya 0. Persamaan (5.1) juga menjelaskan mengapa pembacaan slot mana pun dari sebuah larik memakan waktu singkat yang sama, berapa pun panjang lariknya: komputer melakukan satu perkalian dan satu penjumlahan lalu langsung menuju slot itu, tanpa berjalan melewati slot-slot sebelumnya. Sifat ini, yang disebut *random access* (akses acak), adalah keunggulan besar larik.

Karena itu indeks terakhir adalah $n - 1$, satu kurang dari panjangnya. Jika Anda hanya mengingat satu kalimat dari subbab ini, ingatlah kalimat ini: *larik sepanjang n tidak memiliki elemen berindeks n.*

5.2.2 Membuat, mengisi, dan menelusuri larik

Misalkan pemilik Profitina Laundry menggemari tepat lima jenis buah dan ingin nama-namanya tersimpan dalam sebuah program. Di Java, kita membuat larik dengan lima slot `String` lalu mengisinya satu slot demi satu slot. **Kode 5.1** melakukan hal itu, mencetak lariknya, kemudian menelusurinya dengan perulangan `for`.

Kode 5.1 `FruitArray.java`: membuat, mengisi, dan menelusuri larik Java, ditambah bentuk inisialisasi singkat.

```java
import java.util.Arrays;

public class FruitArray {
    public static void main(String[] args) {
        String[] fruitArray = new String[5];      // lima slot, indeks 0 sampai 4
        System.out.println("A fresh array: " + Arrays.toString(fruitArray));

        fruitArray[0] = "srikaya";                // kita mulai menomori dari nol
        fruitArray[1] = "salak";
        fruitArray[2] = "rambutan";
        fruitArray[3] = "manggis";
        fruitArray[4] = "duku";

        System.out.println("The element at position 0 is " + fruitArray[0]);
        for (int fruitIndex = 0; fruitIndex < fruitArray.length; fruitIndex++) {
            System.out.println("Fruit " + fruitIndex + " is " + fruitArray[fruitIndex]);
        }

        // Bentuk singkat: ukurannya tersirat dari banyaknya nilai di dalam kurung kurawal.
        String[] secondFruitArray = {"strawberry", "raspberry", "blackcurrant"};
        System.out.println("The shorthand array has length " + secondFruitArray.length);
        for (String fruit : secondFruitArray) {
            System.out.println("I like " + fruit);
        }

        int[] tenInts = new int[10];              // bilangan bernilai awal 0
        System.out.println("Ten ints: " + Arrays.toString(tenInts));
    }
}
```

```
A fresh array: [null, null, null, null, null]
The element at position 0 is srikaya
Fruit 0 is srikaya
Fruit 1 is salak
Fruit 2 is rambutan
Fruit 3 is manggis
Fruit 4 is duku
The shorthand array has length 3
I like strawberry
I like raspberry
I like blackcurrant
Ten ints: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

Mari kita baca kodenya baris demi baris.

- `String[] fruitArray = new String[5];` melakukan dua hal. Ruas kiri mendeklarasikan variabel yang tipenya "larik `String`". Ruas kanan, `new String[5]`, membangun objek larik baru dengan lima slot di memori dan menyerahkan referensi kepadanya. Kita menaruh string `"srikaya"` di posisi pertama, yang bernomor nol.
- Larik yang baru dibuat tidak pernah benar-benar kosong: setiap slot memuat sebuah *default value* (nilai bawaan). Untuk larik berisi objek seperti string, nilai bawaannya adalah `null`, yang berarti "belum ada objek di sini"; untuk larik `int` nilai bawaannya `0`; untuk `boolean` nilainya `false`. Baris pertama keluaran menampilkan lima `null`, dan baris terakhir menampilkan sepuluh angka nol. `Arrays.toString` adalah metode pembantu dari `java.util.Arrays` yang mengubah larik menjadi teks yang mudah dibaca; mencetak larik secara langsung hanya akan menampilkan kode yang tidak berguna seperti `[Ljava.lang.String;@1b6d3586`.
- `fruitArray[0] = "srikaya";` menaruh sebuah nilai di slot 0. Bilangan di dalam kurung siku menyatakan posisi mana yang ingin kita isi atau baca. Perhatikan bahwa kita tidak mungkin sekadar *menambahkan* sesuatu ke dalam larik; kita selalu harus menyebutkan di mana benda itu diletakkan.

- `fruitArray.length` adalah cacah slot. Perhatikan bahwa tidak ada tanda kurung: `length` adalah sebuah *field* milik larik, bukan metode. (String, yang agak membingungkan, memakai `length()` dengan tanda kurung. Kompilator akan memberi tahu Anda jika keduanya tertukar.)
- Perulangan `for (int fruitIndex = 0; fruitIndex < fruitArray.length; fruitIndex++)` mengunjungi setiap indeks dari 0 sampai panjangnya, tetapi tidak termasuk panjangnya itu sendiri. Kita bisa saja menulis `fruitIndex < 5`, dan untuk larik ini hasilnya benar, tetapi belum tentu selalu begitu: jika kelak lariknya diperbesar, angka 5 yang ditulis mati akan diam-diam melewatkan slot-slot baru. Kita hendaknya selalu memakai `.length` ketika menelusuri larik.
- `String[] secondFruitArray = {"strawberry", "raspberry", "blackcurrant"};` adalah bentuk *singkat*. Bila kita sudah tahu isinya pada saat larik dibuat, kita menuliskannya di dalam kurung kurawal dan kompilator yang menghitungnya: larik ini panjangnya 3. Kedua cara itu setara; menulis `new String[3]` lalu tiga penugasan akan menghasilkan larik yang persis sama. Ini jalan pintas yang berguna, tetapi pastikan Anda memahami bentuk panjangnya lebih dahulu.
- `for (String fruit : secondFruitArray)` adalah *enhanced for loop* (perulangan `for` yang disempurnakan), yang dibaca "untuk setiap `fruit` di dalam `secondFruitArray`". Perulangan ini mengunjungi elemen-elemen secara berurutan tanpa menyebut indeks sama sekali. Pakailah setiap kali Anda memerlukan nilai-nilainya tetapi tidak memerlukan posisinya.
- `int[] tenInts = new int[10];` menjawab pertanyaan yang wajar: larik dapat menampung tipe apa pun, bukan hanya string. Di sini ada sepuluh `int`. Larik juga dapat menampung `double`, `boolean`, `char`, objek dari kelas buatan Anda sendiri, bahkan larik yang lain.

Python tidak mempunyai larik berukuran tetap yang tertanam dalam bahasanya. Barisan sehari-harinya adalah `list`, yang kita pelajari dengan sungguh-sungguh di Subbab 5.3. Untuk meniru rak dengan lima slot, kita membuat list berisi lima `None`; `None` memerankan `null` versi Java. **Kode 5.2** adalah program yang sama.

Kode 5.2 `fruit_array.py`: program yang sama dengan list Python yang panjangnya tetap.

```
fruit_array = [None] * 5              # lima slot, indeks 0 sampai 4
print("A fresh list:", fruit_array)

fruit_array[0] = "srikaya"            # kita mulai menomori dari nol
fruit_array[1] = "salak"
fruit_array[2] = "rambutan"
fruit_array[3] = "manggis"
fruit_array[4] = "duku"

print("The element at position 0 is " + fruit_array[0])
for fruit_index in range(len(fruit_array)):
    print("Fruit " + str(fruit_index) + " is " + fruit_array[fruit_index])

# Bentuk singkat: ukurannya tersirat dari banyaknya nilai di dalam kurung siku.
second_fruit_array = ["strawberry", "raspberry", "blackcurrant"]
print("The shorthand list has length", len(second_fruit_array))
for fruit in second_fruit_array:
    print("I like " + fruit)

ten_ints = [0] * 10                   # bilangan tidak otomatis bernilai 0 kecuali kita
memintanya
print("Ten ints:", ten_ints)
```

```
A fresh list: [None, None, None, None, None]
The element at position 0 is srikaya
Fruit 0 is srikaya
Fruit 1 is salak
Fruit 2 is rambutan
Fruit 3 is manggis
Fruit 4 is duku
The shorthand list has length 3
I like strawberry
I like raspberry
I like blackcurrant
Ten ints: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

Perbedaannya kecil, tetapi layak disebutkan. `[None] * 5` membangun list berisi lima `None` dengan mengulang list satu elemen sebanyak lima kali. `len(fruit_array)` memerankan `fruitArray.length`. Perulangan `for fruit_index in range(len(fruit_array))` menghitung dari 0 sampai panjangnya, tidak termasuk panjangnya, persis seperti perulangan Java; `range(n)` menghasilkan $0,1,\ldots,n-1$. Bentuk singkatnya hanyalah list yang ditulis di dalam kurung siku. `for fruit in second_fruit_array` di Python adalah satu-satunya jenis perulangan `for` yang dimilikinya, dan gagasannya sama dengan `for` yang disempurnakan di Java. Terakhir, Python tidak mengisi list baru dengan nol atas prakarsanya sendiri: `[0] * 10` menyatakannya secara eksplisit.

> **WAWASAN KUNCI:** Di kedua bahasa, perulangan yang menelusuri larik berbentuk sama: mulai dari 0, berhenti *sebelum* panjangnya. Menulis `<` alih-alih `<=` dalam syarat itu adalah kebiasaan terpenting di subbab ini.

(Python juga memiliki modul `array` untuk larik bilangan yang ringkas, dan pustaka ilmiah NumPy mempunyai larik berukuran tetap yang sangat andal. Untuk program sehari-hari, list-lah yang dipakai para pemrogram Python.)

5.2.3 Melangkah melewati ujung

Apa yang akan terjadi jika kita menulis `fruitArray[5] = "durian";`? Larik itu mempunyai slot 0 sampai 4. Slot 5 tidak ada. **Kode 5.3** sengaja membuat kesalahan ini, dengan cara yang paling sering terjadi dalam praktik: syarat perulangan yang ditulis dengan `<=` alih-alih `<`.

Kode 5.3 `ArrayBounds.java`: perulangan yang berjalan satu langkah terlalu jauh dan menulis melewati ujung larik berisi lima elemen.

```java
import java.util.Arrays;

public class ArrayBounds {
    public static void main(String[] args) {
        String[] words = {"one", "two", "three", "four", "five", "six"};
        String[] stringArray = new String[5];     // hanya lima elemen...
        for (int i = 0; i <= 5; i++) {            // ...tetapi i berjalan sampai 5
            stringArray[i] = words[i];
            System.out.println("Added " + words[i] + " at index " + i);
        }
        System.out.println("Done: " + Arrays.toString(stringArray));
    }
}
```

```
Added one at index 0
Added two at index 1
Added three at index 2
Added four at index 3
Added five at index 4
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index 5 out of bounds
for length 5
    at ArrayBounds.main(ArrayBounds.java:8)
```

Perulangan itu berjalan lancar untuk `i` bernilai 0, 1, 2, 3, dan 4. Ketika `i` menjadi 5, program mencoba menyimpan `"six"` di slot yang tidak ada, dan Java menolaknya. Java tidak diam-diam memperbesar lariknya, dan tidak pula mencoret-coret memori apa pun yang kebetulan terletak sesudah larik itu (hal yang bisa saja dilakukan program C, dengan akibat yang fatal). Java berhenti dengan sebuah *exception* (eksepsi) bernama `ArrayIndexOutOfBoundsException`, dan pesannya menyatakan dengan tepat apa yang salah: "Index 5 out of bounds for length 5" (indeks 5 di luar batas untuk panjang 5). Baris di bawahnya, `at ArrayBounds.main(ArrayBounds.java:8)`, menyebut metode dan nomor barisnya: baris 8 adalah `stringArray[i] = words[i];`. Perhatikan juga bahwa `Done:` tidak pernah tercetak. Begitu eksepsi dilemparkan, sisa `main` ditinggalkan.

Di Subbab 5.5 kita akan mempelajari dengan persis apa itu eksepsi dan bagaimana menangkapnya. Untuk saat ini, perhatikan saja bentuk pesannya dan kenyataan bahwa pesan itu menunjuk ke satu baris yang tepat.

Kode 5.4 membuat kesalahan yang sama di Python.

Kode 5.4 `array_bounds.py`: kesalahan selisih-satu yang sama di Python.

```python
words = ["one", "two", "three", "four", "five", "six"]
string_array = [None] * 5             # hanya lima elemen...
for i in range(6):                    # ...tetapi i berjalan sampai 5
    string_array[i] = words[i]
    print("Added " + words[i] + " at index " + str(i))
print("Done:", string_array)
```

```
Added one at index 0
Added two at index 1
Added three at index 2
Added four at index 3
Added five at index 4
Traceback (most recent call last):
  File "/home/claude/kit/OOP/Code_new/Chapter05/python/array_bounds.py", line 4, in
<module>
    string_array[i] = words[i]
    ~~~~~~~~~~~~~~^^^
IndexError: list assignment index out of range
```

Python menamai masalah ini `IndexError`, dan laporannya, yang disebut *traceback*, memberikan nama berkas, nomor baris (4), baris itu sendiri, serta sederet tanda `~` dan `^` yang menunjuk bagian baris yang gagal. Jalur (*path*) pada baris `File` adalah tempat program itu berada ketika kami menjalankannya; di komputer Anda jalurnya akan berupa folder Anda sendiri.

> **JEBAKAN:** Python menerima indeks *negatif*: `fruit_array[-1]` adalah elemen terakhir dan `fruit_array[-5]` elemen pertama. Itu memudahkan, tetapi artinya indeks yang keliru terhitung menjadi `-1` tidak gagal di Python; ia diam-diam membaca elemen terakhir. Java tidak mengenal indeks negatif, dan `fruitArray[-1]` melempar `ArrayIndexOutOfBoundsException`.

5.2.4 Larik bilangan, dan larik berisi larik

Larik bersinar ketika sebuah program harus melakukan hal yang sama terhadap banyak bilangan. Mesin pertama Profitina Laundry mencuci lima muatan pagi ini, dan pemiliknya ingin mengetahui berat totalnya, muatan terberatnya, dan rata-ratanya. **Kode 5.5** menghitung ketiganya dalam satu kali penelusuran, lalu memakai larik *dua dimensi* untuk menjumlahkan banyaknya muatan yang dicuci oleh masing-masing dari tiga mesin selama tiga hari.

Kode 5.5 `LoadStats.java`: total, maksimum, dan rata-rata sebuah `double[]`, serta jumlah per baris sebuah `int[][]`.

```
public class LoadStats {
    public static void main(String[] args) {
        double[] loadsKg = {4.5, 6.0, 3.2, 7.8, 5.5};

        double total = 0;
        double heaviest = loadsKg[0];
        for (double load : loadsKg) {             // for yang disempurnakan: setiap nilai
bergiliran
            total += load;
            if (load > heaviest) {
                heaviest = load;
            }
        }
        System.out.printf("%d loads, total %.1f kg, heaviest %.1f kg, average %.2f kg%n",
                loadsKg.length, total, heaviest, total / loadsKg.length);

        // Larik dua dimensi: satu baris per mesin, satu kolom per hari.
        int[][] washes = {
            {12, 15, 9},
            {8, 11, 14},
            {10, 10, 13}
        };
        for (int m = 0; m < washes.length; m++) {
            int sum = 0;
            for (int d = 0; d < washes[m].length; d++) {
                sum += washes[m][d];
            }
            System.out.println("Machine " + (m + 1) + " washed " + sum + " loads");
        }
        System.out.println("Machine 2 on day 3: " + washes[1][2]);
    }
}
```

```
5 loads, total 27.0 kg, heaviest 7.8 kg, average 5.40 kg
Machine 1 washed 36 loads
Machine 2 washed 33 loads
Machine 3 washed 33 loads
Machine 2 on day 3: 14
```

Paruh pertama program ini adalah pola yang akan Anda tulis ratusan kali. Kita menyimpan `total` yang berjalan, dimulai dari 0, dan `heaviest` yang berjalan, dimulai dari elemen pertama (bukan dari 0, sebab list berisi bilangan negatif akan melaporkan 0 sebagai maksimumnya). `for` yang disempurnakan menyerahkan setiap `load` secara bergiliran. Di akhir, rata-ratanya adalah total dibagi cacahnya: $27.0 / 5 = 5.40$.

Paruh kedua memperkenalkan larik yang elemen-elemennya sendiri adalah larik. `int[][] washes` adalah larik berisi tiga baris; setiap baris adalah `int[]` dengan tiga kolom. `washes.length` adalah cacah baris (3), dan `washes[m].length` adalah cacah kolom pada baris `m` (juga 3). Elemen pada baris `m` dan kolom `d` adalah `washes[m][d]`. Karena kedua indeks dimulai dari 0, "mesin 2 pada hari 3" adalah `washes[1][2]`, yaitu 14. Larik dua dimensi mudah dibayangkan sebagai tabel,

dan kemudahan itulah salah satu alasan larik tetap populer untuk kisi, papan permainan, dan citra.

Kode 5.6 adalah perhitungan yang sama di Python, tempat larik dua dimensi cukup berupa list berisi list.

Kode 5.6 `load_stats.py`: statistik yang sama dengan list Python dan list berisi list.

```
loads_kg = [4.5, 6.0, 3.2, 7.8, 5.5]

total = 0
heaviest = loads_kg[0]
for load in loads_kg:                 # setiap nilai bergiliran
    total += load
    if load > heaviest:
        heaviest = load
print(f"{len(loads_kg)} loads, total {total:.1f} kg, heaviest {heaviest:.1f} kg, "
      f"average {total / len(loads_kg):.2f} kg")

# List berisi list: satu baris per mesin, satu kolom per hari.
washes = [
    [12, 15, 9],
    [8, 11, 14],
    [10, 10, 13],
]
for m in range(len(washes)):
    total_m = 0
    for d in range(len(washes[m])):
        total_m += washes[m][d]
    print("Machine " + str(m + 1) + " washed " + str(total_m) + " loads")
print("Machine 2 on day 3:", washes[1][2])
```

```
5 loads, total 27.0 kg, heaviest 7.8 kg, average 5.40 kg
Machine 1 washed 36 loads
Machine 2 washed 33 loads
Machine 3 washed 33 loads
Machine 2 on day 3: 14
```

f-string Python `f"{total:.1f}"` mengerjakan tugas `printf("%.1f", total)` di Java: satu angka di belakang titik desimal. Kita menamai jumlah per baris itu `total_m` dan bukan `sum`, sebab `sum` adalah nama fungsi bawaan Python, dan menyembunyikannya di balik sebuah variabel adalah sumber klasik kesalahan yang membingungkan di kemudian hari.

5.2.5 Variabel larik menyimpan referensi

Lihat kembali **Gambar 5.1**. Variabel menyimpan sebuah *referensi*, sebuah anak panah menuju larik, bukan slot-slotnya itu sendiri. Hal ini membawa akibat yang mengejutkan hampir setiap pemula: menugaskan satu variabel larik ke variabel lain menyalin anak panahnya, bukan lariknya. **Kode 5.7** memperagakannya, kemudian menunjukkan tiga cara yang benar untuk membuat salinan yang sungguhan.

Kode 5.7 `ArrayCopy.java`: *aliasing*, penyalinan dengan `Arrays.copyOf` dan `System.arraycopy`, serta pembandingan dengan `==` dan `Arrays.equals`.

```
import java.util.Arrays;

public class ArrayCopy {
    public static void main(String[] args) {
        int[] original = {1, 2, 3};
        int[] alias = original;                    // menyalin referensinya, bukan slotnya
        alias[0] = 99;
        System.out.println("original after alias[0] = 99: " + Arrays.toString(original));

        int[] copy = Arrays.copyOf(original, original.length);
        copy[1] = 42;
        System.out.println("copy:     " + Arrays.toString(copy));
        System.out.println("original: " + Arrays.toString(original));

        int[] bigger = Arrays.copyOf(original, 6);    // "membesar" berarti larik baru
        System.out.println("bigger:   " + Arrays.toString(bigger));

        int[] manual = new int[5];
        System.arraycopy(original, 0, manual, 1, 3);  // dari, awal, ke, awal, jumlah
        System.out.println("manual:   " + Arrays.toString(manual));

        int[] twin = {99, 2, 3};
        System.out.println("original == twin? " + (original == twin));
        System.out.println("Arrays.equals(original, twin)? " + Arrays.equals(original,
twin));
    }
}
```

```
original after alias[0] = 99: [99, 2, 3]
copy:     [99, 42, 3]
original: [99, 2, 3]
bigger:   [99, 2, 3, 0, 0, 0]
manual:   [0, 99, 2, 3, 0]
original == twin? false
Arrays.equals(original, twin)? true
```

Sesudah `int[] alias = original;` tetap hanya ada satu larik, dengan dua nama. Mengubah `alias[0]` mengubah apa yang ditampilkan `original[0]`, sebab keduanya adalah slot yang sama. Dua variabel yang merujuk ke objek yang sama disebut *alias*.

`Arrays.copyOf(original, original.length)` membangun larik baru dan menyalin nilai-nilainya ke sana; sesudahnya `copy` dan `original` saling bebas, sehingga mengubah `copy[1]` tidak menyentuh `original`.

Ini juga menjawab pertanyaan yang muncul dari ukuran yang tetap: jika larik tidak dapat tumbuh, bagaimana kita memperbesarnya? Jawabannya, kita tidak memperbesarnya; kita membuat larik baru yang lebih besar dan menyalin isi yang lama ke dalamnya. `Arrays.copyOf(original, 6)` melakukannya dalam satu panggilan dan mengisi slot-slot tambahannya dengan nol. `System.arraycopy(source, sourceStart, target, targetStart, count)` adalah alat tingkat rendah yang dipakai oleh `copyOf`: di sini ia menyalin tiga nilai ke `manual` mulai indeks 1. Cara itu berhasil, tetapi bertele-tele, dan Subbab 5.3 akan menunjukkan sebuah kelas yang mengurus pembukuan semacam ini untuk kita.

Dua baris terakhir membandingkan larik. `original == twin` bertanya "apakah keduanya objek yang sama?", dan jawabannya `false`, walaupun keduanya menyimpan bilangan-bilangan yang sama. `Arrays.equals(original, twin)` bertanya "apakah keduanya menyimpan nilai-nilai yang sama dengan urutan yang sama?", dan jawabannya `true`. Ini pembedaan yang sama dengan `==` lawan `equals` pada string, dan alasannya pun sama.

Kode 5.8 menuturkan kisah yang sama di Python.

Kode 5.8 `array_copy.py`: *aliasing* dan penyalinan list Python, serta `is` lawan `==`.

```
original = [1, 2, 3]
alias = original                      # menyalin referensinya, bukan slotnya
alias[0] = 99
print("original after alias[0] = 99:", original)

copy = original.copy()                # atau list(original), atau original[:]
copy[1] = 42
print("copy:    ", copy)
print("original:", original)

bigger = original + [0] * 3           # list baru yang lebih panjang
print("bigger:  ", bigger)

manual = [0] * 5
manual[1:4] = original                # penugasan irisan menyalin tiga nilai
print("manual:  ", manual)

twin = [99, 2, 3]
print("original is twin?", original is twin)
print("original == twin?", original == twin)
```

```
original after alias[0] = 99: [99, 2, 3]
copy:     [99, 42, 3]
original: [99, 2, 3]
bigger:   [99, 2, 3, 0, 0, 0]
manual:   [0, 99, 2, 3, 0]
original is twin? False
original == twin? True
```

Penugasan di Python juga hanya menyalin referensinya, sehingga `alias` dan `original` adalah list yang sama. `original.copy()`, `list(original)`, dan irisan `original[:]` semuanya membuat list baru. `original + [0] * 3` membangun list baru yang lebih panjang. Penugasan irisan, `manual[1:4] = original`, mengganti posisi 1, 2, dan 3 dengan ketiga nilai itu. Dan operator Python berkebalikan dengan Java: `is` bertanya "objek yang sama?" sedangkan `==` membandingkan isinya. Jadi di Python `original == twin` bernilai `True`.

5.2.6 Larik di dalam `main`: argumen baris perintah

Sekarang kita sudah cukup tahu untuk menjawab pertanyaan yang menjadi titik awal kita: untuk apa `String[] args` di dalam `main`? Ketika Anda menjalankan program Java dari terminal, kata-kata apa pun yang Anda ketikkan sesudah nama kelasnya dikumpulkan ke dalam larik itu. Jika kita mengetik

```
java Hello2 one two three
```

maka `args` menyimpan tiga string `"one"`, `"two"`, dan `"three"`. **Kode 5.9** mencetaknya.

Kode 5.9 `Hello2.java`: mencetak argumen baris perintah.

```
public class Hello2 {
    public static void main(String[] args) {
        System.out.println("I received " + args.length + " argument(s).");
        for (int i = 0; i < args.length; i++) {
            System.out.println("args[" + i + "] = " + args[i]);
        }
    }
}
```

```
I received 3 argument(s).
args[0] = one
args[1] = two
args[2] = three
```

Argumen baris perintah penting setiap kali sebuah program memerlukan informasi yang berubah dari satu kali jalan ke kali jalan berikutnya, bahkan sesudah dikompilasi: nama berkas yang akan dibuka, sebuah tanggal, sebuah mode. Kita akan memakainya dalam soal latihan untuk menyerahkan nama berkas kepada program. Di Python, kata-kata yang sama tiba di dalam list `sys.argv`, dengan satu perbedaan yang ditunjukkan **Kode 5.10**: `sys.argv[0]` adalah nama skrip itu sendiri, sehingga argumen yang sebenarnya dimulai pada indeks 1. Kita menjalankannya sebagai `python3 hello2.py one two three`.

Kode 5.10 `hello2.py`: mencetak argumen baris perintah dari `sys.argv`.

```
import sys

print("I received " + str(len(sys.argv) - 1) + " argument(s).")
for i in range(1, len(sys.argv)):     # sys.argv[0] adalah nama skrip itu sendiri
    print("sys.argv[" + str(i) + "] = " + sys.argv[i])
```

```
I received 3 argument(s).
sys.argv[1] = one
sys.argv[2] = two
sys.argv[3] = three
```

Di VS Code Anda dapat memberikan argumen tanpa terminal dengan menambahkan entri `"args"` pada konfigurasi jalan di `launch.json`, seperti yang ditunjukkan Bab 2 untuk *debugger*.

5.3 List yang Tumbuh: `ArrayList` dan `list` Python

5.3.1 Mengapa ukuran tetap tidak cukup

Dalam pemrograman sangat lazim kita ingin menyimpan lebih dari satu benda tanpa mengetahui berapa banyaknya. Sebuah binatu tidak tahu pada Senin pagi berapa banyak pesanan yang akan datang sampai Senin sore. Program yang meminta pengguna mengetikkan nama-nama buah sampai ia mengetik `#stop#` tidak mungkin tahu berapa nama yang akan datang. Berapa banyak variabel `String` yang kita perlukan? Seberapa besar lariknya harus dibuat? Angka berapa pun yang kita pilih pasti salah: terlalu kecil dan program kita macet, terlalu besar dan kita memboroskan memori serta harus menyimpan hitungan terpisah tentang berapa slot yang benar-benar terpakai.

Yang kita inginkan adalah deretan kotak surat yang dapat kita tambahi rak baru ketika sudah penuh. Java menyediakannya persis seperti itu sebagai sebuah kelas, `java.util.ArrayList`. Python membangunnya ke dalam bahasa sebagai tipe `list` yang sudah kita pakai selama ini.

> **DEFINISI.** *List* adalah barisan elemen yang terurut dan panjangnya dapat berubah: elemen dapat ditambahkan di akhir, disisipkan, diganti, dan dihapus, dan setiap elemen tetap mempunyai indeks bilangan bulat yang dimulai dari 0. `ArrayList<T>` di Java menampung elemen bertipe `T`; `list` di Python boleh menampung objek apa pun.

`<String>` dalam `ArrayList<String>` adalah sebuah *type parameter* (parameter tipe). Untuk sementara bacalah sebagai "sebuah `ArrayList` yang menampung `String`, dan hanya `String`". Jika kita mencoba melakukan `add` sebuah `Integer` ke dalamnya, kompilator menolak. Bab 9 menjelaskan tipe *generic* (generik) ini secara lengkap.

5.3.2 Operasi sehari-hari

Kode 5.11 mempraktikkan operasi-operasi yang akan Anda pakai setiap hari: *add*, *size*, *get*, *contains*, *indexOf*, *set*, dan dua jenis *remove*.

Kode 5.11 `FruitList.java`: operasi `ArrayList` sehari-hari.

```
import java.util.ArrayList;

public class FruitList {
    public static void main(String[] args) {
        ArrayList<String> fruit = new ArrayList<>();   // mula-mula kosong
        fruit.add("mango");
        fruit.add("banana");
        fruit.add("peach");
        fruit.add("plum");
        fruit.add("durian");
        fruit.add("plum");
        System.out.println("List: " + fruit + ", size " + fruit.size());
        System.out.println("First fruit: " + fruit.get(0));

        boolean peachIsIn = fruit.contains("peach");
        if (peachIsIn) {
            System.out.println("You entered peach, somewhere, apparently");
        } else {
            System.out.println("No peaches");
        }
        int positionOfPeach = fruit.indexOf("peach");
        int positionOfKiwi = fruit.indexOf("kiwi");
        System.out.println("peach at " + positionOfPeach + ", kiwi at " + positionOfKiwi);

        fruit.set(3, "lemon");                          // plum di posisi 3 menjadi lemon
        System.out.println("After set(3, \"lemon\"): " + fruit);
        String removed = fruit.remove(3);               // menurut posisi: mengembalikan
elemennya
        System.out.println("remove(3) returned " + removed + ": " + fruit);
        boolean gone = fruit.remove("plum");            // menurut nilai: mengembalikan
true atau false
        System.out.println("remove(\"plum\") returned " + gone + ": " + fruit);
        System.out.println("remove(\"plum\") again returned " + fruit.remove("plum"));

        for (int j = 0; j < fruit.size(); j++) {
            System.out.println(j + ": " + fruit.get(j));
        }
    }
}
```

```
List: [mango, banana, peach, plum, durian, plum], size 6
First fruit: mango
You entered peach, somewhere, apparently
peach at 2, kiwi at -1
After set(3, "lemon"): [mango, banana, peach, lemon, durian, plum]
remove(3) returned lemon: [mango, banana, peach, durian, plum]
remove("plum") returned true: [mango, banana, peach, durian]
remove("plum") again returned false
0: mango
1: banana
2: peach
3: durian
```

Baris demi baris:

- `import java.util.ArrayList;` diperlukan di bagian atas karena `ArrayList`, berbeda dengan larik, adalah sebuah kelas dari pustaka standar, bukan bagian dari bahasanya.

- `new ArrayList<>()` membuat list kosong. Kurung sudut yang kosong, yang dijuluki *diamond* (berlian), meminta Java menyimpulkan parameter tipenya dari ruas kiri.
- `fruit.add("mango")` menambahkan elemen di akhir. List itu melebar dengan sendirinya ketika penuh, tanpa kita perlu berbuat apa-apa. Mencetak list secara langsung menghasilkan keluaran yang mudah dibaca, tidak seperti larik.
- `fruit.size()` adalah cacah elemen. Perhatikan tiga ejaan berbeda untuk "berapa banyak": `array.length` (sebuah *field*), `string.length()` (sebuah metode), dan `list.size()` (sebuah metode).
- `fruit.get(0)` membaca elemen pada suatu posisi; operasi yang paling jelas dari semuanya.
- `fruit.contains("peach")` menjawab pertanyaan ya-atau-tidak, dan untuk itulah `boolean` ada. Program ini menyimpan jawabannya di `peachIsIn` lalu bercabang berdasarkan nilai itu.
- `fruit.indexOf("peach")` memberi tahu kita *di mana* persiknya: posisi 2. Jika string yang kita cari tidak ada di dalam list, `indexOf` mengembalikan -1, yang tidak mungkin menjadi indeks yang sah, sehingga aman dipakai sebagai tanda "tidak ada".
- `fruit.set(3, "lemon")` mengganti elemen pada posisi 3. Akhir-akhir ini kita sudah bosan dengan plum dan lebih menyukai lemon.
- Kedua panggilan `remove` sangat berbeda. `fruit.remove(3)` menghapus elemen pada *posisi* 3, apa pun isinya, dan mengembalikan elemen itu (di sini `"lemon"`). `fruit.remove("plum")` menghapus elemen pertama yang *sama dengan* `"plum"` dan mengembalikan sebuah `boolean`: `true` jika ia menemukannya, `false` jika tidak. Ia mengembalikan `boolean` karena pemanggilnya mungkin perlu tahu apakah ada yang terjadi, dan panggilan kedua menampilkan `false`. Semua elemen sesudah elemen yang dihapus bergeser satu tempat ke kiri, sehingga indeks-indeksnya berubah.
- Perulangan terakhir mencetak setiap elemen beserta indeksnya, dari 0 sampai `size() - 1`. Perulangan berhenti ketika `j` sama dengan ukurannya.

`list` di Python menawarkan operasi yang sama, kadang dengan nama yang berbeda dan, yang penting, dengan reaksi yang berbeda terhadap nilai yang tidak ada. **Kode 5.12** menjalankan urutan langkah yang sama.

Kode 5.12 `fruit_list.py`: operasi yang sama pada list Python.

```
fruit = []                                  # mula-mula kosong
fruit.append("mango")
fruit.append("banana")
fruit.append("peach")
fruit.append("plum")
fruit.append("durian")
fruit.append("plum")
print("List:", fruit, "size", len(fruit))
print("First fruit:", fruit[0])

peach_is_in = "peach" in fruit
if peach_is_in:
    print("You entered peach, somewhere, apparently")
else:
    print("No peaches")
position_of_peach = fruit.index("peach")
position_of_kiwi = fruit.index("kiwi") if "kiwi" in fruit else -1   # index() melempar
eksepsi jika tidak ada
print("peach at", position_of_peach, "kiwi at", position_of_kiwi)

fruit[3] = "lemon"                          # plum di posisi 3 menjadi lemon
print('After fruit[3] = "lemon":', fruit)
removed = fruit.pop(3)                      # menurut posisi: mengembalikan elemennya
print("pop(3) returned", removed + ":", fruit)
fruit.remove("plum")                        # menurut nilai: mengembalikan None, melempar
eksepsi jika tidak ada
print('remove("plum"):', fruit)
try:
    fruit.remove("plum")
except ValueError as e:
    print('remove("plum") again raised ValueError:', e)

for j in range(len(fruit)):
    print(str(j) + ": " + fruit[j])
```

```
List: ['mango', 'banana', 'peach', 'plum', 'durian', 'plum'] size 6
First fruit: mango
You entered peach, somewhere, apparently
peach at 2 kiwi at -1
After fruit[3] = "lemon": ['mango', 'banana', 'peach', 'lemon', 'durian', 'plum']
pop(3) returned lemon: ['mango', 'banana', 'peach', 'durian', 'plum']
remove("plum"): ['mango', 'banana', 'peach', 'durian']
remove("plum") again raised ValueError: list.remove(x): x not in list
0: mango
1: banana
2: peach
3: durian
```

Tiga perbedaan patut diperhatikan. Pertama, `"peach" in fruit` adalah versi Python dari `contains`, dan bunyinya seperti kalimat bahasa Inggris biasa. Kedua, `fruit.index("kiwi")` *tidak* mengembalikan −1 untuk nilai yang tidak ada; ia melempar `ValueError`. Itulah sebabnya program memeriksanya lebih dahulu dengan `in`. Ketiga, `fruit.remove("plum")` tidak mengembalikan apa-apa dan melempar `ValueError` jika nilainya tidak ada; program menangkap kesalahan itu dengan `try` dan `except`, sebuah konstruksi yang kita pelajari dengan sungguh-sungguh di Subbab 5.5. Menghapus menurut posisi dilakukan dengan `fruit.pop(3)`, yang mengembalikan elemen yang dihapus, atau `del fruit[3]`, yang tidak mengembalikannya.

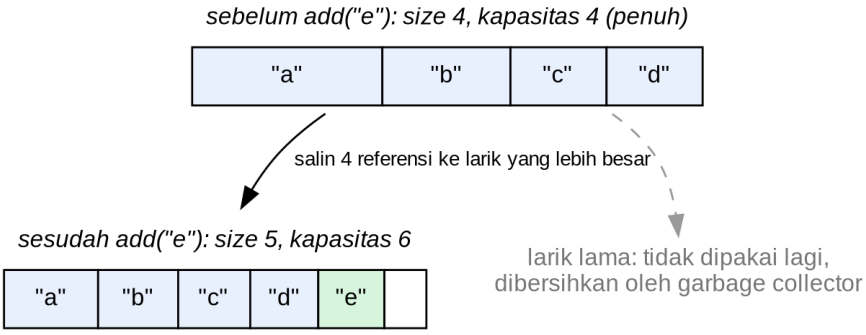
Tabel 5.1 mengumpulkan operasi-operasi itu berdampingan. Simpanlah tabel ini di dekat Anda sepanjang sisa buku.

Tabel 5.1 Operasi sehari-hari pada larik Java, `ArrayList` Java, dan `list` Python.

Operasi	Larik Java `a`	`ArrayList<T> list` Java	`list` Python
Membuat kosong / berukuran	`new T[n]`	`new ArrayList<>()`	`[]` atau `[None] * n`
Membuat dengan nilai	`{x, y, z}`	`new ArrayList<>(List.of(x, y, z))`	`[x, y, z]`
Cacah elemen	`a.length`	`list.size()`	`len(lst)`
Membaca posisi `i`	`a[i]`	`list.get(i)`	`lst[i]`
Mengganti posisi `i`	`a[i] = v`	`list.set(i, v)`	`lst[i] = v`
Menambah di akhir	mustahil	`list.add(v)`	`lst.append(v)`
Menyisipkan di `i`	mustahil	`list.add(i, v)`	`lst.insert(i, v)`
Menghapus posisi `i`	mustahil	`list.remove(i)` (mengembalikannya)	`lst.pop(i)` atau `del lst[i]`
Menghapus nilai `v`	mustahil	`list.remove(v)` (mengembalikan `boolean`)	`lst.remove(v)` (melempar eksepsi jika tidak ada)
Apakah `v` ada?	perulangan	`list.contains(v)`	`v in lst`
Posisi `v`	perulangan	`list.indexOf(v)` (−1 jika tidak ada)	`lst.index(v)` (melempar eksepsi jika tidak ada)
Indeks yang salah	`ArrayIndexOutOfBoundsException`	`IndexOutOfBoundsException`	`IndexError`

5.3.3 Bagaimana list tumbuh

List yang "tumbuh dengan sendirinya" terdengar seperti sulap, tetapi sebenarnya itu hanyalah trik dari Subbab 5.2.5 yang dikerjakan untuk kita. Di dalam setiap `ArrayList` terdapat sebuah larik biasa, umumnya dengan beberapa slot cadangan di ujungnya. List itu menyimpan dua bilangan: *size* (ukuran), yaitu cacah slot yang terpakai, dan *capacity* (kapasitas), yaitu panjang larik tersembunyi itu. Selama ukuran lebih kecil daripada kapasitas, `add` cukup mengisi slot cadangan berikutnya. Ketika lariknya penuh, `add` membuat larik yang lebih besar, menyalin setiap referensi ke sana, lalu menambahkan elemen yang baru. **Gambar 5.2** memperlihatkan saat itu.



Gambar 5.2 Yang dilakukan `add` ketika larik tersembunyinya penuh: larik yang lebih besar dibuat, referensi-referensi lama disalin ke dalamnya, dan larik lama dibuang.

Seberapa besar larik yang baru itu seharusnya? Pilihan ini lebih penting daripada kelihatannya. Misalkan kita memperbesarnya hanya satu slot setiap kali. Menambahkan elemen ke-n berarti menyalin $n-1$ elemen yang sudah ada, dan menambahkan n elemen ke list kosong akan menyalin

$$0 + 1 + 2 + \cdots + (n-1) = \frac{n(n-1)}{2} \tag{5.2}$$

elemen secara keseluruhan. Untuk $n = 1000$ itu berarti $1000 \times 999/2 = 499{,}500$ penyalinan; pekerjaannya tumbuh sebanding dengan kuadrat n. Sekarang misalkan sebaliknya kita menggandakan kapasitas setiap kali lariknya penuh, dimulai dari 1. Penyalinan terjadi ketika ukurannya mencapai 1,2,4,8, ..., dan masing-masing menyalin elemen sebanyak itu. Jika pangkat dua terbesar di bawah n adalah 2^k, totalnya adalah

$$1 + 2 + 4 + \cdots + 2^k = 2^{k+1} - 1 < 2n \tag{5.3}$$

penyalinan. Untuk $n = 1000$, dengan $2^9 = 512$, itu berarti 1023 penyalinan: rata-rata kurang dari satu penyalinan per elemen, meskipun beberapa panggilan `add` tertentu menyalin ratusan elemen. Itulah sebabnya menambahkan elemen di akhir list, secara rata-rata, sama murahnya dengan menulis ke dalam larik. `ArrayList` di Java tumbuh sebesar setengah kapasitasnya setiap kali (faktor 1.5) dan `list` di CPython dengan faktor yang sedikit lebih kecil; keduanya adalah pertumbuhan geometris, dan argumen Persamaan (5.3) berlaku untuk faktor pertumbuhan apa pun yang lebih besar dari 1, hanya dengan konstanta yang berbeda. Soal Latihan 5.4 meminta Anda mengukur kedua strategi, dan penyelesaiannya menegaskan angka 499,500 dan 1023 dengan menjalankan kodenya.

5.3.4 List berisi bilangan: *boxing*

Parameter tipe di Java harus berupa tipe *kelas*, tidak pernah tipe primitif. `ArrayList<int>` tidak dapat dikompilasi. Untuk list bilangan bulat kita menulis `ArrayList<Integer>`, dengan `Integer` sebagai *wrapper class* (kelas pembungkus) bagi `int` (ada pula `Double` untuk `double`, `Boolean` untuk `boolean`, dan seterusnya). Java mengonversi `int` dan `Integer` bolak-balik secara otomatis; konversi menjadi objek disebut *boxing* (pembungkusan) dan konversi sebaliknya *unboxing* (pembukaan bungkus). Umumnya hal ini tidak terlihat dan menyenangkan. Ada satu pengecualian yang terkenal, yang diperlihatkan **Kode 5.13**.

Kode 5.13 `BoxingDemo.java`: *boxing*, *unboxing*, dan dua makna `remove` pada list berisi `Integer`.

```java
import java.util.ArrayList;

public class BoxingDemo {
    public static void main(String[] args) {
        ArrayList<Integer> weights = new ArrayList<>();  // Integer, tidak pernah int
        weights.add(5);                                  // int 5 dibungkus menjadi Integer
        weights.add(10);
        weights.add(3);
        weights.add(7);
        System.out.println("weights: " + weights);

        weights.remove(3);                               // sebuah int: menghapus INDEKS 3
        System.out.println("after remove(3): " + weights);

        weights.remove(Integer.valueOf(3));              // sebuah Integer: menghapus NILAI
3
        System.out.println("after remove(Integer.valueOf(3)): " + weights);

        int first = weights.get(0);                      // dibuka kembali menjadi int
        System.out.println("first * 2 = " + (first * 2));
    }
}
```

```
weights: [5, 10, 3, 7]
after remove(3): [5, 10, 3]
after remove(Integer.valueOf(3)): [5, 10]
first * 2 = 10
```

`ArrayList` mempunyai dua metode `remove`: `remove(int index)` dan `remove(Object value)`. Ketika kita menulis `weights.remove(3)`, argumen `3` adalah sebuah `int`, yang cocok persis dengan metode pertama, sehingga Java menghapus *indeks* 3, yaitu nilai 7, dan bukan nilai 3 seperti yang mungkin kita harapkan. Untuk menghapus nilai 3 kita harus menyerahkan sebuah objek `Integer`: `weights.remove(Integer.valueOf(3))`. Jebakan ini hanya menggigit list berisi `Integer`, dan ia menggigit setiap orang sekali.

Python tidak mempunyai tipe primitif dan tidak mengenal *boxing*: sebuah `int` sudah merupakan objek dan dapat langsung dimasukkan ke dalam list. Python menghindari ambiguitas itu dengan memakai kata-kata yang berbeda, seperti yang ditunjukkan **Kode 5.14**: `del weights[3]` untuk posisi, `weights.remove(3)` untuk nilai.

Kode 5.14 `boxing_demo.py`: menghapus menurut posisi dan menurut nilai di Python.

```python
weights = []                                # objek apa pun boleh masuk; int adalah objek
weights.append(5)
weights.append(10)
weights.append(3)
weights.append(7)
print("weights:", weights)

del weights[3]                              # menurut INDEKS: menghapus posisi 3
print("after del weights[3]:", weights)

weights.remove(3)                           # menurut NILAI: menghapus 3 yang pertama
print("after weights.remove(3):", weights)

first = weights[0]
print("first * 2 =", first * 2)
```

```
weights: [5, 10, 3, 7]
after del weights[3]: [5, 10, 3]
after weights.remove(3): [5, 10]
first * 2 = 10
```

5.3.5 Larik atau list?

Pertanyaan yang bagus. Larik, boleh dikatakan, lebih mendasar daripada list. Larik tidak memerlukan `import`, bekerja langsung dengan tipe primitif (sebuah `int[]` menyimpan bilangan-bilangan itu sendiri, bukan objek `Integer`, sehingga lebih kecil dan lebih cepat), panjangnya tetap (yang memang tepat untuk hal-hal yang jumlahnya sungguh tetap, seperti tujuh hari dalam sepekan atau 8 × 8 petak papan catur), dan larik dua dimensi mudah dibayangkan serta diindeks. Tetapi larik tidak dapat mengubah panjangnya. Jika Anda perlu memperbesarnya, Anda terjebak dengan penyalinan bertele-tele dari Subbab 5.2.5, yang justru dikerjakan `ArrayList` untuk Anda. **Tabel 5.2** merangkum pilihan tersebut.

Tabel 5.2 Memilih antara larik dan list.

Pertanyaan	Larik	List (`ArrayList` / `list`)
Apakah cacah elemen sudah diketahui saat wadah dibuat, dan tetap?	ya: sangat cocok	bisa, tetapi keluwesannya tidak terpakai
Apakah elemen akan ditambah atau dihapus kelak?	canggung: salin ke larik baru	ya: `add`, `remove`, `append`, `pop`
Bilangan primitif di Java?	`int[]`, `double[]` menyimpannya secara langsung	`ArrayList<Integer>` membungkus setiap bilangan
Kisi atau tabel dengan bentuk tetap?	`int[][]` jelas dan cepat	list berisi list juga bisa
Memerlukan `main(String[] args)` atau API yang menerima larik?	wajib	konversikan dengan `toArray`

Dalam Java sehari-hari, bila ragu, pilihlah `ArrayList`. Dalam Python sehari-hari pertanyaan ini hampir tidak pernah muncul: pakailah `list`.

5.4 Membaca Masukan dari Papan Ketik

5.4.1 `Scanner` dan `input()`

Sejauh ini kita tahu cara membuat program mencetak. Namun cukup sering kita juga memerlukannya untuk *membaca*. Alat paling sederhana di Java untuk membaca masukan yang diketik adalah kelas `java.util.Scanner`. Kita membangunnya di sekeliling `System.in`, yaitu *standard input* (masukan standar) program, yang lazimnya adalah papan ketik:

```
Scanner sc = new Scanner(System.in);
String aLineOfText = sc.nextLine();
```

`sc.nextLine()` menunggu sampai pengguna menekan Enter lalu mengembalikan segala sesuatu yang diketik pada baris itu, tanpa karakter pindah baris, sebagai sebuah `String`. `Scanner` juga menawarkan `nextInt()`, `nextDouble()`, `nextBoolean()`, dan `next()` biasa, yang masing-masing membaca satu *token*, yaitu serangkaian karakter yang dipisahkan dari tetangganya oleh spasi atau pindah baris, lalu mengonversinya ke tipe yang diminta. **Kode 5.15** memakai tiga di antaranya.

Kode 5.15 `SquareInput.java`: membaca satu baris, satu bilangan bulat, satu bilangan desimal, dan satu baris lagi dengan `Scanner`.

```java
import java.util.Scanner;

public class SquareInput {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("What is your name?");
        String name = sc.nextLine();                  // seluruh baris
        System.out.println("Pick an integer: ");
        int value = sc.nextInt();                     // satu token, dibaca sebagai int
        System.out.println("Hello, " + name + ". You picked " + value
                + ". The square of that is " + (value * value) + ".");
        System.out.println("How many kilograms of laundry?");
        double kg = sc.nextDouble();
        sc.nextLine();                                // telan sisa baris itu
        System.out.println("Any note for the courier?");
        String note = sc.nextLine();
        System.out.println(kg + " kg noted, with the note \"" + note + "\".");
    }
}
```

Agar setiap kali jalan program dapat diulang persis sama, dan agar Anda dapat melihat dengan tepat apa yang diketikkan, kita menyimpan masukan yang diketik itu di sebuah berkas di sebelah program, `SquareInput.input.txt`:

```
Sari
12
3.5
Please ring twice
```

lalu kita *mengalihkan* (*redirect*) masukan standar program dari berkas itu, sehingga `Scanner` membaca berkas alih-alih papan ketik:

```
java SquareInput < SquareInput.input.txt
```

Tanda `<` bekerja dengan cara yang sama di Command Prompt Windows, di PowerShell (sebagai `Get-Content SquareInput.input.txt | java SquareInput`), dan di terminal macOS maupun Linux. Karena masukan kini datang dari berkas, teks yang diketik tidak ditampilkan ulang di layar, sehingga keluaran di bawah ini hanya memuat apa yang dicetak oleh program:

```
What is your name?
Pick an integer:
Hello, Sari. You picked 12. The square of that is 144.
How many kilograms of laundry?
Any note for the courier?
3.5 kg noted, with the note "Please ring twice".
```

Jika Anda menjalankan program tanpa pengalihan, ia akan menunggu Anda mengetik sesudah setiap pertanyaan.

Satu baris dalam program itu perlu dijelaskan: `sc.nextLine();` yang hasilnya dibuang. `nextDouble()` membaca token `3.5` dan berhenti *tepat sesudahnya*, sehingga karakter pindah baris yang mengakhiri baris itu belum terbaca. Jika sesudah itu kita memanggil `nextLine()` untuk membaca catatan bagi kurir, ia akan langsung menemukan sisa pindah baris itu dan mengembalikan string kosong `""`, dan catatan yang sebenarnya tidak akan dibaca oleh siapa pun. `nextLine()` tambahan itu menelan sisa baris tempat bilangan tadi berada. Setiap pemrogram Java pernah terjerat oleh hal ini sekali. Aturannya: *sesudah `nextInt()`, `nextDouble()`, atau `next()`, panggil `nextLine()` satu kali sebelum Anda menginginkan satu baris utuh.*

Padanannya di Python adalah fungsi bawaan `input()`, yang membaca satu baris utuh dan mengembalikannya sebagai string, tanpa karakter pindah baris. Tidak ada varian bertipe; sebagai gantinya, Anda sendiri yang mengonversi string itu dengan `int(...)` atau `float(...)`. **Kode 5.16**

adalah program yang sama, dijalankan dengan `python3 square_input.py < square_input.input.txt` dan berkas masukan yang sama.

Kode 5.16 `square_input.py`: dialog yang sama dengan `input()`, `int()`, dan `float()`.

```
print("What is your name?")
name = input()                              # seluruh baris, sebagai str
print("Pick an integer: ")
value = int(input())                        # ubah teks menjadi int
print("Hello, " + name + ". You picked " + str(value)
      + ". The square of that is " + str(value * value) + ".")
print("How many kilograms of laundry?")
kg = float(input())
print("Any note for the courier?")
note = input()                              # tidak ada yang perlu ditelan: input()
membaca baris utuh
print(str(kg) + ' kg noted, with the note "' + note + '".')
```

```
What is your name?
Pick an integer:
Hello, Sari. You picked 12. The square of that is 144.
How many kilograms of laundry?
Any note for the courier?
3.5 kg noted, with the note "Please ring twice".
```

Karena `input()` selalu menghabiskan satu baris utuh, Python tidak mempunyai jebakan sisa pindah baris. Anda juga dapat menyerahkan pertanyaannya langsung kepada `input`, seperti `name = input("What is your name? ")`, yang mencetak pertanyaan itu tanpa pindah ke baris baru. Kita memakai panggilan `print` terpisah agar keluaran Java dan Python sama baris demi baris.

Bagaimana jika pengguna mengetik sesuatu yang bukan bilangan, misalnya `twelve`? `nextInt()` di Java dan `int()` di Python sama-sama menolak, dan programnya macet. Jangan khawatir: itu hal yang wajar, dan Subbab 5.5 dikhususkan untuk menangani kesalahan semacam itu dengan cara yang lebih baik.

5.4.2 Membaca sampai bertemu *sentinel*

Ketika sebuah program tidak mungkin mengetahui berapa banyak nilai yang akan datang, kebiasaan yang lazim adalah menyepakati sebuah nilai khusus, yang disebut *sentinel* (penanda akhir), yang berarti "sudah habis". **Kode 5.17** membaca nama-nama buah sampai pengguna mengetik `#stop#`, sambil mengumpulkannya di dalam sebuah `ArrayList`. Program ini menyatukan semua yang ada di bab ini sejauh ini: masukan, perulangan `while`, dan list yang tumbuh.

Kode 5.17 `FruitCollector.java`: membaca baris-baris ke dalam `ArrayList` sampai bertemu *sentinel* `#stop#`.

```java
import java.util.ArrayList;
import java.util.Scanner;

public class FruitCollector {
    public static void main(String[] args) {
        System.out.println("Enter some fruit names. Enter #stop# to finish");
        Scanner s_in = new Scanner(System.in);
        ArrayList<String> fruit = new ArrayList<String>();
        String line = s_in.nextLine();
        // Terus tambahkan buah ke dalam list
        while (!line.equals("#stop#")) {
            System.out.println("You just typed " + line + ". Adding it!");
            fruit.add(line);
            line = s_in.nextLine();
        }
        System.out.println("You entered " + fruit.size()
                + " fruit. Can't you do better?");
    }
}
```

Dengan berkas masukan ini:

```
srikaya
salak
rambutan
#stop#
```

program mencetak:

```
Enter some fruit names. Enter #stop# to finish
You just typed srikaya. Adding it!
You just typed salak. Adding it!
You just typed rambutan. Adding it!
You entered 3 fruit. Can't you do better?
```

Perhatikan apa yang sedang terjadi. Kita menyiapkan `Scanner`, membuat `ArrayList` berisi `String`, membaca baris pertama, lalu berulang: selama baris itu bukan *sentinel*, laporkan, tambahkan ke list, dan baca baris berikutnya. Di akhir, `size()` memberi tahu kita berapa banyak yang telah dimasukkan. Perhatikan *priming read* (pembacaan pancingan) sebelum perulangan: `nextLine()` yang pertama terjadi sebelum pengujian `while`, dan setiap pembacaan berikutnya terjadi di bagian bawah badan perulangan, sehingga pengujian selalu memeriksa baris yang baru saja kita baca. Perhatikan pula bahwa kita membandingkan string dengan `equals`, tidak pernah dengan `==`, sebagaimana dijelaskan Bab 3.

Kode 5.18 adalah versi Python-nya.

Kode 5.18 `fruit_collector.py`: perulangan *sentinel* yang sama di Python.

```python
print("Enter some fruit names. Enter #stop# to finish")
fruit = []
line = input()
# Terus tambahkan buah ke dalam list
while line != "#stop#":
    print("You just typed " + line + ". Adding it!")
    fruit.append(line)
    line = input()
print("You entered " + str(len(fruit)) + " fruit. Can't you do better?")
```

```
Enter some fruit names. Enter #stop# to finish
You just typed srikaya. Adding it!
You just typed salak. Adding it!
You just typed rambutan. Adding it!
You entered 3 fruit. Can't you do better?
```

Kedua versi berbagi satu kelemahan: jika masukan berakhir tanpa baris `#stop#` (misalnya berkasnya terpotong, atau pengguna menekan Ctrl+D di macOS atau Linux, atau Ctrl+Z lalu Enter di Windows), maka `nextLine()` melempar `NoSuchElementException` dan `input()` melempar `EOFError`. Subbab 5.5 menunjukkan cara menanggapinya, dan Subbab 5.8 menunjukkan perulangan yang berhenti dengan rapi di akhir masukan tanpa *sentinel* sama sekali.

5.5 Ketika Sesuatu Salah: Eksepsi

5.5.1 Program berjumpa dengan pengguna

Salah satu masalah dalam pemrograman adalah pengguna. Kita tidak pernah tahu kapan kesalahan akan terjadi; yang paling baik dapat kita lakukan adalah menduga apa yang mungkin menyebabkannya dan meredam akibatnya. Terkadang, harus diakui, kesalahan terjadi karena kita sendiri, para pemrogram, sama sekali tidak mengantisipasi sebuah masalah. Untuk mencakup semua kasus itu, Java dan Python memberi kita satu cara yang seragam untuk menyatakan bahwa ada yang salah: *exception* (eksepsi).

> **DEFINISI.** *Exception* (eksepsi) adalah objek yang menguraikan suatu kejadian tidak normal selama eksekusi program. Ketika kejadian itu terdeteksi, eksepsi *dilemparkan* (*thrown*; Python menyebutnya *raised*): eksekusi normal berhenti di titik itu, dan eksepsi berjalan kembali melalui metode-metode yang telah dipanggil sampai ada kode yang *menangkap*-nya (*catch*) dan menanganinya. Jika tidak ada yang menangkapnya, program berakhir dan mencetak sebuah laporan.

Kita sudah bertemu tiga eksepsi: `ArrayIndexOutOfBoundsException` dan `IndexError` di Subbab 5.2.3, serta `ValueError` dari list Python di Subbab 5.3.2. Eksepsi lain yang akan segera Anda jumpai adalah `InputMismatchException` dan `NullPointerException`. Semuanya adalah cara bahasa pemrograman berkata "ada yang salah, tetapi ini jenis hal yang dapat ditangani oleh program".

Perhatikan **Kode 5.19**. Apa yang bisa salah di dalamnya?

Kode 5.19 `InputTest.java`: membaca bilangan bulat sampai bertemu 99, tanpa perlindungan terhadap masukan yang buruk.

```
import java.util.*;

public class InputTest {
    public static void main(String[] args) {
        Scanner s = new Scanner(System.in);
        ArrayList<Integer> intList = new ArrayList<>();
        System.out.println("Enter some numbers!");
        int currentVal = s.nextInt();
        while (currentVal != 99) {
            System.out.println("Adding " + currentVal);
            intList.add(currentVal);
            currentVal = s.nextInt();
        }
        System.out.println("Exiting: List is " + intList.toString());
    }
}
```

Jika Anda menyadari bahwa bahayanya terletak pada masukan dari pengguna, bagus sekali. Pikirkan apa yang diharapkan oleh `int currentVal = s.nextInt()`: pengguna seharusnya memasukkan sebuah bilangan. Jika pengguna memasukkan string yang tidak dapat dibaca sebagai bilangan, misalnya `b`, Java tidak akan senang. Inilah berkas masukannya,

```
1
2
b
3
99
```

dan inilah yang terjadi:

```
Enter some numbers!
Adding 1
Adding 2
Exception in thread "main" java.util.InputMismatchException
    at java.base/java.util.Scanner.throwFor(Scanner.java:947)
    at java.base/java.util.Scanner.next(Scanner.java:1602)
    at java.base/java.util.Scanner.nextInt(Scanner.java:2267)
    at java.base/java.util.Scanner.nextInt(Scanner.java:2221)
    at InputTest.main(InputTest.java:12)
```

Aduh. Apa artinya ini? Artinya kita perlu membetulkan sesuatu, dan laporan itu memberi tahu kita apa dan di mana.

5.5.2 Membaca *stack trace*

Baris pertama laporan menyebut nama eksepsinya: kita boleh mengabaikan kata-kata `in thread "main"` dan langsung melihat tipenya, `java.util.InputMismatchException`. Itu masuk akal: `nextInt` mengharapkan sebuah `int`, dan kita mengetik sesuatu yang tidak dapat ditafsirkan sebagai `int`. Yang menyusul sesudahnya disebut *stack trace* (jejak tumpukan). Isinya adalah daftar setiap metode yang sedang berjalan ketika eksepsi terjadi, dimulai dari yang paling baru. Empat baris teratas berada di dalam `Scanner` itu sendiri, di pustaka Java, tempat masalahnya terdeteksi; nama berkas dan nomor baris di sana (`Scanner.java:947` dan seterusnya) milik pustaka dan dapat berbeda antarversi Java. Baris terakhirlah yang penting bagi kita: `at InputTest.main(InputTest.java:12)`. Kelas kita menyebabkan kesalahan pada baris 12, yaitu `s.nextInt()` yang *kedua*, yang berada di dalam perulangan. `nextInt()` yang pertama berhasil membaca `1`, begitu pula panggilan yang membaca `2`; panggilan ketiga, yang membaca `b`, gagal.

> **WAWASAN KUNCI:** Bacalah *stack trace* Java dari atas untuk mengetahui *apa* yang salah, lalu telusuri ke bawah sampai baris pertama yang menyebut kelas *Anda sendiri* untuk mengetahui *di mana*. Baris-baris yang diawali `java.base/` adalah pustaka; masalahnya hampir selalu berada pada baris pertama milik Anda di bawahnya.

Kode 5.20 adalah program yang sama di Python, dengan masukan yang sama.

Kode 5.20 `input_test.py`: perulangan tanpa perlindungan yang sama di Python.

```python
int_list = []
print("Enter some numbers!")
current_val = int(input())
while current_val != 99:
    print("Adding " + str(current_val))
    int_list.append(current_val)
    current_val = int(input())
print("Exiting: List is " + str(int_list))
```

```
Enter some numbers!
Adding 1
Adding 2
Traceback (most recent call last):
  File "/home/claude/kit/OOP/Code_new/Chapter05/python/input_test.py", line 7, in <module>
    current_val = int(input())
                  ^^^^^^^^^^^
ValueError: invalid literal for int() with base 10: 'b'
```

Python menyebut laporannya *traceback*, dan urutannya terbalik: panggilan yang paling baru berada di *bawah*, itulah sebabnya baris pertamanya berbunyi `most recent call last` (panggilan terbaru ditaruh terakhir). Jadi *traceback* Python dibaca dari bawah ke atas: baris terakhir memberikan tipe eksepsinya, `ValueError`, beserta pesan yang bahkan mengutip teks penyebabnya, `'b'`; baris-baris di atasnya memberikan nama berkas, nomor baris (7), dan kodenya. Baris 7, yaitu `int(input())` di dalam perulangan, itulah yang gagal.

Stack trace menjadi lebih menarik ketika metode memanggil metode lain. **Kode 5.21** membaca tiga baris pesanan pendek. `main` memanggil `loadAll`, yang memanggil `readOrder` untuk setiap baris, yang memanggil `parseWeight`. Ketika `parseWeight` menjumpai berat `-2`, ia sendiri *melempar* eksepsi, dengan kata kunci `throw` yang dijelaskan Subbab 5.5.7. Eksepsi itu ditangkap di `main`, yang mencetak sebuah pesan lalu *stack trace*-nya dengan `printStackTrace`.

Kode 5.21 `PropagationDemo.java`: eksepsi yang dilempar tiga panggilan di bawah dan ditangkap di `main`.

```
public class PropagationDemo {
    static double parseWeight(String text) {
        double kg = Double.parseDouble(text);         // dapat melempar
NumberFormatException
        if (kg <= 0) {
            throw new IllegalArgumentException("weight must be positive: " + text);
        }
        return kg;
    }

    static void readOrder(String line) {
        String[] parts = line.split(",");
        double kg = parseWeight(parts[1]);
        System.out.println("Read " + parts[0] + ", " + kg + " kg");
    }

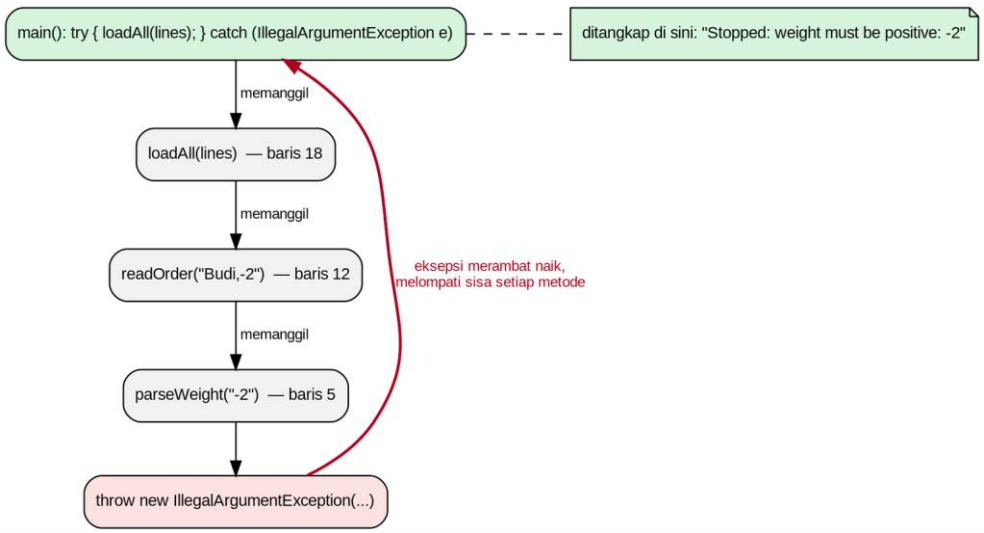
    static void loadAll(String[] lines) {
        for (String line : lines) {
            readOrder(line);
        }
        System.out.println("All lines loaded.");       // baris di bawah tidak pernah
tercapai
    }

    public static void main(String[] args) {
        String[] lines = {"Sari,3.5", "Budi,-2", "Tono,4"};
        try {
            loadAll(lines);
        } catch (IllegalArgumentException e) {
            System.out.println("Stopped: " + e.getMessage());
            e.printStackTrace(System.out);            // jejak yang ditinggalkan eksepsi
        }
    }
}
```

```
Read Sari, 3.5 kg
Stopped: weight must be positive: -2
java.lang.IllegalArgumentException: weight must be positive: -2
    at PropagationDemo.parseWeight(PropagationDemo.java:5)
    at PropagationDemo.readOrder(PropagationDemo.java:12)
    at PropagationDemo.loadAll(PropagationDemo.java:18)
    at PropagationDemo.main(PropagationDemo.java:26)
```

Gambar 5.3 memperlihatkan apa yang terjadi. Panggilan-panggilan bergerak turun: dari `main` ke `loadAll` ke `readOrder` ke `parseWeight`. Eksepsinya kembali naik melalui jalur yang sama. `parseWeight` tidak sampai ke `return`-nya; `readOrder` tidak sampai ke `println`-nya; `loadAll` tidak

melanjutkan ke baris ketiga, `"Tono,4"`, dan tidak pernah mencetak “All lines loaded.”; setiap metode ditinggalkan tepat di titik tempat ia melakukan panggilannya. Baru ketika eksepsi mencapai `try` di `main`, yang klausa `catch`-nya menyebut `IllegalArgumentException`, perjalanannya berhenti. Perjalanan kembali menaiki rantai panggilan ini disebut *propagation* (propagasi), dan *stack trace* adalah buku harian eksepsi tentang perjalanan itu: baris 5 di `parseWeight`, dipanggil dari baris 12 di `readOrder`, dipanggil dari baris 18 di `loadAll`, dipanggil dari baris 26 di `main`.



Gambar 5.3 Propagasi. Panggilan bergerak turun dari `main`; eksepsi berjalan kembali ke atas, meninggalkan setiap metode yang dilaluinya, sampai mencapai `catch` yang cocok.

Versi Python-nya, **Kode 5.22**, berperilaku persis sama. `traceback.print_exc` mencetak *traceback* dari eksepsi yang sedang ditangani, dengan panggilan terbaru di akhir.

Kode 5.22 `propagation_demo.py`: propagasi yang sama di Python.

```python
import sys
import traceback


def parse_weight(text):
    kg = float(text)                          # dapat melempar ValueError
    if kg <= 0:
        raise ValueError("weight must be positive: " + text)
    return kg


def read_order(line):
    parts = line.split(",")
    kg = parse_weight(parts[1])
    print("Read " + parts[0] + ", " + str(kg) + " kg")


def load_all(lines):
    for line in lines:
        read_order(line)
    print("All lines loaded.")                # baris di bawah tidak pernah tercapai


def main():
    lines = ["Sari,3.5", "Budi,-2", "Tono,4"]
    try:
        load_all(lines)
    except ValueError as e:
        print("Stopped: " + str(e))
        traceback.print_exc(file=sys.stdout)  # jejak yang ditinggalkan eksepsi


if __name__ == "__main__":
    main()
```

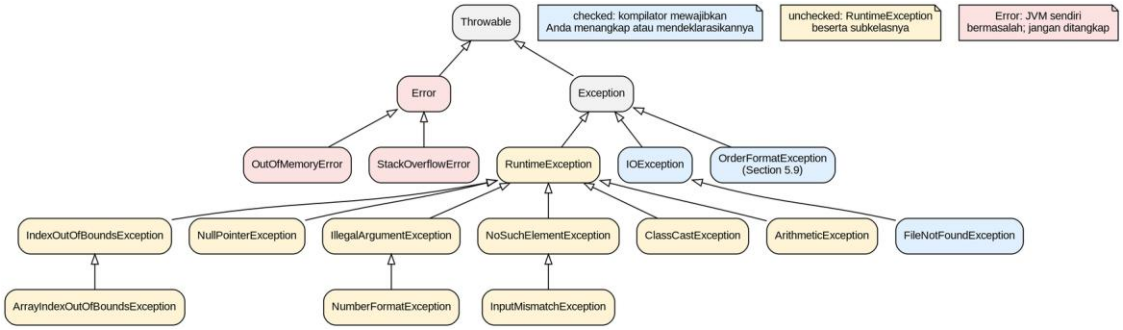
```
Read Sari, 3.5 kg
Stopped: weight must be positive: -2
Traceback (most recent call last):
  File "/home/claude/kit/OOP/Code_new/Chapter05/python/propagation_demo.py", line 27, in
main
    load_all(lines)
  File "/home/claude/kit/OOP/Code_new/Chapter05/python/propagation_demo.py", line 20, in
load_all
    read_order(line)
  File "/home/claude/kit/OOP/Code_new/Chapter05/python/propagation_demo.py", line 14, in
read_order
    kg = parse_weight(parts[1])
         ^^^^^^^^^^^^^^^^^^^^^^
  File "/home/claude/kit/OOP/Code_new/Chapter05/python/propagation_demo.py", line 8, in
parse_weight
    raise ValueError("weight must be positive: " + text)
ValueError: weight must be positive: -2
```

5.5.3 Silsilah kegagalan

Ada ratusan tipe eksepsi. Semuanya bukan tumpukan yang acak-acakan: mereka membentuk silsilah kelas, tempat eksepsi yang lebih spesifik *memperluas* (*extends*) eksepsi yang lebih umum. (Kata *extends* adalah kata kunci Java untuk pewarisan, yang dipelajari Bab 7 secara mendalam; untuk sementara bacalah "A extends B" sebagai "setiap A adalah sejenis B".) Inilah garis keturunan `ArrayIndexOutOfBoundsException` dalam dokumentasi Java:

```
java.lang.Object
    extended by java.lang.Throwable
        extended by java.lang.Exception
            extended by java.lang.RuntimeException
                extended by java.lang.IndexOutOfBoundsException
                    extended by java.lang.ArrayIndexOutOfBoundsException
```

Jadi `ArrayIndexOutOfBoundsException` *adalah* sebuah `IndexOutOfBoundsException`, yang *adalah* sebuah `RuntimeException`, yang *adalah* sebuah `Exception`, yang *adalah* sebuah `Throwable`. Segala sesuatu yang dapat dilemparkan di Java memperluas `Throwable`. **Gambar 5.4** memperlihatkan bagian pohon yang kita jumpai di bab ini.

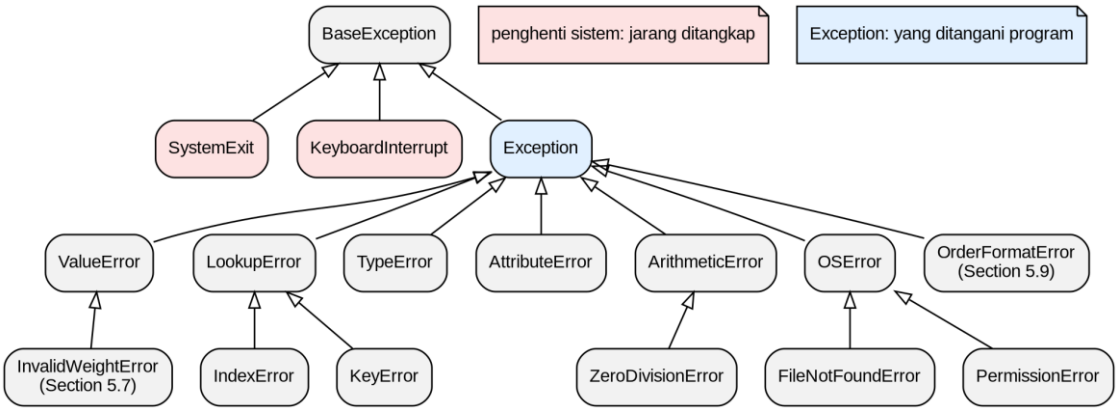


Gambar 5.4 Sebagian hierarki eksepsi Java. Kelas berwarna biru adalah eksepsi terperiksa, yang kuning tak terperiksa, dan yang merah adalah *error* yang tidak semestinya dicoba ditangani oleh program.

Tepat di bawah `Throwable` terdapat dua cabang.

- `Error` mempunyai makna khusus yang buruk di Java. Jika program kita menghasilkan `Error` dan bukan `Exception`, ada sesuatu yang sedemikian salahnya sehingga Java tidak dapat memulihkan diri: mesin kehabisan memori (`OutOfMemoryError`), atau sebuah metode memanggil dirinya sendiri begitu banyak kali sehingga tumpukan panggilannya meluap (`StackOverflowError`). Kita tidak akan menangkap yang seperti ini.
- `Exception` adalah cabang untuk masalah-masalah yang mungkin masuk akal ditangani oleh program. Di dalamnya ada satu percabangan lagi yang sangat penting: subkelas-subkelas `RuntimeException` di satu sisi (kuning), dan semua yang lain di sisi lainnya (biru). Subbab 5.6 dikhususkan untuk percabangan itu.

Pohon milik Python, pada **Gambar 5.5**, mengusung gagasan yang sama dengan nama-nama yang berbeda [1]. Di akarnya terdapat `BaseException`. Di bawahnya ada beberapa eksepsi yang berarti "program sedang diminta berhenti", seperti `SystemExit` (dilempar oleh `sys.exit`) dan `KeyboardInterrupt` (dilempar ketika pengguna menekan Ctrl+C), lalu `Exception`, leluhur segala sesuatu yang ditangani program biasa: `ValueError`, `TypeError`, `IndexError` dan `KeyError` (keduanya jenis `LookupError`), `ZeroDivisionError`, serta `OSError` dengan anak-anaknya `FileNotFoundError` dan `PermissionError`.



Gambar 5.5 Sebagian hierarki eksepsi Python. Program menangkap subkelas-subkelas `Exception`; eksepsi-eksepsi di sampingnya meminta program berhenti.

Mengapa silsilah ini penting? Karena klausa `catch` yang menyebut sebuah kelas juga menangkap setiap keturunan kelas itu. Jaring berlabel `IndexOutOfBoundsException` juga menangkap `ArrayIndexOutOfBoundsException`; jaring berlabel `Exception` menangkap hampir segalanya. Kita memanfaatkan kenyataan itu di Subbab 5.5.5.

5.5.4 `try` dan `catch`

Kita sudah mengatakan bahwa eksepsi dilemparkan. Untuk menanganinya, eksepsi itu harus *ditangkap*. Caranya, kita membungkus kode yang dapat menimbulkan eksepsi di dalam blok `try` dan menambahkan blok `catch` yang menyatakan apa yang harus dilakukan:

```
try {
    // kode yang dapat menimbulkan eksepsi
} catch (SomeExceptionType et) {
    // apa yang dilakukan bila eksepsi itu terjadi
}
```

Versi Python memakai `try` dan `except`:

```
try:
    # kode yang dapat menimbulkan eksepsi
except SomeExceptionType as et:
    # apa yang dilakukan bila eksepsi itu terjadi
```

Aturannya sama di kedua bahasa. Kita mengelilingi kode yang "berbahaya" dengan blok `try`. Jika tidak terjadi eksepsi, seluruh blok `try` dijalankan dan eksekusi berlanjut ke kode sesudah keseluruhan konstruksi itu; blok `catch` dilewati. Jika terjadi eksepsi, sisa blok `try` dilewati dan tipe eksepsinya dibandingkan dengan tipe di dalam tanda kurung. Jika cocok (artinya eksepsi itu berasal dari kelas tersebut *atau dari kelas mana pun di bawahnya* dalam silsilah), objek eksepsinya diberi nama yang kita pilih, di sini `et`, dan blok `catch` dijalankan. Sesudah blok `catch`, eksekusi berlanjut sesudah keseluruhan konstruksi. Program tidak macet. Jika tipenya tidak cocok, eksepsi itu terus berpropagasi seolah-olah `try` itu tidak ada.

Tujuan utama kita menangkap eksepsi adalah mencegah program berhenti dengan kesalahan. **Kode 5.23** memperbaiki `InputTest`: setiap kali pengguna memasukkan sesuatu yang bukan bilangan, Java melempar eksepsi, blok `catch` melaporkannya, dan perulangan terus berjalan, semuanya tanpa macet.

Kode 5.23 `SafeInput.java`: menangkap `InputMismatchException` sehingga masukan yang buruk dilaporkan lalu dilewati.

```
import java.util.ArrayList;
import java.util.InputMismatchException;
import java.util.Scanner;

public class SafeInput {
    public static void main(String[] args) {
        Scanner s = new Scanner(System.in);
        ArrayList<Integer> intList = new ArrayList<>();
        System.out.println("Enter some numbers!");
        int currentVal;
        while (s.hasNext()) {                          // berhenti dengan rapi di akhir
masukan
            try {
                currentVal = s.nextInt();
                if (currentVal == 99) {
                    break;
                }
                System.out.println("Adding " + currentVal);
                intList.add(currentVal);
            } catch (InputMismatchException ime) {
                String bad = s.next();                 // buang token yang bermasalah
                System.out.println(ime + ": \"" + bad + "\" skipped, please enter only
numbers.");
            }
        }
        System.out.println("Exiting: List is " + intList);
    }
}
```

Dengan masukan ini, yang memuat dua token buruk,

```
1
2
b
3
ten
99
```

keluarannya adalah:

```
Enter some numbers!
Adding 1
Adding 2
java.util.InputMismatchException: "b" skipped, please enter only numbers.
Adding 3
java.util.InputMismatchException: "ten" skipped, please enter only numbers.
Exiting: List is [1, 2, 3]
```

Tiga rincian dalam program itu layak dipelajari.

Pertama, baris terpenting di dalam blok `catch` adalah `String bad = s.next();`. Ketika `nextInt()` gagal, ia *tidak* menghabiskan token yang buruk itu; `b` masih bertengger di depan masukan. Jika kita hanya mencetak pesan lalu mengulang, `nextInt()` berikutnya akan bertemu `b` yang sama, gagal lagi, dan program akan mencetak pesan yang sama selamanya. Memanggil `s.next()` membaca token penyebab masalah itu sebagai string biasa lalu membuangnya, sehingga perulangan dapat bergerak maju. Percobaan pertama yang populer adalah membuat `Scanner` yang benar-benar baru di dalam blok `catch`. Cara itu bisa tampak berhasil ketika Anda mengetik di papan ketik, tetapi rapuh: `Scanner` yang lama mungkin sudah membaca lebih dahulu dan menyangga masukan yang tidak akan pernah dilihat oleh `Scanner` yang baru, dan dengan masukan yang dialihkan dari berkas, `Scanner` yang baru sering tidak menemukan apa-apa sama sekali. Soal Latihan 5.12 meminta Anda menjelaskan mengapa. Membuang token itulah perbaikan yang andal.

Kedua, `ime` adalah objek eksepsinya, dan mencetaknya dengan `+` memanggil metode `toString()`-nya, yang memberikan nama kelasnya. Objek eksepsi juga mempunyai `getMessage()`, yang hanya mengembalikan pesannya (untuk `InputMismatchException` dari `nextInt` tidak ada pesan sama sekali, `getMessage()` mengembalikan `null`, itulah sebabnya kita sendiri yang mencetak tokennya).

Ketiga, perulangannya adalah `while (s.hasNext())`, bukan `while (true)`. `hasNext()` mengembalikan `false` ketika masukan sudah habis, sehingga program juga berakhir dengan anggun jika masukan berhenti sebelum `99`; tanpa itu, `nextInt()` di akhir masukan akan melempar `NoSuchElementException`, yang tidak disebut oleh `catch` kita.

Kode 5.24 adalah versi Python-nya. `for line in sys.stdin` membaca masukan standar satu baris setiap kali dan berhenti dengan rapi di ujungnya, yang menyelesaikan butir ketiga tanpa pengujian tambahan apa pun.

Kode 5.24 `safe_input.py`: menangkap `ValueError` dari `int()` di Python.

```
import sys

int_list = []
print("Enter some numbers!")
for line in sys.stdin:                          # berhenti dengan rapi di akhir masukan
    text = line.strip()
    try:
        current_val = int(text)
        if current_val == 99:
            break
        print("Adding " + str(current_val))
        int_list.append(current_val)
    except ValueError as ve:
        print(type(ve).__name__ + ": " + repr(text) + " skipped, please enter only
numbers.")
print("Exiting: List is " + str(int_list))
```

```
Enter some numbers!
Adding 1
Adding 2
ValueError: 'b' skipped, please enter only numbers.
Adding 3
ValueError: 'ten' skipped, please enter only numbers.
Exiting: List is [1, 2, 3]
```

Karena Python membaca baris utuh lalu mengonversinya sesudahnya, teks yang buruk sudah terhabiskan ketika `int()` gagal, dan tidak ada yang perlu dibuang. `type(ve).__name__` adalah nama kelas eksepsinya, `ValueError`, dan `repr(text)` menampilkan teks itu beserta tanda kutipnya.

5.5.5 Beberapa klausa `catch`: jaring yang makin lama makin besar

Sebuah `try` boleh diikuti oleh beberapa klausa `catch`, satu untuk setiap jenis masalah. Ketika eksepsi dilemparkan, Java mencoba klausa-klausa itu secara berurutan dari atas, lalu menjalankan klausa *pertama* yang tipenya cocok; sisanya dilewati.

```
try {
    // kode yang dapat menimbulkan eksepsi
} catch (SomeExceptionType et) {
    // apa yang dilakukan bila eksepsi itu terjadi
} catch (AnotherExceptionType et2) {
    // lakukan hal lain
} catch (Exception e) {
    // ini akan menangkap semua eksepsi lainnya
}
```

Kita sebaiknya memandang beberapa klausa `catch` sebagai serangkaian jaring yang makin lama makin besar, masing-masing menangkap rentang eksepsi yang lebih luas, dengan jaring pertama sebagai yang paling spesifik. **Kode 5.25** mencoba membagi 100 dengan bilangan-bilangan yang diambil dari larik string, pada empat posisi, dan menjumpai tiga masalah berbeda di sepanjang jalan.

Kode 5.25 `CatchOrder.java`: tiga klausa `catch`, dari yang paling spesifik sampai yang paling umum.

```
public class CatchOrder {
    public static void main(String[] args) {
        String[] tokens = {"12", "x7", "0"};
        int[] positions = {0, 1, 2, 5};
        for (int pos : positions) {
            try {
                int n = Integer.parseInt(tokens[pos]);
                System.out.println("position " + pos + ": 100 / " + n + " = " + (100 / n));
            } catch (NumberFormatException nfe) {           // jaring yang paling spesifik
lebih dahulu
                System.out.println("position " + pos + ": not a number ("
                        + nfe.getMessage() + ")");
            } catch (ArrayIndexOutOfBoundsException aioobe) {
                System.out.println("position " + pos + ": no such position ("
                        + aioobe.getMessage() + ")");
            } catch (Exception e) {                         // jaring yang paling lebar
terakhir
                System.out.println("position " + pos + ": something else went wrong: " +
e);
            }
        }
        System.out.println("All positions processed.");
    }
}
```

```
position 0: 100 / 12 = 8
position 1: not a number (For input string: "x7")
position 2: something else went wrong: java.lang.ArithmeticException: / by zero
position 5: no such position (Index 5 out of bounds for length 3)
All positions processed.
```

Posisi 0 berhasil. Pada posisi 1, `Integer.parseInt("x7")` melempar `NumberFormatException`, dan jaring pertama menangkapnya. Pada posisi 2, teks `"0"` berhasil diurai, tetapi `100 / 0` melempar `ArithmeticException`; kedua jaring pertama tidak cocok, sehingga jaring terlebar, `Exception`, yang menangkapnya. Pada posisi 5, larik itu hanya mempunyai tiga elemen, dan jaring kedua menangkap `ArrayIndexOutOfBoundsException` bahkan sebelum `parseInt` sempat dipanggil. Sesudah setiap masalah, perulangan tinggal berlanjut ke posisi berikutnya.

Urutan jaring bukanlah soal selera. Jika kita menaruh `catch (Exception e)` paling depan, klausa itu akan menangkap segalanya, dan klausa-klausa yang lebih spesifik di bawahnya tidak akan pernah bisa berjalan. Java menganggap hal itu sebagai kesalahan dan menolak mengompilasinya, dengan pesan “exception NumberFormatException has already been caught” (eksepsi NumberFormatException sudah ditangkap); Soal Latihan 5.9 memperlihatkannya.

Bila dua jenis eksepsi memerlukan penanganan yang persis sama, Java mengizinkan satu klausa menyebut keduanya, dipisahkan dengan garis tegak: `catch (NumberFormatException | ArithmeticException e)`. Ini disebut *multi-catch*.

Kode 5.26 adalah versi Python-nya. Python memeriksa klausa-klausa `except`-nya secara berurutan dengan cara yang sama, dan menangkap beberapa tipe dalam satu klausa dengan sebuah *tuple*: `except (ValueError, ZeroDivisionError) as e:`.

Kode 5.26 `catch_order.py`: tiga klausa `except`, dari yang paling spesifik sampai yang paling umum.

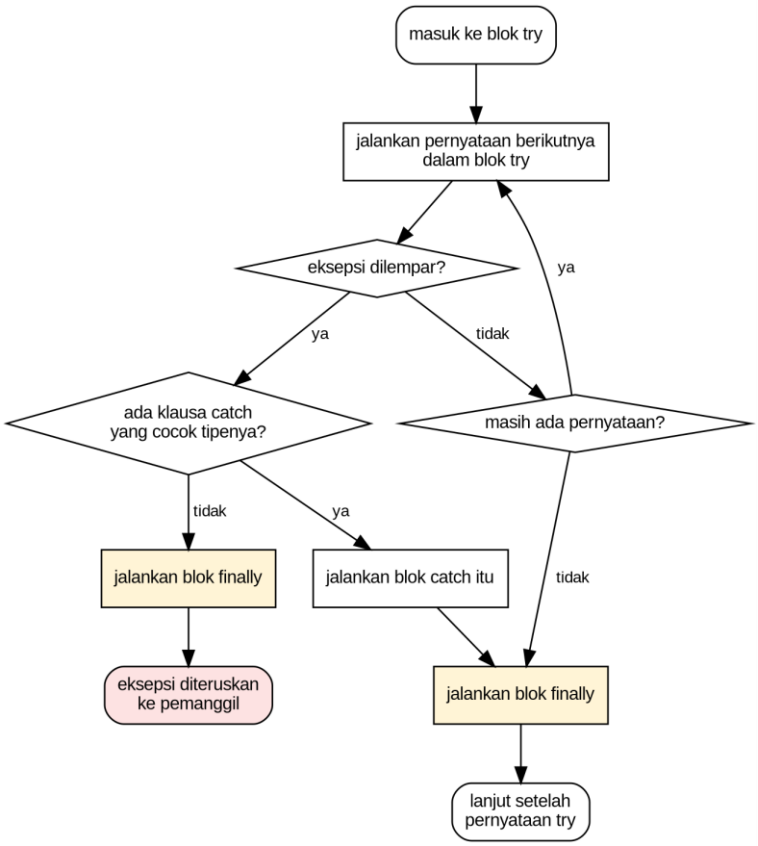
```
tokens = ["12", "x7", "0"]
positions = [0, 1, 2, 5]
for pos in positions:
    try:
        n = int(tokens[pos])
        print("position " + str(pos) + ": 100 / " + str(n) + " = " + str(100 // n))
    except ValueError as ve:                # jaring yang paling spesifik lebih dahulu
        print("position " + str(pos) + ": not a number (" + str(ve) + ")")
    except IndexError as ie:
        print("position " + str(pos) + ": no such position (" + str(ie) + ")")
    except Exception as e:                  # jaring yang paling lebar terakhir
        print("position " + str(pos) + ": something else went wrong: " + repr(e))
print("All positions processed.")
```

```
position 0: 100 / 12 = 8
position 1: not a number (invalid literal for int() with base 10: 'x7')
position 2: something else went wrong: ZeroDivisionError('integer division or modulo by
zero')
position 5: no such position (list index out of range)
All positions processed.
```

> **JEBAKAN:** Jaring berlabel `Exception` menangkap masalah-masalah yang tidak pernah Anda pikirkan, termasuk *bug* dalam kode Anda sendiri, sehingga dapat menyembunyikannya. Pakailah hanya sebagai klausa terakhir, dan hanya di tempat yang memang tepat untuk melanjutkan, seperti dalam perulangan yang memproses banyak butir yang saling lepas. Jangan pernah menulis blok `catch` yang *kosong*: blok seperti itu membuang satu-satunya informasi yang sebenarnya akan memberi tahu Anda apa yang salah.

5.5.6 `finally`: blok yang selalu dijalankan

Terkadang ada pekerjaan yang harus dilakukan, entah terjadi eksepsi entah tidak: menutup berkas, melepaskan sambungan jaringan, mematikan pemanas sebuah mesin. Untuk itu, pernyataan `try` boleh diakhiri dengan blok `finally`. Kode di dalam `finally` dijalankan dalam setiap kasus: sesudah blok `try` selesai secara normal, sesudah blok `catch` menangani eksepsi, bahkan ketika eksepsi *tidak* ditangkap dan hendak berpropagasi lebih lanjut, atau ketika blok `try` mengeksekusi `return`. **Gambar 5.6** menelusuri semua jalur itu.



Gambar 5.6 Jalur-jalur yang melewati pernyataan `try`–`catch`–`finally`. Setiap jalur melewati blok `finally`.

Kode 5.27 memperagakan dua di antara jalur-jalur itu dengan metode `divide` yang mempunyai `try` dan `finally` tetapi tidak mempunyai `catch`.

Kode 5.27 `FinallyDemo.java`: `finally` dijalankan sesudah `return` yang normal dan ketika sebuah eksepsi lolos.

```
public class FinallyDemo {
    static int divide(int a, int b) {
        try {
            System.out.println("  try: dividing " + a + " by " + b);
            return a / b;
        } finally {
            System.out.println("  finally: this line always runs");
        }
    }

    public static void main(String[] args) {
        System.out.println("Case 1: no exception");
        System.out.println("  result = " + divide(10, 2));

        System.out.println("Case 2: an exception, caught by the caller");
        try {
            System.out.println("  result = " + divide(1, 0));
        } catch (ArithmeticException e) {
            System.out.println("  caught in main: " + e.getMessage());
        }
        System.out.println("Done.");
    }
}
```

```
Case 1: no exception
  try: dividing 10 by 2
  finally: this line always runs
  result = 5
Case 2: an exception, caught by the caller
  try: dividing 1 by 0
  finally: this line always runs
  caught in main: / by zero
Done.
```

Pada kasus 1, blok `try` mencapai `return a / b;`. Nilai 5 sudah dihitung dan hendak dikembalikan, tetapi sebelum metode itu benar-benar menyerahkannya, blok `finally` dijalankan dan mencetak barisnya; baru sesudah itu `main` menerima 5 dan mencetak `result = 5`. Pada kasus 2, `1 / 0` melempar `ArithmeticException`. Tidak ada `catch` di `divide`, sehingga eksepsi akan berpropagasi ke `main`, tetapi sekali lagi, blok `finally` dijalankan lebih dahulu, dalam perjalanan keluar. Kemudian `catch` di `main` menangani eksepsinya. Perhatikan bahwa `result =` tidak pernah tercetak pada kasus 2: `println` di `main` menunggu nilai yang tidak pernah datang.

`try` di Python mempunyai `finally` yang sama, ditambah satu klausa opsional lagi, `else`, yang hanya dijalankan ketika blok `try` tidak melempar apa-apa. **Kode 5.28** menambahkan kasus ketiga untuk memperlihatkannya.

Kode 5.28 `finally_demo.py`: `finally` di Python, dan klausa `else`.

```python
def divide(a, b):
    try:
        print("  try: dividing " + str(a) + " by " + str(b))
        return a // b
    finally:
        print("  finally: this line always runs")


def main():
    print("Case 1: no exception")
    print("  result = " + str(divide(10, 2)))

    print("Case 2: an exception, caught by the caller")
    try:
        print("  result = " + str(divide(1, 0)))
    except ZeroDivisionError as e:
        print("  caught in main: " + str(e))

    print("Case 3: Python's else clause")
    try:
        result = divide(9, 3)
    except ZeroDivisionError:
        print("  never printed here")
    else:
        print("  else: runs only when the try block raised nothing, result = " +
str(result))
    print("Done.")


if __name__ == "__main__":
    main()
```

```
Case 1: no exception
  try: dividing 10 by 2
  finally: this line always runs
  result = 5
Case 2: an exception, caught by the caller
  try: dividing 1 by 0
  finally: this line always runs
  caught in main: integer division or modulo by zero
Case 3: Python's else clause
  try: dividing 9 by 3
  finally: this line always runs
  else: runs only when the try block raised nothing, result = 3
Done.
```

Urutan dalam bentuk lengkap Python adalah `try`, lalu klausa-klausa `except` (jika ada), lalu `else`, lalu `finally`. Menaruh kode di dalam `else` alih-alih di ujung `try` adalah cara untuk menyatakan "hanya bagian kode inilah yang diharapkan melempar eksepsi-eksepsi yang saya tangkap", sehingga jaring-jaringnya tidak menangkap lebih banyak daripada semestinya.

5.5.7 Melempar eksepsi sendiri

Eksepsi tidak hanya dilemparkan oleh pustaka. Terkadang kita sendiri ingin memaksakan terjadinya eksepsi, ketika pemanggil melakukan sesuatu yang ingin kita larang. Untuk itu, Java mempunyai kata kunci `throw` dan Python kata kunci `raise`, masing-masing diikuti oleh objek eksepsi yang baru:

```
throw new IllegalArgumentException("The numbers you've entered are not acceptable.");
raise ValueError("The numbers you've entered are not acceptable.")
```

`IllegalArgumentException` sangat lazim di Java di mana pun sebuah parameter harus berada dalam batas-batas tertentu; padanannya di Python adalah `ValueError`. Contoh klasiknya adalah nilai *grayscale* (skala keabuan), yaitu tingkat kecerahan satu piksel dalam citra hitam-putih, yang harus berada di antara 0 dan 255. **Kode 5.29** mempunyai metode yang melempar eksepsi untuk nilai yang melebihi 255, dan sebuah `main` yang menangkapnya lalu, alih-alih berhenti di tengah jalan, mengatur ulang nilainya menjadi 255 dan terus berjalan.

Kode 5.29 `IllegalArgumentExceptionExample.java`: melempar `IllegalArgumentException` untuk nilai *grayscale* di luar rentang, lalu memulihkan diri dengan memangkasnya menjadi 255.

```java
public class IllegalArgumentExceptionExample {
    public static boolean isExceeded(int x) {
        boolean result = false;
        if (x > 255) {
            throw new IllegalArgumentException("Value exceeds 255");
        } else {
            result = true;
        }
        return result;
    }

    public static void main(String[] args) {
        int value = 270;
        try {
            // Anggaplah kita menangani nilai Grayscale: 0-255
            // Periksa rentang nilainya
            if (isExceeded(value)) {
                System.out.println(value + " is a valid Grayscale value");
            }
        } catch (IllegalArgumentException ie) {
            System.out.println(value + " is an invalid Grayscale value");
            System.out.println("Resetting the value to 255");
            value = 255;
        }
        System.out.println("Continue processing the value...");
    }
}
```

```
270 is an invalid Grayscale value
Resetting the value to 255
Continue processing the value...
```

Di `main` nilainya 270. `isExceeded(270)` mendapati bahwa 270 > 255 lalu melempar eksepsi; `if` itu tidak pernah selesai, dan kendali melompat ke blok `catch`, yang mencetak dua baris dan mengatur nilainya menjadi 255. Kemudian eksekusi berlanjut di bawah pernyataan `try`. Sering kali kita cukup mencetak bahwa telah terjadi kesalahan lalu membiarkan pengguna melanjutkan di bawah blok `try`; di sini kita melakukan sesuatu yang sedikit lebih cerdas dan memperbaiki nilainya.

Program ini sengaja dibiarkan persis seperti yang mungkin ditulis pada versi pertama, dan di dalamnya terkandung pelajaran tentang penamaan. Metode bernama `isExceeded` akan diharapkan mengembalikan `true` ketika nilainya *memang* terlampaui; metode ini justru mengembalikan `true` ketika nilainya baik dan melempar eksepsi ketika nilainya terlampaui. Programnya berjalan, tetapi pembaca berikutnya akan salah membacanya. Nama yang lebih baik adalah `checkGrayscale`, yang tidak mengembalikan apa-apa dan cukup melempar eksepsi untuk masukan yang buruk; Subbab 5.11 akan kembali membahas hal ini.

Kode 5.30 adalah versi Python-nya, yang melempar `ValueError`.

Kode 5.30 `grayscale.py`: melempar `ValueError` untuk nilai *grayscale* di luar rentang.

```
def is_exceeded(x):
    result = False
    if x > 255:
        raise ValueError("Value exceeds 255")
    else:
        result = True
    return result


def main():
    value = 270
    try:
        # Anggaplah kita menangani nilai Grayscale: 0-255
        # Periksa rentang nilainya
        if is_exceeded(value):
            print(str(value) + " is a valid Grayscale value")
    except ValueError:
        print(str(value) + " is an invalid Grayscale value")
        print("Resetting the value to 255")
        value = 255
    print("Continue processing the value...")


if __name__ == "__main__":
    main()
```

```
270 is an invalid Grayscale value
Resetting the value to 255
Continue processing the value...
```

Dua catatan prinsip melengkapi subbab ini. Pertama, jika kita melempar eksepsi, eksepsi itu harus ditangkap di suatu tempat, atau program akan macet karenanya. Kedua, merupakan praktik yang baik untuk melempar eksepsi hanya sebagai isyarat bahwa telah terjadi sesuatu yang keliru. Eksepsi tidak semestinya dipakai sebagai cara cerdik untuk keluar dari perulangan atau untuk mengoper nilai-nilai biasa: eksepsi lebih lambat daripada alur kendali biasa dan, lebih buruk lagi, membingungkan setiap pembaca yang mengira eksepsi berarti kesalahan [2].

5.6 Eksepsi Terperiksa dan Tak Terperiksa

Silsilah pada **Gambar 5.4** memunculkan satu hal yang menarik. Ada dua macam eksepsi di Java: eksepsi yang dapat terjadi hampir di mana saja, sehingga sebenarnya tidak dapat direncanakan oleh kompilator, dan eksepsi yang *diketahui* Java dapat terjadi di tempat tertentu, sehingga Java memaksa kita untuk berbuat sesuatu terhadapnya.

5.6.1 Eksepsi tak terperiksa: biasanya kesalahan kita sendiri

Kita sudah bertemu beberapa eksepsi. Masih ingat? **Kode 5.31** memancing tiga eksepsi yang paling terkenal, masing-masing di dalam `try`-nya sendiri agar kita dapat melihat ketiga pesannya.

Kode 5.31 `UncheckedDemo.java`: tiga eksepsi tak terperiksa yang disebabkan oleh kesalahan pemrograman.

```java
public class UncheckedDemo {
    static String[] arr = new String[6];

    public static void main(String[] args) {
        try {
            System.out.println(arr[6]);                  // Apa yang terjadi di sini?
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("1: " + e);
        }
        try {
            System.out.println(arr[3].length());         // Bagaimana dengan di sini?
        } catch (NullPointerException e) {
            System.out.println("2: " + e);
        }
        Object x = Integer.valueOf(0);
        try {
            System.out.println((String) x);              // Bagaimana kalau sekarang?
        } catch (ClassCastException e) {
            System.out.println("3: " + e);
        }
    }
}
```

```
1: java.lang.ArrayIndexOutOfBoundsException: Index 6 out of bounds for length 6
2: java.lang.NullPointerException: Cannot invoke "String.length()" because
"UncheckedDemo.arr[3]" is null
3: java.lang.ClassCastException: class java.lang.Integer cannot be cast to class
java.lang.String (java.lang.Integer and java.lang.String are in module java.base of loader
'bootstrap')
```

Yang pertama sudah tidak asing: larik sepanjang 6 tidak mempunyai indeks 6. Yang kedua adalah `NullPointerException` yang tersohor: `arr[3]` masih memuat nilai bawaannya, `null`, tidak ada string di sana, sehingga tidak ada apa pun yang dapat dipanggil `length()`-nya. Java modern menjelaskan hal ini dengan sangat membantu: `Cannot invoke "String.length()" because "UncheckedDemo.arr[3]" is null` (tidak dapat memanggil `String.length()` karena `UncheckedDemo.arr[3]` bernilai `null`). Yang ketiga adalah `ClassCastException`: variabel `x` bertipe `Object`, yang boleh merujuk ke apa saja, dan ia merujuk ke sebuah `Integer`; *cast* `(String) x` mengklaim bahwa isinya `String`, dan Java memeriksa klaim itu pada saat program berjalan lalu mendapatinya keliru.

Ketiganya adalah *unchecked exception* (eksepsi tak terperiksa). Semuanya memperluas `RuntimeException`, dan kompilator tidak memaksa kita menangkapnya. Ada dua alasannya. Pertama, kompilator biasanya tidak dapat mengetahui apakah eksepsi itu akan terjadi: hampir setiap ekspresi dengan indeks larik, tanda titik, atau *cast* dapat melemparnya, dan menuntut `try` di sekeliling setiap ekspresi semacam itu akan membuat program tidak terbaca. Kedua, dan ini lebih penting, eksepsi tak terperiksa umumnya disebabkan oleh kesalahan pemrogram. Hal itu seharusnya jelas dari ketiga contoh tadi: pada masing-masing, programlah, bukan pengguna, yang melakukan sesuatu yang tidak akan pernah bisa berhasil.

> **WAWASAN KUNCI:** Jika kode kita sendiri menghasilkan eksepsi tak terperiksa, biasanya itu kesalahan kita, dan tanggapan yang tepat adalah membetulkan kodenya: periksa indeksnya, inisialisasi variabelnya, uji tipenya. Sekadar menangkap eksepsi-eksepsi ini mungkin sama sekali tidak menyentuh akar masalahnya.

Python mempunyai tiga kesalahan yang sama, dengan nama-nama yang berbeda, seperti ditunjukkan **Kode 5.32**. Memakai indeks yang terlalu besar melempar `IndexError`; menanyakan panjang `None` melempar `TypeError`; meminta sebuah `int` menjalankan metode yang tidak dimilikinya melempar `AttributeError`. (Python tidak mengenal *cast*, sehingga padanan terdekat

`ClassCastException` adalah memakai sebuah nilai seakan-akan bertipe sesuatu yang bukan tipenya.)

Kode 5.32 `unchecked_demo.py`: tiga jenis kesalahan yang sama di Python.

```
arr = [None] * 6

try:
    print(arr[6])                           # Apa yang terjadi di sini?
except IndexError as e:
    print("1:", repr(e))
try:
    print(len(arr[3]))                      # Bagaimana dengan di sini?
except TypeError as e:
    print("2:", repr(e))
x = 0
try:
    print(x.upper())                        # Bagaimana kalau sekarang?
except AttributeError as e:
    print("3:", repr(e))
```

```
1: IndexError('list index out of range')
2: TypeError("object of type 'NoneType' has no len()")
3: AttributeError("'int' object has no attribute 'upper'")
```

5.6.2 Eksepsi terperiksa: kompilator bersikeras

Eksepsi terperiksa berbeda. Hampir segala sesuatu di dalam paket `java.io` dapat melemparnya, dan eksepsi-eksepsi ini mewakili masalah yang bukan kesalahan pemrograman siapa pun: kita semestinya dapat melihat potensi eksepsi yang jelas ketika mencoba membaca dari sebuah berkas. Bagaimana jika berkasnya tidak ada? Jelas Java tidak dapat membacanya, sehingga ia harus melempar eksepsi. Bagaimana jika disk penuh, atau berkas itu milik pengguna lain, atau *drive* jaringan baru saja terputus? Secermat apa pun pemrogramannya, hal-hal ini tidak dapat dicegah; program yang benar hanya dapat *bersiap* menghadapinya.

Untuk eksepsi semacam itu Java mewajibkan persiapan tersebut. Eksepsi-eksepsi itu memperluas `Exception` tetapi tidak memperluas `RuntimeException` (kotak-kotak biru pada **Gambar 5.4**), dan disebut *checked exception* (eksepsi terperiksa) karena kompilator memeriksa bahwa masing-masing sudah ditangani. Jika kita mencoba mengompilasi kode berikut,

```
import java.io.*;

public class Test {
    public static void main(String[] args) {
        BufferedReader br = new BufferedReader(new FileReader("filename.txt"));
    }
}
```

kompilasinya gagal. Kompilator memberi tahu kita:

```
Test.java:5: error: unreported exception FileNotFoundException; must be caught or declared
to be thrown
        BufferedReader br = new BufferedReader(new FileReader("filename.txt"));
                                               ^
1 error
```

Bunyi kalimat itu patut dibaca dengan cermat. Konstruktor `FileReader(String)` dideklarasikan sebagai konstruktor yang dapat melempar `FileNotFoundException`. Karena itu kode kita harus melakukan salah satu dari dua hal berikut:

1. **menangkapnya**: mengelilingi panggilan itu dengan `try` dan sebuah `catch` yang menangani `FileNotFoundException` (atau salah satu leluhurnya); atau
2. **mendeklarasikannya**: menyatakan, pada kepala metode, bahwa metode ini dengan sengaja tidak menangani eksepsi itu melainkan meneruskannya kepada siapa pun yang memanggil metode tersebut. Deklarasinya memakai kata kunci `throws`: `static String firstLine(String fileName) throws IOException`.

Membiarkan `main` sendiri mendeklarasikan `throws FileNotFoundException` memang dapat dikompilasi, tetapi itu gagasan yang sangat buruk: `main` tidak mempunyai pemanggil di dalam program kita, sehingga eksepsi itu hanya akan membuat program macet dengan *stack trace*, dan itulah persis yang sedang kita coba hindari. Sebagai gantinya, kita hendaknya menangani kemungkinan bahwa berkasnya tidak ada. **Kode 5.33** melakukan kedua hal itu, masing-masing di tempat yang tepat.

Kode 5.33 `OpenFile.java`: `firstLine` mendeklarasikan `IOException` dengan `throws`; `main` menangkapnya.

```
import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;

public class OpenFile {
    // Mendeklarasikan eksepsi terperiksa alih-alih menanganinya: pemanggil yang
memutuskan.
    static String firstLine(String fileName) throws IOException {
        BufferedReader br = new BufferedReader(new FileReader(fileName));
        String line = br.readLine();
        br.close();
        return line;
    }

    public static void main(String[] args) {
        String[] names = {"filename.txt", "notes.txt"};
        for (String name : names) {
            try {
                System.out.println(name + " begins: " + firstLine(name));
            } catch (FileNotFoundException fnfe) {
                System.out.println("File not found: " + fnfe.getMessage());
            } catch (IOException ioe) {
                System.out.println("Miraculously, some other IO exception: "
                        + ioe.getMessage());
            }
        }
    }
}
```

```
File not found: filename.txt (No such file or directory)
notes.txt begins: Profitina Laundry opens at 07:00.
```

`firstLine` adalah metode pembantu kecil yang membuka berkas, membaca baris pertamanya, lalu menutupnya. Metode itu bukan tempat yang tepat untuk memutuskan apa yang harus dilakukan terhadap berkas yang tidak ada: dalam satu program hal itu berarti "cetak pesan yang ramah", dalam program lain "pakai pengaturan bawaan", dalam program ketiga "minta pengguna memberikan nama lain". Karena itu `firstLine` *mendeklarasikan* `throws IOException` dan menyerahkan keputusan kepada pemanggilnya. `main` tahu apa yang diinginkannya, yaitu sebuah pesan, sehingga `main`-lah yang *menangkap*. Berkas pertama tidak ada, dan kita melihat pesan `File not found: filename.txt (No such file or directory)`, dengan bagian di dalam tanda kurung berasal dari sistem operasi. Berkas kedua, `notes.txt`, ada di folder yang sama:

```
Profitina Laundry opens at 07:00.
Express orders are ready in six hours.
The courier leaves at 16:00.
```

dan baris pertamanya tercetak.

Perhatikan bahwa kita tidak dapat hanya menulis `catch (FileNotFoundException fnfe)` saja: `readLine()` dan `close()` dapat melempar `IOException` yang lebih umum, dan kompilator bersikeras agar eksepsi itu juga ditangani. Karena itulah ada klausa kedua, yang pesannya ("Miraculously, some other IO exception", ajaibnya, ada eksepsi IO lain) mengakui bahwa kita tidak menduganya. `FileNotFoundException` memperluas `IOException`, sehingga jaring yang spesifik harus didahulukan.

Ada pula pelajaran dari hal yang *tidak* dikerjakan kode ini dengan baik. Jika `readLine()` melempar eksepsi, `br.close()` akan terlewati dan berkasnya tetap terbuka. Subbab 5.8.2 menunjukkan cara menjamin bahwa berkas pasti ditutup.

Kesalahan terkait yang sangat lazim adalah menulis blok `try` hanya di sekeliling pembukaan berkas, lalu mencoba membaca dari *reader*-nya *sesudah* `try`:

```
try {
    BufferedReader br = new BufferedReader(new FileReader("filename.txt"));
} catch (FileNotFoundException fnfe) {
    System.out.println("File not found: " + fnfe.getMessage());
}
// membaca dari br di sini?  Tidak bisa: br tidak ada di luar blok try.
```

Kode ini dapat dikompilasi, tetapi tidak berguna, karena dua alasan. Variabel `br` dideklarasikan di dalam blok `try` dan tidak ada lagi sesudah kurung kurawal penutupnya; dan seandainya pun masih ada, ketika berkasnya tidak ditemukan program akan mencetak pesannya lalu melanjutkan dengan memakai *reader* yang tidak pernah dibuat. Pembacaannya harus berada *di dalam* `try`, atau di dalam metode yang mendeklarasikan eksepsinya, persis seperti yang dilakukan **Kode 5.33**.

5.6.3 Tangkap, atau deklarasikan?

Dengan eksepsi terperiksa Anda selalu mempunyai dua pilihan dari Subbab 5.6.2, dan memilih dengan baik adalah sebuah keterampilan. Pegangan praktis yang baik:

- **Tangkaplah** eksepsi di metode yang tahu *apa yang harus dilakukan* terhadapnya: menampilkan pesan, mencoba berkas lain, memakai nilai bawaan, melewati rekaman yang buruk.
- **Deklarasikanlah** eksepsi di metode-metode yang tidak tahu. Metode yang membuka dan membaca berkas atas nama pihak lain biasanya tidak dapat memutuskan apa arti berkas yang hilang, sehingga ia mendeklarasikan `throws IOException` dan membiarkan eksepsinya berpropagasi ke pemanggil yang mampu memutuskan.

Kengototan kompilator bukanlah birokrasi. Ketika Anda memanggil metode yang kepalanya berbunyi `throws IOException`, kompilator memastikan Anda tidak bisa lupa bahwa panggilan itu dapat gagal. Justru itulah sebabnya para perancang Java menjadikan eksepsi I/O sebagai eksepsi terperiksa: kegagalan di dunia luar adalah hal yang normal, dan program yang mengabaikannya adalah program yang salah [3].

5.6.4 Sikap Python

Python tidak mengenal eksepsi terperiksa. Setiap eksepsi berperilaku seperti eksepsi tak terperiksa di Java: interpreter tidak pernah memaksa pemanggil menangkap apa pun, dan tidak ada klausa `throws`. **Kode 5.34** adalah versi Python dari **Kode 5.33**.

Kode 5.34 `open_file.py`: `first_line` mendokumentasikan eksepsinya di dalam *docstring*; `main` menangkapnya.

```
def first_line(file_name):
    """Mengembalikan baris pertama sebuah berkas teks.

    Raises:
        OSError: jika berkas tidak dapat dibuka atau dibaca (FileNotFoundError salah satu
jenisnya).
    """
    f = open(file_name, encoding="utf-8")
    line = f.readline().rstrip("\n")
    f.close()
    return line


def main():
    names = ["filename.txt", "notes.txt"]
    for name in names:
        try:
            print(name + " begins: " + first_line(name))
        except FileNotFoundError as fnfe:
            print("File not found: " + str(fnfe))
        except OSError as ose:
            print("Miraculously, some other OS error: " + str(ose))


if __name__ == "__main__":
    main()
```

```
File not found: [Errno 2] No such file or directory: 'filename.txt'
notes.txt begins: Profitina Laundry opens at 07:00.
```

Karena bahasanya tidak akan mengingatkan pemanggil, para pemrogram Python menyatakan di dalam dokumentasi eksepsi apa saja yang dapat dilempar sebuah fungsi; tempat yang lazim adalah bagian `Raises:` di dalam *docstring* fungsi itu, seperti pada `first_line`. Bab 6 membahas *docstring* secara layak. Disiplinnya sama dengan di Java; hanya pemaksaannya yang tidak ada, sehingga semuanya bergantung pada Anda.

Pendekatan mana yang lebih baik adalah perdebatan yang setua Java itu sendiri. Eksepsi terperiksa membuat kegagalan terlihat dan sukar diabaikan; para pengkritiknya mengatakan eksepsi terperiksa membuat kode berantakan dan menggoda pemrogram menelan eksepsi dengan blok `catch` kosong sekadar untuk membungkam kompilator. Kedua kubu sepakat tentang hal-hal pokoknya, dan itulah yang diajarkan bab ini: ketahuilah operasi mana yang dapat gagal, tangani setiap kegagalan di tempat yang tahu apa yang harus dilakukan, dan jangan pernah menyembunyikannya [2].

5.7 Eksepsi Buatan Sendiri

Eksepsi-eksepsi pustaka menguraikan masalah-masalah pustaka. Program binatu mempunyai masalahnya sendiri: berat yang bukan bilangan, berat −1 kilogram, pesanan untuk layanan yang tidak disediakan binatu itu. Kita bisa saja melaporkan semuanya dengan `IllegalArgumentException` atau `ValueError`, tetapi *custom exception class* (kelas eksepsi buatan sendiri) mempunyai tiga keunggulan. Namanya menyatakan dengan tepat apa yang salah; ia dapat membawa informasi tambahan, seperti teks yang buruk atau nomor baris di dalam berkas; dan pemanggil dapat menangkapnya secara khusus, tanpa ikut menangkap masalah-masalah lain yang tidak berkaitan.

Menulisnya cukup singkat. Di Java kita menulis sebuah kelas yang memperluas `Exception`, untuk eksepsi terperiksa, atau `RuntimeException`, untuk eksepsi tak terperiksa, lalu menyerahkan pesannya kepada kelas induk dengan `super(message)`. Kita memilih *terperiksa* bila kegagalannya merupakan hasil yang dapat diduga dan harus dipersiapkan oleh pemanggil, seperti data buruk dari luar program, dan *tak terperiksa* bila kegagalannya menandakan kesalahan pemrograman. Berat tidak sah yang diketik pengguna termasuk jenis yang pertama. **Kode 5.35** mendefinisikan `InvalidWeightException` dan memakainya.

Kode 5.35 `WeightCheck.java`: eksepsi terperiksa buatan sendiri yang membawa teks penyebab masalah dan, bila ada, penyebab (*cause*) aslinya.

```
public class WeightCheck {
    // Eksepsi terperiksa buatan kita sendiri, bersarang di sini agar berkasnya tetap utuh.
    static class InvalidWeightException extends Exception {
        private final String weightText;

        InvalidWeightException(String weightText, String message, Throwable cause) {
            super(message, cause);
            this.weightText = weightText;
        }

        String getWeightText() {
            return weightText;
        }
    }

    static double parseWeight(String text) throws InvalidWeightException {
        double kg;
        try {
            kg = Double.parseDouble(text);
        } catch (NumberFormatException nfe) {
            throw new InvalidWeightException(text, "not a number: " + text, nfe);
        }
        if (kg <= 0 || kg > 50) {
            throw new InvalidWeightException(text, "out of range 0-50 kg: " + text, null);
        }
        return kg;
    }

    public static void main(String[] args) {
        String[] inputs = {"3.5", "abc", "-1", "80", "12"};
        for (String input : inputs) {
            try {
                System.out.println("OK       " + parseWeight(input) + " kg");
            } catch (InvalidWeightException e) {
                System.out.println("REJECTED \"" + e.getWeightText() + "\": " +
e.getMessage());
                if (e.getCause() != null) {
                    System.out.println("         caused by " + e.getCause());
                }
            }
        }
    }
}
```

```
OK       3.5 kg
REJECTED "abc": not a number: abc
         caused by java.lang.NumberFormatException: For input string: "abc"
REJECTED "-1": out of range 0-50 kg: -1
REJECTED "80": out of range 0-50 kg: 80
OK       12.0 kg
```

Beberapa hal yang perlu diperhatikan:

- Agar program tetap dalam satu berkas, `InvalidWeightException` ditulis sebagai *nested class* (kelas bersarang), yaitu kelas yang dideklarasikan di dalam `WeightCheck` dengan pengubah `static`. Kelas itu berperilaku seperti kelas lainnya. Dalam program yang lebih besar, kelas itu lazimnya mempunyai berkasnya sendiri, seperti `OrderFormatException` di Subbab 5.9.
- Konstruktornya memanggil `super(message, cause)`. Kelas `Exception` sudah tahu cara menyimpan pesan dan sebuah *cause* (penyebab), sehingga kita memakainya ulang alih-alih menulis sendiri. Kita menambahkan satu *field* milik kita sendiri, `weightText`, beserta *getter*-nya.
- `parseWeight` mendeklarasikan `throws InvalidWeightException`. Eksepsi ini terperiksa, sehingga kompilator kini memastikan setiap pemanggil `parseWeight` menanganinya, dan `main` memang menanganinya.
- Ketika `Double.parseDouble("abc")` gagal, kita menangkap `NumberFormatException`-nya lalu melempar eksepsi kita sendiri sebagai gantinya, sambil menyerahkan eksepsi asli sebagai *cause*. Ini disebut *exception chaining* (perantaian eksepsi) atau *wrapping* (pembungkusan). Pemanggil melihat satu pesan yang jelas pada tingkat binatu, "not a number: abc", sementara tidak ada yang hilang: `getCause()` tetap mengembalikan eksepsi aslinya, dan *stack trace* akan memperlihatkan keduanya. Menerjemahkan eksepsi tingkat rendah menjadi eksepsi yang bermakna pada tingkat pemanggil adalah kebiasaan yang baik [2].
- Untuk -1 dan 80 tidak ada eksepsi yang mendasarinya: bilangannya baik sebagai bilangan tetapi salah sebagai berat cucian. Kita menyerahkan `null` sebagai *cause*.

Di Python, eksepsi buatan sendiri adalah kelas yang memperluas `Exception` atau salah satu subkelasnya. Karena berat yang tidak sah sungguh-sungguh merupakan sejenis nilai yang buruk, **Kode 5.36** memperluas `ValueError`, sehingga kode yang sudah menangkap `ValueError` akan ikut menangkapnya.

Kode 5.36 `weight_check.py`: kelas eksepsi buatan sendiri di Python, dirantai dengan `raise ... from`.

```
class InvalidWeightError(ValueError):
    """Dilempar bila berat cucian bukan bilangan atau di luar rentang."""

    def __init__(self, weight_text, message):
        super().__init__(message)
        self.weight_text = weight_text


def parse_weight(text):
    try:
        kg = float(text)
    except ValueError as ve:
        raise InvalidWeightError(text, "not a number: " + text) from ve
    if kg <= 0 or kg > 50:
        raise InvalidWeightError(text, "out of range 0-50 kg: " + text)
    return kg


def main():
    inputs = ["3.5", "abc", "-1", "80", "12"]
    for text in inputs:
        try:
            print("OK       " + str(parse_weight(text)) + " kg")
        except InvalidWeightError as e:
            print('REJECTED "' + e.weight_text + '": ' + str(e))
            if e.__cause__ is not None:
                print("         caused by " + repr(e.__cause__))


if __name__ == "__main__":
    main()
```

```
OK       3.5 kg
REJECTED "abc": not a number: abc
         caused by ValueError("could not convert string to float: 'abc'")
REJECTED "-1": out of range 0-50 kg: -1
REJECTED "80": out of range 0-50 kg: 80
OK       12.0 kg
```

`super().__init__(message)` menyerahkan pesannya kepada kelas induk, sehingga `str(e)` mengembalikannya. Perantaiannya memakai sintaksis `raise NewError(...) from original`; sesudahnya, eksepsi asli tersedia sebagai `e.__cause__`, dan *traceback* yang tidak tertangkap akan mencetak keduanya, dihubungkan oleh kalimat "The above exception was the direct cause of the following exception" (eksepsi di atas adalah penyebab langsung eksepsi berikut).

> **WAWASAN KUNCI:** Eksepsi yang baik menjawab tiga pertanyaan bagi siapa pun yang menangkapnya: *jenis* masalahnya apa (kelasnya), *persisnya* apa (pesannya), dan *dengan data* apa (*field* tambahan dan penyebabnya). Rancanglah eksepsi buatan Anda sendiri agar menjawab ketiganya.

5.8 Berkas: *Stream*, *Reader*, dan *Writer*

5.8.1 *Stream* dan *reader*

Sebagian besar masukan dan keluaran di Java dikerjakan dengan *stream* (aliran). *Stream* adalah barisan data yang mengalir dari sebuah sumber, seperti papan ketik, berkas, atau sambungan jaringan, ke dalam program (*input stream*, aliran masukan), atau dari program ke sebuah tujuan (*output stream*, aliran keluaran). `System.in`, yang selama ini kita serahkan kepada `Scanner`, adalah sebuah `InputStream`: aliran bita mentah dari papan ketik.

Bita belumlah teks. `InputStreamReader` membungkus sebuah *input stream* dan mengubah bita-bitanya menjadi karakter:

```
InputStreamReader isr = new InputStreamReader(System.in);
char character = (char) isr.read();   // baca satu karakter
```

Membaca satu karakter setiap kali itu melelahkan, jadi kita membungkusnya sekali lagi. `BufferedReader` membungkus *reader* apa pun, mengumpulkan karakter di dalam sebuah *buffer* (penyangga), yaitu sebongkah memori, lalu menyerahkannya satu baris utuh setiap kali melalui `readLine()`:

```
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
```

Pembungkusan satu objek di dalam objek lain, yang masing-masing menambahkan satu kemampuan, adalah pola yang akan Anda jumpai di seluruh pustaka I/O Java. **Kode 5.37** memakainya untuk membaca baris-baris ke dalam sebuah `ArrayList` sampai pengguna mengetik `stop`.

Kode 5.37 `ReadUntilStop.java`: membaca baris-baris dari papan ketik dengan `BufferedReader` sampai bertemu `stop` atau akhir masukan.

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.ArrayList;

public class ReadUntilStop {
    public static void main(String[] args) {
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        ArrayList<String> stringList = new ArrayList<>();
        try {
            String currentLine = br.readLine();
            while (currentLine != null && !currentLine.equals("stop")) {
                stringList.add(currentLine);
                currentLine = br.readLine();
            }
        } catch (IOException e) {
            System.out.println("Could not read the keyboard: " + e.getMessage());
        }
        // Cetak isi list
        System.out.println("Read " + stringList.size() + " line(s): " + stringList);
    }
}
```

Dengan berkas masukan

```
wash 3 kg
iron 2 shirts
express please
stop
this line is never read
```

keluarannya adalah:

```
Read 3 line(s): [wash 3 kg, iron 2 shirts, express please]
```

Dua perbedaan dari `Scanner` menonjol. Pertama, `readLine()` mengembalikan `null` di akhir masukan alih-alih melempar eksepsi, sehingga perulangan menguji `currentLine != null` sebelum memeriksa isi barisnya. (Urutan pengujian itu penting: jika barisnya `null`, `&&` berhenti di situ dan tidak pernah memanggil `equals` pada `null`, yang akan melempar `NullPointerException`.) Kedua, `readLine()` dapat melempar `IOException` yang terperiksa, sehingga kompilator bersikeras adanya `try` dan `catch`.

Bentuk ringkas perulangan yang sama, yang akan Anda jumpai dalam kode orang lain, membaca dan menguji dalam satu syarat saja:

```
while (!(currentLine = br.readLine()).equals("stop")) {
    stringList.add(currentLine);
}
```

Bentuk itu berhasil, tetapi versi di dalam program tadi lebih mudah dipahami dan juga menangani akhir masukan.

Di Python, `sys.stdin` sudah merupakan aliran teks yang mempunyai *buffer*-nya sendiri, sehingga tidak ada yang perlu dibungkus. **Kode 5.38** membaca satu baris dengan `sys.stdin.readline()`, yang mengembalikan string kosong `""` di akhir masukan. (Sebuah *baris* kosong yang diketik pengguna bukanlah `""` melainkan `"\n"`, karena karakter pindah barisnya ikut disertakan, itulah sebabnya keduanya dapat dibedakan.)

Kode 5.38 `read_until_stop.py`: membaca baris-baris dari masukan standar sampai bertemu `stop` atau akhir masukan.

```
import sys

string_list = []
current_line = sys.stdin.readline()          # "" berarti akhir masukan
while current_line != "" and current_line.rstrip("\n") != "stop":
    string_list.append(current_line.rstrip("\n"))
    current_line = sys.stdin.readline()
# Cetak isi list
print("Read " + str(len(string_list)) + " line(s): " + str(string_list))
```

```
Read 3 line(s): ['wash 3 kg', 'iron 2 shirts', 'express please']
```

Satu sifat *reader* yang sangat berguna adalah bahwa *reader* dapat diterapkan pada apa saja yang menyediakan *input stream*. `BufferedReader` yang sama dapat membaca dari halaman web, dengan membungkus aliran yang dibuka oleh `java.net.URL` atau `HttpClient` yang lebih baru, atau dari berkas, dengan membungkus sebuah `FileReader`:

```
BufferedReader fr  = new BufferedReader(new FileReader(new
File("/path/to/my/filename.txt")));
// setara dengan
BufferedReader fr2 = new BufferedReader(new FileReader("/path/to/my/filename.txt"));
```

Mana yang sebaiknya dipakai, `Scanner` atau `BufferedReader`? Keduanya bertumpang tindih, dan keduanya baik. `Scanner` lebih nyaman untuk program interaktif kecil, karena dapat membaca bilangan secara langsung (`nextInt`, `nextDouble`) dan memecah masukan menjadi token-token. `BufferedReader` lebih sederhana dan jauh lebih cepat untuk membaca teks dalam jumlah besar baris demi baris, melaporkan akhir masukan dengan `null` alih-alih eksepsi, dan memaksa Anda menghadapi `IOException` dengan jujur. Praktik yang lazim dan masuk akal adalah `Scanner` untuk dialog papan ketik dan `BufferedReader` untuk berkas, dan itulah yang dilakukan sisa bab ini. Di Python tidak ada pilihan seperti itu yang perlu diambil: `input()` untuk dialog, dan iterasi atas objek berkas untuk berkas.

5.8.2 Menutup apa yang Anda buka

Setiap berkas yang terbuka memakai sebuah sumber daya sistem operasi, yaitu *file handle* (pegangan berkas), dan cacah pegangan yang boleh dimiliki sebuah program terbatas. *Output stream* juga menyimpan data di dalam *buffer* dan baru menjamin data itu sampai ke disk ketika aliran tersebut *ditutup*. Karena itu setiap *stream* harus ditutup begitu Anda selesai memakainya,

entah terjadi eksepsi di tengah jalan entah tidak. Itulah persis jenis pekerjaan yang tempatnya di dalam blok `finally`.

Kode 5.39 menyalin sebuah berkas satu bita setiap kali, memakai `FileInputStream` dan `FileOutputStream` (aliran bita mentah, cocok untuk jenis berkas apa pun), lalu menutup keduanya di dalam `finally`. Beginilah cara yang terpaksa dipakai sebelum Java 7, dan Anda masih akan menjumpainya dalam kode yang lebih lama.

Kode 5.39 `CopyFileClassic.java`: menyalin berkas dan menutup kedua *stream* di dalam `finally`, cara sebelum Java 7.

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;

public class CopyFileClassic {
    public static void main(String[] args) {
        String from = args.length > 0 ? args[0] : "notes.txt";
        String to = args.length > 1 ? args[1] : "notes_copy.txt";
        int exitCode = 0;
        int count = 0;
        FileInputStream source = null;
        FileOutputStream target = null;
        try {
            source = new FileInputStream(from);
            target = new FileOutputStream(to);
            for (int c = source.read(); c != -1; c = source.read()) {
                target.write(c);                    // satu bita setiap kali
                count++;
            }
        } catch (IOException e) {
            System.err.println("IO error " + from);
            System.err.println(e.getMessage());
            exitCode = 1;                           // kita berubah pikiran
        } finally {
            try {
                if (source != null) {
                    source.close();
                }
            } catch (IOException e) { /* tidak ada lagi yang masuk akal untuk dilakukan */
}
            try {
                if (target != null) {
                    target.close();
                }
            } catch (IOException e) { /* tidak ada lagi yang masuk akal untuk dilakukan */
}
        }
        System.out.println("Copied " + count + " bytes from " + from + " to " + to
                + ", exit code " + exitCode);
        System.exit(exitCode);
    }
}
```

```
Copied 102 bytes from notes.txt to notes_copy.txt, exit code 0
```

`notes.txt` berukuran 102 bita (99 karakter ditambah tiga pindah baris, sebagaimana ditegaskan penyelesaian Soal Latihan 5.13), dan ke-102 bita itu tersalin semuanya. Bacalah strukturnya dengan cermat, sebab strukturnya lebih sulit daripada kelihatannya.

- Kedua variabel dideklarasikan *sebelum* `try`, dan diberi nilai `null`, agar blok `finally` dapat melihatnya. Variabel yang dideklarasikan di dalam blok `try` tidak akan ada di dalam `finally`.

- Di dalam `finally` kita harus memeriksa `null`, karena eksepsi mungkin terjadi sebelum sebuah *stream* sempat dibuka. Jika `notes.txt` tidak ada, `source` akan tetap `null` dan `target` tidak akan pernah dibuat; memanggil `close()` pada `null` akan melempar `NullPointerException` dari dalam `finally`, sehingga menyembunyikan masalah yang sebenarnya.
- `close()` sendiri dapat melempar `IOException`, sehingga setiap panggilan memerlukan `try` dan `catch`-nya sendiri, *di dalam* `finally`. Keduanya dipisahkan agar kegagalan menutup `source` tidak menghalangi kita menutup `target`.
- Pesan-pesan kesalahan dikirim ke `System.err`, yaitu aliran *error* standar, yang biasanya juga layar tetapi dapat dialihkan secara terpisah dari keluaran biasa.

Hampir separuh kode itu mengurusi penutupan. Hal ini bisa menjadi rumit, dan dalam praktik para pemrogram sering keliru secara halus. **Kode 5.40** adalah pola yang sama di Python, tempat `close()` pada sebuah berkas tidak melempar eksepsi dalam pemakaian normal, sehingga blok `finally`-nya lebih pendek.

Kode 5.40 `copy_file_classic.py`: penyalinan yang sama dengan panggilan `close()` eksplisit di dalam `finally`.

```
import sys


def main():
    source_name = sys.argv[1] if len(sys.argv) > 1 else "notes.txt"
    target_name = sys.argv[2] if len(sys.argv) > 2 else "notes_copy.txt"
    exit_code = 0
    count = 0
    source = None
    target = None
    try:
        source = open(source_name, "rb")
        target = open(target_name, "wb")
        c = source.read(1)
        while c != b"":                     # satu bita setiap kali
            target.write(c)
            count += 1
            c = source.read(1)
    except OSError as e:
        print("IO error " + source_name, file=sys.stderr)
        print(e, file=sys.stderr)
        exit_code = 1                       # kita berubah pikiran
    finally:
        if source is not None:
            source.close()
        if target is not None:
            target.close()
    print("Copied " + str(count) + " bytes from " + source_name + " to " + target_name
          + ", exit code " + str(exit_code))
    sys.exit(exit_code)


if __name__ == "__main__":
    main()
```

```
Copied 102 bytes from notes.txt to notes_copy.txt, exit code 0
```

5.8.3 `try`-with-resources dan `with`

Karena pola pada **Kode 5.39** begitu lazim dan begitu mudah keliru, Java 7 menambahkan pola yang lebih baik: *`try`-with-resources* (`try` dengan sumber daya). Objek apa pun yang kelasnya mengimplementasikan antarmuka `AutoCloseable` (semua *stream*, *reader*, dan *writer*

mengimplementasikannya) dapat dibuka di dalam tanda kurung tepat sesudah kata `try`. Java lalu menjamin akan menutupnya ketika blok `try` berakhir, baik berakhir secara normal, lewat `return`, maupun karena eksepsi, dan Java menutup beberapa sumber daya dalam urutan kebalikan dari urutan pembukaannya. **Kode 5.41** adalah versi modern dari penyalinan tadi. Versi ini juga memindahkan hingga 8 KB per panggilan `read` alih-alih satu bita, yang jauh lebih cepat untuk berkas-berkas besar.

Kode 5.41 `CopyFile.java`: menyalin dengan `try`-with-resources, yang menutup kedua *stream* secara otomatis.

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;

public class CopyFile {
    static long copy(String from, String to) throws IOException {
        long count = 0;
        byte[] buffer = new byte[8192];             // pindahkan hingga 8 KB per panggilan
        try (FileInputStream source = new FileInputStream(from);
             FileOutputStream target = new FileOutputStream(to)) {
            int n;
            while ((n = source.read(buffer)) != -1) {
                target.write(buffer, 0, n);
                count += n;
            }
        }                                           // kedua stream ditutup di sini, selalu
        return count;
    }

    public static void main(String[] args) {
        String[][] jobs = {{"notes.txt", "notes_copy2.txt"}, {"nothing.txt", "x.txt"}};
        for (String[] job : jobs) {
            try {
                long n = copy(job[0], job[1]);
                System.out.println("Copied " + n + " bytes from " + job[0] + " to " +
job[1]);
            } catch (IOException e) {
                System.out.println("Could not copy " + job[0] + ": " + e.getMessage());
            }
        }
    }
}
```

```
Copied 102 bytes from notes.txt to notes_copy2.txt
Could not copy nothing.txt: nothing.txt (No such file or directory)
```

Seluruh kode penutupan telah lenyap, namun jaminannya lebih kuat daripada sebelumnya. Perhatikan pula pembagian kerja dari Subbab 5.6.3: `copy` mendeklarasikan `throws IOException`, karena ia tidak tahu apa arti sebuah kegagalan, dan `main` menangkap lalu melaporkannya. Pekerjaan kedua meminta berkas yang tidak ada, dan program melaporkan kenyataan itu lalu terus berjalan.

`source.read(buffer)` mengisi larik `buffer` sebanyak yang dapat diisinya dan mengembalikan berapa bita yang ditaruhnya di sana, atau -1 di akhir berkas. `target.write(buffer, 0, n)` menulis tepat `n` bita itu, bukan seluruh *buffer*, karena pembacaan terakhir biasanya hanya mengisinya sebagian.

Python sudah mempunyai gagasan yang sama sejak versi 2.5: pernyataan `with`. Objek yang dipakai sesudah `with` adalah sebuah *context manager* (pengelola konteks), dan objek berkas

termasuk di antaranya: ketika blok `with` berakhir, dengan alasan apa pun, berkasnya ditutup. **Kode 5.42** membuka kedua berkas dalam satu `with`.

Kode 5.42 `copy_file.py`: menyalin dengan pernyataan `with`.

```
def copy(source_name, target_name):
    count = 0
    with open(source_name, "rb") as source, open(target_name, "wb") as target:
        while True:
            chunk = source.read(8192)       # pindahkan hingga 8 KB per panggilan
            if not chunk:
                break
            target.write(chunk)
            count += len(chunk)
    return count                            # kedua berkas ditutup di sini, selalu


def main():
    jobs = [("notes.txt", "notes_copy2.txt"), ("nothing.txt", "x.txt")]
    for source_name, target_name in jobs:
        try:
            n = copy(source_name, target_name)
            print("Copied " + str(n) + " bytes from " + source_name + " to " + target_name)
        except OSError as e:
            print("Could not copy " + source_name + ": " + str(e))


if __name__ == "__main__":
    main()
```

```
Copied 102 bytes from notes.txt to notes_copy2.txt
Could not copy nothing.txt: [Errno 2] No such file or directory: 'nothing.txt'
```

> **WAWASAN KUNCI:** Dalam kode yang baru, bukalah setiap berkas dengan `try`-with-resources di Java dan dengan `with` di Python. Tulislah blok `finally` dengan tangan hanya untuk sumber daya yang tidak dapat dikelola oleh mekanisme-mekanisme ini.

5.8.4 Menulis dan menambahkan teks

Untuk menulis *teks* kita memakai *writer*, padanan karakter dari *output stream*. `FileWriter` menulis karakter ke sebuah berkas; membungkusnya dengan `PrintWriter` memberi kita `print`, `println`, dan `printf` yang sudah akrab. `FileWriter` biasanya *mengganti* isi berkas (atau membuat berkasnya jika belum ada); memberikan `true` sebagai argumen tambahan membukanya dalam mode *append* (tambah di akhir), sehingga teks baru ditaruh sesudah apa yang sudah ada. Python menyatakan pilihan-pilihan yang sama dengan argumen *mode* pada `open`: `"r"` untuk membaca (bawaan), `"w"` untuk menulis dari awal, `"a"` untuk menambahkan di akhir.

Profitina Laundry mencetak struk. **Kode 5.43** menyusun teks struk di memori, menuliskannya ke `receipt.txt`, menambahkan satu baris ucapan terima kasih, lalu membaca berkas itu kembali sambil menomori baris-barisnya.

Kode 5.43 `ReceiptWriter.java`: menyusun teks dengan `StringBuilder`, menulis berkas, menambahkan isi ke dalamnya, lalu membacanya kembali.

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.nio.charset.StandardCharsets;

public class ReceiptWriter {
    public static void main(String[] args) {
        String[] items = {"WASH 3.5 kg", "IRON 2.0 kg", "EXPRESS surcharge"};
        long[] prices = {24500, 10000, 17250};

        StringBuilder sb = new StringBuilder();       // susun teksnya sekali saja, dengan
murah
        long total = 0;
        for (int i = 0; i < items.length; i++) {
            sb.append(String.format("%-20s %,10d%n", items[i], prices[i]));
            total += prices[i];
        }
        sb.append(String.format("%-20s %,10d%n", "TOTAL (Rp)", total));

        try (PrintWriter out = new PrintWriter(
                new FileWriter("receipt.txt", StandardCharsets.UTF_8))) {
            out.print(sb);                            // "w": buat baru atau timpa
        } catch (IOException e) {
            System.out.println("Cannot write receipt: " + e.getMessage());
            return;
        }
        try (PrintWriter log = new PrintWriter(
                new FileWriter("receipt.txt", StandardCharsets.UTF_8, true))) {
            log.println("Thank you for choosing Profitina Laundry!");   // true = tambahkan
di akhir
        } catch (IOException e) {
            System.out.println("Cannot append: " + e.getMessage());
            return;
        }

        try (BufferedReader in = new BufferedReader(
                new FileReader("receipt.txt", StandardCharsets.UTF_8))) {
            String line;
            int number = 1;
            while ((line = in.readLine()) != null) {  // null berarti akhir berkas
                System.out.println(number + " | " + line);
                number++;
            }
        } catch (IOException e) {
            System.out.println("Cannot read receipt: " + e.getMessage());
        }
```

Kode 5.43 (Lanjutan) *ReceiptWriter.java: menyusun teks dengan StringBuilder, menulis berkas, menambahkan isi ke dalamnya, lalu membacanya kembali.*

```
    }
}
```

```
1 | WASH 3.5 kg             24,500
2 | IRON 2.0 kg             10,000
3 | EXPRESS surcharge       17,250
4 | TOTAL (Rp)              51,750
5 | Thank you for choosing Profitina Laundry!
```

Beberapa gagasan bertemu dalam program ini.

- **Menyusun teks secara efisien.** String Java bersifat *immutable* (tidak dapat diubah), sehingga `text = text + more` di dalam perulangan membangun string yang benar-benar baru setiap

kali, sambil menyalin semua yang sudah ada. Itulah jebakan kuadratik yang sama dengan memperbesar larik satu slot setiap kali (Persamaan (5.2)). `StringBuilder` adalah *buffer* karakter yang dapat tumbuh, dikelola persis seperti larik tersembunyi di dalam `ArrayList`: `append` menambahkan di akhir dengan murah, dan `toString()` (yang dipanggil secara implisit oleh `print`) menghasilkan string jadinya satu kali saja. Idiom Python, pada **Kode 5.44**, adalah mengumpulkan potongan-potongannya di dalam sebuah list lalu menggabungkannya sekali dengan `"".join(parts)`.

- **Pemformatan.** `%-20s` mencetak string rata kiri dalam 20 kolom, `%,10d` mencetak bilangan bulat rata kanan dalam 10 kolom dengan pemisah ribuan, dan `%n` adalah pindah baris yang sesuai dengan sistem operasinya.
- **Pengodean karakter.** Berkas menyimpan bita, dan *encoding* (pengodean) adalah aturan yang mengubah karakter menjadi bita dan sebaliknya. Kita menyerahkan `StandardCharsets.UTF_8` secara eksplisit kepada setiap `FileWriter` dan `FileReader`, dan `encoding="utf-8"` kepada setiap `open` di Python. Tanpa itu, kedua bahasa kembali ke pengodean bawaan platform, dan khususnya sistem Windows yang lebih lama memakai bawaan yang berbeda, sehingga berkas yang memuat nama-nama seperti “José” atau “Müller” yang ditulis di satu mesin dapat menjadi kacau di mesin lain. Java 18 dan sesudahnya memakai UTF-8 sebagai bawaan, tetapi menyatakannya secara eksplisit tidak memakan biaya apa pun dan membuat maksud program menjadi terang.
- **Membaca sampai habis.** `while ((line = in.readLine()) != null)` adalah perulangan Java yang baku untuk membaca berkas teks: penugasan di dalam tanda kurung membaca satu baris dan perbandingannya menguji baris itu. Perulangan berhenti di akhir berkas, ketika `readLine` mengembalikan `null`.
- **Blok `try` yang terpisah.** Penulisan dan pembacaan berada dalam pernyataan terpisah dengan pesannya masing-masing, agar pengguna mengetahui langkah mana yang gagal. `return` di dalam setiap `catch` menghentikan metode, karena tidak ada gunanya membaca struk yang tidak berhasil ditulis.

Kode 5.44 `receipt_writer.py`: struk yang sama dengan `"".join`, mode `open` `"w"` dan `"a"`, serta perulangan `for` atas berkas.

```
def main():
    items = ["WASH 3.5 kg", "IRON 2.0 kg", "EXPRESS surcharge"]
    prices = [24500, 10000, 17250]

    parts = []                              # kumpulkan potongannya, gabungkan sekali
    total = 0
    for i in range(len(items)):
        parts.append(f"{items[i]:<20} {prices[i]:>10,}\n")
        total += prices[i]
    parts.append(f"{'TOTAL (Rp)':<20} {total:>10,}\n")
    text = "".join(parts)

    try:
        with open("receipt.txt", "w", encoding="utf-8") as out:
            out.write(text)                 # "w": buat baru atau timpa
        with open("receipt.txt", "a", encoding="utf-8") as log:
            log.write("Thank you for choosing Profitina Laundry!\n")   # "a": tambahkan di
akhir
    except OSError as e:
        print("Cannot write receipt:", e)
        return

    try:
        with open("receipt.txt", encoding="utf-8") as f:
            for number, line in enumerate(f, start=1):   # perulangan berakhir di akhir
berkas
                print(str(number) + " | " + line.rstrip("\n"))
    except OSError as e:
        print("Cannot read receipt:", e)


if __name__ == "__main__":
    main()
```

```
1 | WASH 3.5 kg             24,500
2 | IRON 2.0 kg             10,000
3 | EXPRESS surcharge       17,250
4 | TOTAL (Rp)              51,750
5 | Thank you for choosing Profitina Laundry!
```

Di Python, objek berkas yang dibuka untuk dibaca dapat langsung dipakai dalam perulangan `for`, yang menyerahkan satu baris setiap kali (masing-masing masih diakhiri `"\n"`, yang kita buang dengan `rstrip`) dan berhenti di akhir berkas. `enumerate(f, start=1)` menomori baris-baris itu mulai dari 1. Spesifikasi format `{prices[i]:>10,}` berarti "rata kanan dalam 10 kolom, dengan pemisah ribuan", yaitu cara Python mengeja `%,10d` milik Java.

5.8.5 Berkas CSV

Berkas CSV, singkatan dari *comma-separated values* (nilai-nilai yang dipisahkan koma), adalah cara paling sederhana untuk menyimpan tabel sebagai teks: satu baris per baris tabel, dan *field-field* setiap baris dipisahkan dengan koma. Baris pertama biasanya berupa *header* (kepala) yang menamai kolom-kolomnya. Aplikasi *spreadsheet* dapat membuka dan menyimpan berkas CSV, dan hampir setiap perkakas data dapat membacanya, sehingga CSV menjadi format yang paling lazim untuk bertukar tabel antarprogram. Uraian formalnya tersedia sebagai RFC 4180 [4].

Cara yang paling jelas untuk membaca satu baris CSV adalah memecahnya di setiap koma. **Kode 5.45** menunjukkan bahwa cara yang jelas itu menyembunyikan dua jebakan, dan mencetak setiap *field* di antara `<` dan `>` agar spasi dan *field* kosong menjadi terlihat.

Kode 5.45 `CsvSplit.java`: dua jebakan dalam memecah baris CSV di Java.

```java
public class CsvSplit {
    // Menampilkan setiap field di antara < dan > agar spasi dan field kosong terlihat.
    static String show(String[] fields) {
        StringBuilder sb = new StringBuilder(fields.length + " fields: ");
        for (String f : fields) {
            sb.append('<').append(f).append('>');
        }
        return sb.toString();
    }

    public static void main(String[] args) {
        String line = "L-001, Sari ,WASH,3.5,";
        // split(",") diam-diam membuang field kosong di ujung; -1 mempertahankan semua
field.
        System.out.println("split(\",\")     -> " + show(line.split(",")));
        String[] exact = line.split(",", -1);
        System.out.println("split(\",\", -1) -> " + show(exact));
        for (int i = 0; i < exact.length; i++) {
            exact[i] = exact[i].trim();              // buang spasi di sekitar setiap field
        }
        System.out.println("trimmed        -> " + show(exact));

        String quoted = "L-002,\"Santoso, Budi\",IRON,2.0,no";
        System.out.println("naive split    -> " + show(quoted.split(",", -1)));
    }
}
```

```
split(",")     -> 4 fields: <L-001>< Sari ><WASH><3.5>
split(",", -1) -> 5 fields: <L-001>< Sari ><WASH><3.5><>
trimmed        -> 5 fields: <L-001><Sari><WASH><3.5><>
naive split    -> 6 fields: <L-002><"Santoso>< Budi"><IRON><2.0><no>
```

Jebakan pertama khas Java: `String.split(",")` *diam-diam membuang field kosong di ujung* hasilnya. Baris kita mempunyai lima *field*, yang terakhir kosong, tetapi `split(",")` mengembalikan empat. Program yang kemudian membaca `fields[4]` akan macet dengan `ArrayIndexOutOfBoundsException`, dan hanya pada baris-baris yang *field* terakhirnya kebetulan kosong, sehingga *bug*-nya muncul-hilang secara menjengkelkan. Obatnya adalah argumen kedua: `split(",", -1)` mempertahankan semua *field*. *Field-field* itu juga dapat membawa spasi nyasar, yang dibuang oleh `trim()`.

Jebakan kedua dialami setiap pemecah yang naif: sebuah *field* dapat memuat koma di dalamnya. Konvensi CSV adalah menaruh *field* seperti itu di antara tanda kutip ganda, `"Santoso, Budi"`, tetapi memecah di setiap koma akan memotong nama itu menjadi dua dan menghasilkan enam *field*. Menangani tanda kutip dengan benar memerlukan sebuah mesin keadaan kecil yang mengingat apakah ia sedang berada di dalam tanda kutip; pustaka standar Java tidak menyertakannya, dan program Java yang serius memakai pustaka seperti Apache Commons CSV. Pustaka standar Python menyertakannya, yaitu modul `csv` [5], seperti ditunjukkan **Kode 5.46**.

Kode 5.46 `csv_split.py`: memecah dengan tangan, dan memecah dengan benar memakai modul `csv`.

```
import csv
import io


def show(fields):
    """Menampilkan setiap field di antara < dan > agar spasi dan field kosong terlihat."""
    return str(len(fields)) + " fields: " + "".join("<" + f + ">" for f in fields)


line = "L-001, Sari ,WASH,3.5,"
exact = line.split(",")                       # Python mempertahankan field kosong di ujung
print('split(",")     -> ' + show(exact))
exact = [f.strip() for f in exact]            # buang spasi di sekitar setiap field
print("trimmed        -> " + show(exact))

quoted = 'L-002,"Santoso, Budi",IRON,2.0,no'
print("naive split    -> " + show(quoted.split(",")))
row = next(csv.reader(io.StringIO(quoted)))  # modul csv memahami tanda kutip
print("csv.reader     -> " + show(row))
```

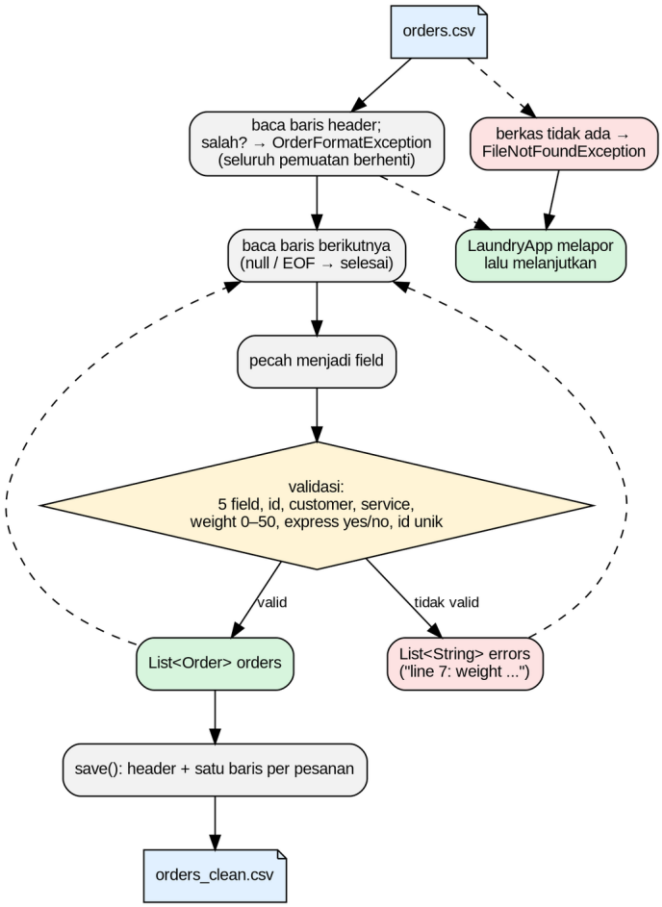
```
split(",")     -> 5 fields: <L-001>< Sari ><WASH><3.5><>
trimmed        -> 5 fields: <L-001><Sari><WASH><3.5><>
naive split    -> 6 fields: <L-002><"Santoso>< Budi"><IRON><2.0><no>
csv.reader     -> 5 fields: <L-002><Santoso, Budi><IRON><2.0><no>
```

`split(",")` di Python mempertahankan *field* terakhir yang kosong, sehingga jebakan pertama tidak ada di sana. `csv.reader` memahami tanda kutip dan mengembalikan lima *field*, dengan koma di dalam nama tetap terjaga. (`io.StringIO` membuat sebuah string berperilaku seperti berkas, sehingga `csv.reader` dapat membaca darinya; dengan berkas sungguhan Anda menyerahkan objek berkasnya.)

Untuk Profitina Laundry kita memakai aturan yang sederhana dan tegas, yang dapat ditegakkan oleh kedua bahasa: *tidak ada field yang boleh memuat koma*. Pemuat versi Java memecah dengan `split(",", -1)`, pemuat versi Python memakai `csv.reader`, dan setiap baris dengan cacah *field* yang salah ditolak dengan pesan yang jelas. Hal itu membawa kita ke studi kasus.

5.9 Profitina Laundry: Memuat dan Menyimpan Pesanan

Profitina Laundry menyimpan pesanan-pesanannya dalam sebuah berkas CSV, `orders.csv`, yang kadang disunting dengan tangan oleh petugas loket di aplikasi *spreadsheet*. Tugas kita adalah memuat berkas itu ke dalam list berisi objek `Order`, menolak setiap baris yang tidak sah dengan pesan yang menyatakan dengan tepat apa yang salah dan di mana, terus berjalan sesudah menjumpai baris yang buruk, menghitung harga, dan menyimpan pesanan-pesanan yang sah ke sebuah berkas bersih yang dapat dibaca kembali oleh program. Rencananya digambarkan pada **Gambar 5.7**.



Gambar 5.7 Alur pemuatan pesanan. Baris yang buruk menjadi satu pesan dalam daftar kesalahan dan pemuatan berlanjut; berkas yang tidak ada atau *header* yang salah menghentikan pemuatan dan dilaporkan oleh aplikasi.

Dua jenis kegagalan diperlakukan secara berbeda, dan pembedaan itulah jantung rancangannya. *Baris yang buruk* adalah masalah setempat: sisa berkas mungkin sepenuhnya baik, jadi kita mencatat masalahnya lalu melanjutkan. *Berkas yang tidak ada*, atau berkas yang *header*-nya menunjukkan bahwa ia sama sekali bukan berkas pesanan, adalah masalah menyeluruh: tidak ada yang masuk akal untuk dimuat, sehingga pemuat berhenti dan melempar eksepsi, dan aplikasilah yang memutuskan apa yang akan disampaikan kepada pengguna.

Inilah berkasnya. Berkas ini sengaja dirusak dengan tujuh cara yang berbeda, dan memuat satu baris kosong:

```
id,customer,service,weight_kg,express
L-001,Sari,WASH,3.5,no
L-002,Budi Santoso,WASH_IRON,5.0,yes
L-003,Rina,IRON,2,no
L-004,,WASH,4.0,no
L-005,Tono,DRY_CLEAN,1.5,no
L-006,Wati,WASH,abc,no
L-007,Agus,WASH,-3,no
L-002,Dewi,WASH,2.0,no
L-008,Joko,IRON,1.2

L-009,Lestari,WASH_IRON,7.25,maybe
L-010,Hendra,WASH,12,yes
```

Binatu itu menawarkan tiga layanan, yang ditarifkan per kilogram: `WASH` (cuci) Rp 7,000, `IRON` (setrika) Rp 5,000, dan `WASH_IRON` (cuci dan setrika) Rp 10,000. Pesanan ekspres 50% lebih mahal. Untuk pesanan seberat w kilogram dengan tarif r rupiah per kilogram, harganya adalah

$$p = \begin{cases} \text{round}(w \cdot r) & \text{pesanan biasa,} \\ \text{round}(w \cdot r) + \lfloor \text{round}(w \cdot r)/2 \rfloor & \text{pesanan ekspres.} \end{cases} \tag{5.4}$$

Kita menghitung dalam rupiah bulat dengan bilangan bulat, karena jumlah uang tidak semestinya disimpan dalam variabel *floating-point* lebih lama daripada yang diperlukan: $0.1 + 0.2$ tidak persis sama dengan 0.3 dalam *floating-point* biner. (Buku ini menulis angka dengan titik desimal dan koma pemisah ribuan, sama seperti keluaran program, agar teks dan keluaran selalu cocok.) Sebagai contoh, menurut Persamaan (5.4), cucian Sari seberat 3.5 kg berharga $\text{round}(3.5 \times 7000) = 24{,}500$, dan cuci-setrika ekspres Budi seberat 5 kg berharga $50{,}000 + 25{,}000 = 75{,}000$.

Aturan validasinya adalah: tepat lima *field*; id dan nama pelanggan yang tidak kosong; layanan yang dikenal; berat berupa bilangan yang lebih besar dari 0 dan paling banyak 50 kg; penanda ekspres berupa `yes` atau `no`; dan id yang belum dipakai pada baris sebelumnya.

5.9.1 Kelas `Order`

Kode 5.47 adalah kelas yang mewakili satu pesanan. Kelas ini mengetahui harganya sendiri dan cara menuliskan dirinya sebagai satu baris CSV.

Kode 5.47 `Order.java`: satu pesanan cucian, beserta harganya dan bentuk CSV-nya.

```
import java.util.Locale;

public class Order {
    // Larik paralel: SERVICES[i] bertarif RATES[i] rupiah per kilogram.
    static final String[] SERVICES = {"WASH", "IRON", "WASH_IRON"};
    static final long[] RATES = {7000, 5000, 10000};

    private final String id;
    private final String customer;
    private final String service;
    private final double weightKg;
    private final boolean express;

    public Order(String id, String customer, String service, double weightKg, boolean
express) {
        this.id = id;
        this.customer = customer;
        this.service = service;
        this.weightKg = weightKg;
        this.express = express;
    }

    /** Mengembalikan tarif sebuah kode layanan, atau -1 jika kodenya tidak dikenal. */
    static long rateFor(String service) {
        for (int i = 0; i < SERVICES.length; i++) {
            if (SERVICES[i].equals(service)) {
                return RATES[i];
            }
        }
        return -1;
    }

    public long price() {
        long base = Math.round(weightKg * rateFor(service));
        return express ? base + base / 2 : base;       // ekspres 50% lebih mahal
    }

    public String getId() { return id; }
    public String getService() { return service; }
    public double getWeightKg() { return weightKg; }

    public String toCsv() {
        return id + "," + customer + "," + service + "," + weightKg + ","
                + (express ? "yes" : "no");
    }
```

Kode 5.47 (Lanjutan) *Order.java: satu pesanan cucian, beserta harganya dan bentuk CSV-nya.*

```
    @Override
    public String toString() {
        return String.format(Locale.US, "%-6s %-13s %-9s %5.2f kg %-3s Rp %,9d",
                id, customer, service, weightKg, express ? "exp" : "", price());
    }
}
```

Layanan-layanan dan tarifnya disimpan dalam dua *parallel array* (larik paralel): `SERVICES[i]` bertarif `RATES[i]` per kilogram. Keduanya larik, bukan list, karena himpunan layanannya sudah tetap ketika program ditulis. `rateFor` mencari di dalam `SERVICES` dengan perulangan biasa dan mengembalikan −1 untuk kode yang tidak dikenal, konvensi yang sama dengan `indexOf`. `price()` mengimplementasikan Persamaan (5.4): `Math.round` membulatkan hasil kali ke rupiah bulat terdekat dan mengembalikan sebuah `long`, dan `base / 2` adalah pembagian bilangan bulat. `toString` memakai `Locale.US` agar pemisah ribuannya berupa koma di setiap komputer; *locale*

Indonesia akan mencetak `24.500`, yang benar di Indonesia tetapi akan membuat keluarannya berbeda dari satu mesin ke mesin lain.

Kelas Python-nya, **Kode 5.48**, memakai sebuah *dictionary* (kamus) untuk tarif-tarifnya. *Dictionary* memetakan setiap kunci langsung ke sebuah nilai dan merupakan alat yang wajar di Python untuk pekerjaan ini; Bab 9 mempelajari padanannya di Java, yaitu `Map`.

Kode 5.48 `order.py`: kelas yang sama di Python, dengan sebuah *dictionary* berisi tarif.

```
# Sebuah dictionary memerankan dua larik paralel versi Java.
RATES = {"WASH": 7000, "IRON": 5000, "WASH_IRON": 10000}


def rate_for(service):
    """Mengembalikan tarif sebuah kode layanan, atau -1 jika kodenya tidak dikenal."""
    return RATES.get(service, -1)


class Order:
    def __init__(self, order_id, customer, service, weight_kg, express):
        self.order_id = order_id
        self.customer = customer
        self.service = service
        self.weight_kg = weight_kg
        self.express = express

    def price(self):
        base = round(self.weight_kg * rate_for(self.service))
        return base + base // 2 if self.express else base   # ekspres 50% lebih mahal

    def to_csv(self):
        return ",".join([self.order_id, self.customer, self.service,
                         str(self.weight_kg), "yes" if self.express else "no"])

    def __str__(self):
        return (f"{self.order_id:<6} {self.customer:<13} {self.service:<9} "
                f"{self.weight_kg:5.2f} kg {'exp' if self.express else '':<3} "
                f"Rp {self.price():9,}")
```

5.9.2 Eksepsinya

Sebuah baris yang buruk diuraikan oleh eksepsi terperiksa buatan sendiri yang mencatat nomor barisnya, **Kode 5.49**. Eksepsi ini memperluas `Exception`, bukan `RuntimeException`, karena data tidak sah dari luar program adalah hasil yang dapat diduga dan harus dipersiapkan oleh setiap pemanggil.

Kode 5.49 `OrderFormatException.java`: eksepsi terperiksa untuk satu baris yang tidak sah.

```java
/** Eksepsi terperiksa: satu baris berkas pesanan bukan pesanan yang sah. */
public class OrderFormatException extends Exception {
    private final int lineNumber;

    public OrderFormatException(int lineNumber, String message) {
        this(lineNumber, message, null);
    }

    public OrderFormatException(int lineNumber, String message, Throwable cause) {
        super("line " + lineNumber + ": " + message, cause);
        this.lineNumber = lineNumber;
    }

    public int getLineNumber() {
        return lineNumber;
    }
}
```

Konstruktor pertama menyerahkan pekerjaannya kepada konstruktor kedua dengan `this(...)`, sambil memberikan `null` sebagai penyebabnya, sehingga pesan hanya disusun di satu tempat. Setiap pesan diawali nomor baris, agar pengguna dapat langsung menuju masalahnya di aplikasi *spreadsheet*. **Kode 5.50** adalah padanannya di Python.

Kode 5.50 `order_errors.py`: eksepsi yang sama di Python.

```python
class OrderFormatError(Exception):
    """Satu baris berkas pesanan bukan pesanan yang sah."""

    def __init__(self, line_number, message):
        super().__init__("line " + str(line_number) + ": " + message)
        self.line_number = line_number
```

5.9.3 Memuat dan menyimpan

Kode 5.51 adalah jantung studi kasus ini.

Kode 5.51 `OrderCsv.java`: mengurai dan memvalidasi satu baris, memuat seluruh berkas, dan menyimpan.

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.nio.charset.StandardCharsets;
import java.util.ArrayList;
import java.util.List;

public class OrderCsv {
    static final String HEADER = "id,customer,service,weight_kg,express";

    /** Mengubah satu baris CSV menjadi Order, atau menjelaskan persis mengapa tidak bisa.
*/
    static Order parseLine(String line, int lineNumber) throws OrderFormatException {
        String[] f = line.split(",", -1);
        if (f.length != 5) {
            throw new OrderFormatException(lineNumber, "expected 5 fields, found " +
f.length);
        }
        for (int i = 0; i < f.length; i++) {
            f[i] = f[i].trim();
        }
        if (f[0].isEmpty()) {
            throw new OrderFormatException(lineNumber, "order id is empty");
        }
        if (f[1].isEmpty()) {
            throw new OrderFormatException(lineNumber, "customer name is empty");
        }
        if (Order.rateFor(f[2]) < 0) {
            throw new OrderFormatException(lineNumber, "unknown service \"" + f[2] + "\"");
        }
        double kg;
        try {
            kg = Double.parseDouble(f[3]);
        } catch (NumberFormatException nfe) {
            throw new OrderFormatException(lineNumber,
                    "weight \"" + f[3] + "\" is not a number", nfe);
        }
        if (!(kg > 0 && kg <= 50)) {
            throw new OrderFormatException(lineNumber, "weight " + f[3] + " is outside 0-50
kg");
        }
        if (!f[4].equals("yes") && !f[4].equals("no")) {
            throw new OrderFormatException(lineNumber,
                    "express must be yes or no, found \"" + f[4] + "\"");
        }
        return new Order(f[0], f[1], f[2], kg, f[4].equals("yes"));
    }
```

Kode 5.51 (Lanjutan) *OrderCsv.java: mengurai dan memvalidasi satu baris, memuat seluruh berkas, dan menyimpan.*

```
    /**
     * Memuat setiap pesanan yang sah; setiap baris tidak sah menjadi satu pesan di errors.
     * Berkas yang tidak ada atau header yang salah menghentikan seluruh pemuatan dengan
eksepsi.
     */
    static List<Order> load(String fileName, List<String> errors)
            throws IOException, OrderFormatException {
        List<Order> orders = new ArrayList<>();
        List<String> seenIds = new ArrayList<>();
        try (BufferedReader in = new BufferedReader(
                new FileReader(fileName, StandardCharsets.UTF_8))) {
            String header = in.readLine();
            if (header == null || !header.trim().equals(HEADER)) {
                throw new OrderFormatException(1, "header must be " + HEADER);
            }
            int lineNumber = 1;
            String line;
            while ((line = in.readLine()) != null) {
                lineNumber++;
                if (line.isBlank()) {
                    continue;                              // toleransi terhadap baris
kosong
                }
                try {
                    Order order = parseLine(line, lineNumber);
                    if (seenIds.contains(order.getId())) {
                        throw new OrderFormatException(lineNumber, "duplicate id " +
order.getId());
                    }
                    seenIds.add(order.getId());
                    orders.add(order);
                } catch (OrderFormatException e) {
                    errors.add(e.getMessage());            // catat lalu lanjutkan
                }
            }
        }
        return orders;
    }
```

Kode 5.51 (Lanjutan) *OrderCsv.java: mengurai dan memvalidasi satu baris, memuat seluruh berkas, dan menyimpan.*

```
    static void save(String fileName, List<Order> orders) throws IOException {
        try (PrintWriter out = new PrintWriter(new FileWriter(fileName,
StandardCharsets.UTF_8))) {
            out.println(HEADER);
            for (Order o : orders) {
                out.println(o.toCsv());
            }
        }
    }
}
```

Ikutilah perjalanan satu baris melalui `parseLine`. Baris itu dipecah dengan `split(",", -1)`, agar *field* terakhir yang kosong tetap terhitung, dan cacah *field*-nya diperiksa *sebelum* satu *field* pun dipakai, sehingga baris yang terlalu pendek tidak mungkin menimbulkan `ArrayIndexOutOfBoundsException`. Setiap *field* dirapikan dengan `trim`. Kemudian aturan-aturannya diperiksa satu per satu, dan aturan pertama yang gagal melempar `OrderFormatException` dengan pesan yang spesifik. Berat adalah satu-satunya *field* yang

memerlukan konversi yang dapat gagal, sehingga `NumberFormatException` ditangkap di sana lalu *dibungkus* sebagai penyebab sebuah `OrderFormatException`, persis seperti pada **Kode 5.35**. Pengujian `!(kg > 0 && kg <= 50)` sengaja ditulis dengan susunan seperti itu: ia juga menolak nilai istimewa "bukan bilangan" (`NaN`), yang diterima oleh `Double.parseDouble("NaN")` dan yang membuat setiap perbandingan bernilai salah. Hanya baris yang lolos dari setiap pemeriksaan yang menjadi sebuah `Order`.

`load` mendeklarasikan dua eksepsi terperiksa, `IOException` dan `OrderFormatException`, dan komentar di kepalanya menyatakan kapan masing-masing dilemparkan. Berkasnya dibuka dengan `try`-with-resources, sehingga berkas itu tertutup apa pun yang terjadi. Baris pertama harus berupa *header*; jika tidak, metode itu melempar eksepsi, dan pemuatan berhenti. Kemudian setiap baris selebihnya melewati `try` kecilnya sendiri: hasil penguraian yang berhasil diperiksa terhadap id ganda lalu ditambahkan ke list; sebuah `OrderFormatException` ditangkap *di dalam perulangan*, pesannya ditambahkan ke list `errors` yang disediakan pemanggil, dan perulangan berlanjut ke baris berikutnya. Letak `try` bagian dalam itulah keputusan rancangan pada **Gambar 5.7** yang diwujudkan dalam kode. Pengujian id ganda memakai ulang eksepsi itu dengan melemparkannya di dalam `try` yang sama, sehingga semua baris yang buruk dilaporkan dengan satu cara yang seragam. List `seenIds` dicari dengan `contains`, yang memeriksa setiap elemen secara bergiliran; Bab 9 memperkenalkan `Set`, yang menjawab pertanyaan yang sama jauh lebih cepat untuk berkas-berkas besar.

`save` menulis *header* dan satu baris per pesanan dengan `try`-with-resources. Metode ini mendeklarasikan `IOException` dan menyerahkan keputusannya kepada pemanggil.

Modul Python-nya, **Kode 5.52**, mempunyai struktur yang sama. Modul ini membaca berkas melalui `csv.reader`, yang atribut `line_num`-nya memberikan nomor baris yang baru saja dibaca, dan membuka berkas dengan `newline=""`, sebagaimana diharuskan dokumentasi `csv`. `parse_row` menerima list berisi *field* dari `csv.reader`, dan `parse_line` membungkusnya untuk satu baris teks.

Kode 5.52 `order_csv.py`: pemuatan, validasi, dan penyimpanan yang sama di Python.

```
import csv

from order import Order, rate_for
from order_errors import OrderFormatError

HEADER = ["id", "customer", "service", "weight_kg", "express"]


def parse_row(f, line_number):
    """Mengubah satu baris CSV (list berisi string) menjadi Order, atau menjelaskan mengapa
tidak bisa."""
    if len(f) != 5:
        raise OrderFormatError(line_number, "expected 5 fields, found " + str(len(f)))
    f = [field.strip() for field in f]
    if f[0] == "":
        raise OrderFormatError(line_number, "order id is empty")
    if f[1] == "":
        raise OrderFormatError(line_number, "customer name is empty")
    if rate_for(f[2]) < 0:
        raise OrderFormatError(line_number, 'unknown service "' + f[2] + '"')
    try:
        kg = float(f[3])
    except ValueError as ve:
        raise OrderFormatError(line_number, 'weight "' + f[3] + '" is not a number') from
ve
    if not (0 < kg <= 50):
        raise OrderFormatError(line_number, "weight " + f[3] + " is outside 0-50 kg")
    if f[4] not in ("yes", "no"):
        raise OrderFormatError(line_number, 'express must be yes or no, found "' + f[4] +
'"')
    return Order(f[0], f[1], f[2], kg, f[4] == "yes")


def parse_line(line, line_number):
    return parse_row(line.split(","), line_number)


def load(file_name, errors):
    """Memuat setiap pesanan yang sah; setiap baris tidak sah menjadi satu pesan di errors.
```

Kode 5.52 (Lanjutan) *order_csv.py: pemuatan, validasi, dan penyimpanan yang sama di Python.*

```
    Raises:
        FileNotFoundError: berkasnya tidak ada.
        OrderFormatError: header-nya salah, jadi ini bukan berkas pesanan.
    """
    orders = []
    seen_ids = []
    with open(file_name, newline="", encoding="utf-8") as f:
        reader = csv.reader(f)
        header = next(reader, None)
        if header is None or [h.strip() for h in header] != HEADER:
            raise OrderFormatError(1, "header must be " + ",".join(HEADER))
        for row in reader:
            line_number = reader.line_num
            if row == [] or all(field.strip() == "" for field in row):
                continue                          # toleransi terhadap baris kosong
            try:
                order = parse_row(row, line_number)
                if order.order_id in seen_ids:
                    raise OrderFormatError(line_number, "duplicate id " + order.order_id)
                seen_ids.append(order.order_id)
                orders.append(order)
            except OrderFormatError as e:
                errors.append(str(e))             # catat lalu lanjutkan
    return orders


def save(file_name, orders):
    with open(file_name, "w", newline="", encoding="utf-8") as f:
        f.write(",".join(HEADER) + "\n")
        for o in orders:
            f.write(o.to_csv() + "\n")
```

5.9.4 Aplikasinya

Akhirnya, **Kode 5.53** mengikat semuanya menjadi satu. `loadAndReport` adalah tempat keputusan tentang kegagalan menyeluruh diambil: metode ini menangkap `FileNotFoundException`, `OrderFormatException`, dan `IOException` lainnya, masing-masing dengan pesan yang sesuai, yang paling spesifik lebih dahulu. `main` memuat `orders.csv`, menambahkan satu pesanan baru yang diketik di loket, mencetak setiap pesanan beserta harganya dan total pendapatan, menyimpan pesanan-pesanan yang sah ke `orders_clean.csv`, lalu membaca berkas itu kembali untuk membuktikan bahwa apa yang kita tulis dapat kita baca. Kemudian, untuk memperlihatkan kegagalan menyeluruh, program mencoba berkas yang tidak ada dan berkas yang sama sekali bukan berkas pesanan.

Kode 5.53 `LaundryApp.java`: memuat, melaporkan, menghitung harga, menyimpan, dan memuat ulang pesanan-pesanan.

```
import java.io.FileNotFoundException;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import java.util.Locale;

public class LaundryApp {
    /** Memuat berkas dan melaporkan apa yang terjadi; mengembalikan null jika tidak ada
yang dapat dimuat. */
    static List<Order> loadAndReport(String fileName) {
        List<String> errors = new ArrayList<>();
        try {
            List<Order> orders = OrderCsv.load(fileName, errors);
            System.out.println("Loaded " + orders.size() + " order(s) from " + fileName
                    + ", rejected " + errors.size() + " line(s).");
            for (String error : errors) {
                System.out.println("  rejected " + error);
            }
            return orders;
        } catch (FileNotFoundException e) {
            System.out.println("No such file: " + fileName + ". Starting with no orders.");
        } catch (OrderFormatException e) {
            System.out.println("Not an orders file (" + fileName + ", " + e.getMessage() +
").");
        } catch (IOException e) {
            System.out.println("Could not read " + fileName + ": " + e.getMessage());
        }
        return null;
    }

    public static void main(String[] args) {
        List<Order> orders = loadAndReport("orders.csv");
        if (orders == null) {
            return;
        }
        try {
            orders.add(OrderCsv.parseLine("L-011,Maya,IRON,3.0,yes", 0));
        } catch (OrderFormatException e) {
            System.out.println("Cannot add order: " + e.getMessage());
        }

        long revenue = 0;
        for (Order o : orders) {
            System.out.println(o);
            revenue += o.price();
        }
        System.out.println(String.format(Locale.US, "Revenue: Rp %,d", revenue));
```

Kode 5.53 (Lanjutan) *LaundryApp.java: memuat, melaporkan, menghitung harga, menyimpan, dan memuat ulang pesanan-pesanan.*

```
        try {
            OrderCsv.save("orders_clean.csv", orders);
            System.out.println("Saved " + orders.size() + " order(s) to orders_clean.csv");
        } catch (IOException e) {
            System.out.println("Could not save: " + e.getMessage());
        }
        loadAndReport("orders_clean.csv");                // perjalanan pulang-pergi: baca
kembali berkas buatan kita sendiri
        loadAndReport("missing.csv");
        loadAndReport("notes.txt");
    }
}
```

```
Loaded 4 order(s) from orders.csv, rejected 7 line(s).
  rejected line 5: customer name is empty
  rejected line 6: unknown service "DRY_CLEAN"
  rejected line 7: weight "abc" is not a number
  rejected line 8: weight -3 is outside 0-50 kg
  rejected line 9: duplicate id L-002
  rejected line 10: expected 5 fields, found 4
  rejected line 12: express must be yes or no, found "maybe"
L-001  Sari          WASH       3.50 kg     Rp    24,500
L-002  Budi Santoso  WASH_IRON  5.00 kg exp Rp    75,000
L-003  Rina          IRON       2.00 kg     Rp    10,000
L-010  Hendra        WASH      12.00 kg exp Rp   126,000
L-011  Maya          IRON       3.00 kg exp Rp    22,500
Revenue: Rp 258,000
Saved 5 order(s) to orders_clean.csv
Loaded 5 order(s) from orders_clean.csv, rejected 0 line(s).
No such file: missing.csv. Starting with no orders.
Not an orders file (notes.txt, line 1: header must be
id,customer,service,weight_kg,express).
```

Cocokkanlah keluaran itu dengan berkas masukannya. Baris 1 adalah *header* dan baris 2, 3, dan 4 sah. Baris 5 mempunyai nama pelanggan yang kosong. Baris 6 menyebut layanan yang tidak disediakan binatu itu. Berat pada baris 7 bukan bilangan. Berat pada baris 8 negatif. Baris 9 memakai ulang id `L-002`. Baris 10 hanya mempunyai empat *field*. Baris 11 kosong dan dilewati tanpa komentar. Baris 12 berisi `maybe` pada penanda ekspres. Baris 13 sah. Empat pesanan lolos, dan dengan pesanan Maya yang ditambahkan di loket jumlahnya menjadi lima, yang harga-harganya menurut Persamaan (5.4) berjumlah $24{,}500 + 75{,}000 + 10{,}000 + 126{,}000 + 22{,}500 = 258{,}000$ rupiah. Berkas yang disimpan adalah:

```
id,customer,service,weight_kg,express
L-001,Sari,WASH,3.5,no
L-002,Budi Santoso,WASH_IRON,5.0,yes
L-003,Rina,IRON,2.0,no
L-010,Hendra,WASH,12.0,yes
L-011,Maya,IRON,3.0,yes
```

Berkas itu termuat kembali tanpa kesalahan, dan itulah uji pulang-pergi (*round trip*) yang layak diterima setiap rutin penyimpanan. Perhatikan bahwa berat yang disimpan ditulis sebagai `2.0` dan `12.0`, sesuai cara `Double.toString` dan `str` di Python menuliskan sebuah `double`; berkasnya tetap sah, karena pemuat kita membaca kedua bentuk itu.

Kode 5.54 adalah aplikasi Python-nya. Satu-satunya perbedaan struktural adalah pemakaian `else` sesudah klausa-klausa `except`: laporan keberhasilan hanya dijalankan jika `load` tidak melempar apa-apa.

Kode 5.54 `laundry_app.py`: aplikasi Python-nya.

```
import order_csv
from order_errors import OrderFormatError


def load_and_report(file_name):
    """Memuat berkas dan melaporkan apa yang terjadi; mengembalikan None jika tidak ada
yang dapat dimuat."""
    errors = []
    try:
        orders = order_csv.load(file_name, errors)
    except FileNotFoundError:
        print("No such file: " + file_name + ". Starting with no orders.")
    except OrderFormatError as e:
        print("Not an orders file (" + file_name + ", " + str(e) + ").")
    except OSError as e:
        print("Could not read " + file_name + ": " + str(e))
    else:
        print("Loaded " + str(len(orders)) + " order(s) from " + file_name
              + ", rejected " + str(len(errors)) + " line(s).")
        for error in errors:
            print("  rejected " + error)
        return orders
    return None


def main():
    orders = load_and_report("orders.csv")
    if orders is None:
        return
    try:
        orders.append(order_csv.parse_line("L-011,Maya,IRON,3.0,yes", 0))
    except OrderFormatError as e:
        print("Cannot add order: " + str(e))

    revenue = 0
    for o in orders:
        print(o)
        revenue += o.price()
    print(f"Revenue: Rp {revenue:,}")
```

Kode 5.54 (Lanjutan) *laundry_app.py: aplikasi Python-nya.*

```
    try:
        order_csv.save("orders_clean.csv", orders)
        print("Saved " + str(len(orders)) + " order(s) to orders_clean.csv")
    except OSError as e:
        print("Could not save: " + str(e))
    load_and_report("orders_clean.csv")           # perjalanan pulang-pergi: baca kembali
berkas buatan kita sendiri
    load_and_report("missing.csv")
    load_and_report("notes.txt")


if __name__ == "__main__":
    main()
```

```
Loaded 4 order(s) from orders.csv, rejected 7 line(s).
  rejected line 5: customer name is empty
  rejected line 6: unknown service "DRY_CLEAN"
  rejected line 7: weight "abc" is not a number
  rejected line 8: weight -3 is outside 0-50 kg
  rejected line 9: duplicate id L-002
  rejected line 10: expected 5 fields, found 4
  rejected line 12: express must be yes or no, found "maybe"
L-001  Sari          WASH       3.50 kg     Rp    24,500
L-002  Budi Santoso  WASH_IRON  5.00 kg exp Rp    75,000
L-003  Rina          IRON       2.00 kg     Rp    10,000
L-010  Hendra        WASH      12.00 kg exp Rp   126,000
L-011  Maya          IRON       3.00 kg exp Rp    22,500
Revenue: Rp 258,000
Saved 5 order(s) to orders_clean.csv
Loaded 5 order(s) from orders_clean.csv, rejected 0 line(s).
No such file: missing.csv. Starting with no orders.
Not an orders file (notes.txt, line 1: header must be
id,customer,service,weight_kg,express).
```

Kedua keluaran itu sama baris demi baris, dan kedua program menulis berkas `orders_clean.csv` yang identik sampai ke tingkat bita. Untuk menjalankan versi Java, kompilasikan keempat kelas itu bersama-sama lalu jalankan `LaundryApp`:

```
javac Order.java OrderFormatException.java OrderCsv.java LaundryApp.java
java LaundryApp
```

dan untuk Python, dengan `order.py`, `order_errors.py`, dan `order_csv.py` berada di folder yang sama, `python3 laundry_app.py`. Di Windows, ketiklah `python` alih-alih `python3`. Berkas `README.md` di folder kode bab ini memberikan rinciannya untuk setiap program di bab ini.

5.10 Profitina dalam Praktik

Profitina, platform visibilitas AI di app.profitina.com, menghabiskan sebagian besar hidupnya untuk mengerjakan dalam skala besar apa yang dikerjakan Subbab 5.9 dalam skala kecil: menerima data yang tidak dikuasainya, memeriksanya, dan dengan sopan menolak apa yang tidak dapat dipakainya.

List yang ukurannya tidak diketahui siapa pun sebelumnya. Ketika Profitina memindai sebuah situs web untuk mengukur seberapa siap situs itu dikutip oleh mesin penjawab AI, ia tidak tahu sebelumnya berapa banyak halaman yang akan ditemukannya, berapa blok data terstruktur yang dimuat sebuah halaman, atau berapa banyak sinyal yang perlu dijelaskan. Setiap kumpulan ini adalah list yang dapat tumbuh, yaitu `list` milik Python sendiri di *back end* FastAPI, persis dengan alasan yang diberikan di Subbab 5.3.1. Larik berukuran tetap muncul di tempat-tempat yang ukurannya memang tetap, misalnya dalam komputasi numerik atas vektor-vektor skor.

CSV di tepian. Dasbor analitik historis menawarkan ekspor CSV atas tren skor mingguan setiap merek, agar pemilik merek dapat membukanya di aplikasi *spreadsheet*, dan ekspor data swalayan yang diwajibkan undang-undang pelindungan data pribadi Indonesia (UU PDP 27/2022) memberikan data pengguna sendiri dalam bentuk yang mudah dibawa. Menulis berkas-berkas seperti itu adalah Subbab 5.8.4 dan 5.8.5 yang diterapkan dengan cermat: pengodean UTF-8 yang eksplisit, sebuah baris *header*, dan *field-field* yang diberi tanda kutip dengan benar ketika memuat koma.

Kegagalan yang dilaporkan, bukan disembunyikan. Fitur-fitur AI platform itu berjalan sepenuhnya pada model lokal (Qwen2.5, dilayani oleh Ollama) sehingga data pengguna tidak

pernah meninggalkan server. Model lokal bisa saja tidak tersedia: prosesnya mungkin sedang dimulai ulang, atau mesinnya sedang sibuk. Ketika hal itu terjadi, aplikasi *mengatakannya*, alih-alih diam-diam menyajikan jawaban siap pakai seakan-akan berasal dari model. Itulah prinsip Subbab 5.5 dalam skala sebuah produk: eksepsi adalah informasi, dan menelannya berarti menipu pengguna. Dengan semangat yang sama, rilis bertahap baru melaporkan keberhasilan sesudah versi yang baru dipasang menjawab pemeriksaan kesehatannya, sehingga kegagalan dalam pemasangan tidak dapat menyamar sebagai keberhasilan.

Validasi sebelum percaya. Setiap tautan yang diunggah di komunitas Profitina disaring sebelum ditampilkan, dan unggahan yang gagal terkirim tidak hilang: kotak penyusun pesan memulihkan teksnya agar pengguna dapat mencoba lagi. Keduanya adalah pola Subbab 5.9, yang mempertahankan apa yang sah, menjelaskan apa yang tidak, dan tidak pernah membiarkan satu butir yang buruk meruntuhkan semua yang lain.

Aturan di balik semua ini adalah aturan rumah seluruh grup Profitina, dan aturan itu berlaku pula bagi program binatu kita: perangkat lunak memberi saran, penggunalah yang memutuskan. Sebuah program hanya dapat memberi saran yang baik jika ia berkata jujur tentang apa yang salah.

5.11 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya

Perulangan yang melangkah satu kali terlalu jauh. `for (int i = 0; i <= a.length; i++)` mengunjungi indeks `a.length` yang tidak ada dan melempar `ArrayIndexOutOfBoundsException`. *Cara menghindarinya:* syaratnya selalu `i < a.length` (atau `i < list.size()`); lebih baik lagi, pakailah `for` yang disempurnakan atau `for x in lst` di Python bila indeksnya tidak diperlukan.

Menulis mati sebuah panjang. `i < 5` berfungsi sampai lariknya membesar menjadi enam. *Cara menghindarinya:* pakailah `.length`, `size()`, atau `len()`.

Mencampuradukkan `length`, `length()`, dan `size()`. Larik mempunyai *field* `length`; string mempunyai metode `length()`; list mempunyai metode `size()`. *Cara menghindarinya:* ingatlah "larik *field*, string metode, list *size*", dan biarkan kompilator menangkap sisanya.

Menyalin referensi lalu menyebutnya salinan. `int[] b = a;` atau `b = a` membuat sebuah alias. *Cara menghindarinya:* pakailah `Arrays.copyOf` di Java, `a.copy()` atau `list(a)` di Python, dan gambarlah anak-anak panahnya seperti pada **Gambar 5.1** setiap kali Anda ragu.

Membandingkan larik atau list dengan operator yang salah. Di Java `==` membandingkan referensi; pakailah `Arrays.equals` untuk larik dan `equals` untuk list. Di Python `==` membandingkan isi dan `is` membandingkan identitas.

`remove(3)` pada list berisi `Integer`. Panggilan ini menghapus elemen pada indeks 3, bukan nilai 3. *Cara menghindarinya:* tulislah `remove(Integer.valueOf(3))` untuk nilainya.

Mengharapkan `-1` dari `index` di Python. `lst.index(x)` melempar `ValueError` jika `x` tidak ada. *Cara menghindarinya:* ujilah `x in lst` lebih dahulu, atau tangkap eksepsinya.

Sisa pindah baris sesudah `nextInt()`. `nextLine()` yang langsung menyusul `nextInt()` mengembalikan `""`. *Cara menghindarinya:* panggillah `nextLine()` sekali untuk menuntaskan baris bilangan itu, seperti pada **Kode 5.15**, atau bacalah setiap baris dengan `nextLine()` lalu konversikan dengan `Integer.parseInt`.

Mengulang `nextInt()` yang gagal tanpa membuang tokennya. Token yang buruk tetap tinggal di masukan dan perulangannya tidak pernah berakhir. *Cara menghindarinya:* panggillah `s.next()` di dalam blok `catch`, seperti pada **Kode 5.23**.

Menangkap `Exception` lebih dahulu, atau menangkapnya di mana-mana. Jaring umum yang mendahului jaring spesifik tidak dapat dikompilasi di Java dan, di Python, diam-diam menjadikan klausa spesifiknya kode mati. Jaring umum di tempat yang salah menyembunyikan *bug*. *Cara menghindarinya:* jaring spesifik lebih dahulu, dan jaring umum hanya di tingkat teratas sebuah perulangan atau program, tempat melanjutkan memang benar-benar tepat.

Blok `catch` yang kosong. `catch (IOException e) { }` atau `except: pass` membuang barang buktinya. *Cara menghindarinya:* setidak-tidaknya cetaklah pesannya; lebih baik lagi, tangani masalahnya atau biarkan ia berpropagasi.

Menangkap eksepsi tak terperiksa alih-alih membetulkan *bug*-nya. Membungkus `arr[i]` dengan `try`/`catch (ArrayIndexOutOfBoundsException e)` untuk menyembunyikan kesalahan selisih-satu hanya memindahkan *bug*-nya. *Cara menghindarinya:* betulkan indeksnya.

Mendeklarasikan `throws` pada `main` untuk membungkam kompilator. Eksepsi terperiksa itu lalu membuat program macet dengan *stack trace*. *Cara menghindarinya:* tangkaplah eksepsi terperiksa di metode yang tahu apa yang harus disampaikan kepada pengguna.

Memakai variabel dari dalam blok `try` sesudah blok itu. Variabel itu tidak ada di sana. *Cara menghindarinya:* kerjakan pekerjaannya di dalam `try`, atau deklarasikan variabelnya sebelum `try`.

Lupa menutup berkas. Data yang ditulis mungkin tidak pernah sampai ke disk, dan pegangan berkas bisa habis. *Cara menghindarinya:* `try`-with-resources di Java, `with` di Python, setiap kali.

Bergantung pada pengodean karakter bawaan. Berkas yang ditulis di satu komputer menjadi kacau di komputer lain. *Cara menghindarinya:* serahkan `StandardCharsets.UTF_8` atau `encoding="utf-8"` secara eksplisit.

`split(",")` di Java yang kehilangan *field* kosong di ujung. *Cara menghindarinya:* `split(",", -1)`, dan periksa cacah *field* sebelum memakai satu pun di antaranya.

Metode yang namanya berbohong. `isExceeded` pada **Kode 5.29** mengembalikan `true` ketika nilainya *tidak* terlampaui. *Cara menghindarinya:* namailah metode sesuai dengan apa yang dikembalikannya; metode yang hanya memeriksa dan melempar eksepsi dapat dinamai `checkGrayscale` dan tidak mengembalikan apa-apa.

Memakai eksepsi untuk alur kendali biasa. Melempar eksepsi untuk keluar dari perulangan, atau untuk mengembalikan hasil yang normal, itu lambat dan menyesatkan. *Cara menghindarinya:* simpanlah eksepsi untuk kesalahan.

5.12 Ringkasan Bab

- **Larik** adalah deretan slot bertipe sama yang panjangnya tetap, berindeks 0 sampai $n-1$. Menurut Persamaan (5.1), slot mana pun dapat dicapai secara langsung. `new T[n]` mengisinya dengan nilai-nilai bawaan; `{a, b, c}` adalah bentuk singkatnya. Telusurilah dengan `i < a.length` atau dengan `for` yang disempurnakan. Indeks n melempar `ArrayIndexOutOfBoundsException` (`IndexError` di Python).

- Variabel larik menyimpan sebuah **referensi**. Penugasan membuat alias; `Arrays.copyOf` dan `System.arraycopy` membuat salinan. Larik tidak dapat tumbuh; membuat larik yang lebih besar lalu menyalin adalah satu-satunya cara.
- `main(String[] args)` menerima **argumen baris perintah**; Python mempunyai `sys.argv`, dengan indeks 0 berisi nama skrip.
- **List** (`ArrayList<T>`, `list` Python) tumbuh dengan sendirinya. Operasi intinya adalah *add*/*append*, *get*/`[i]`, *set*, *size*/`len`, *contains*/`in`, *indexOf*/`index`, serta *remove* menurut posisi atau menurut nilai (**Tabel 5.1**). Tumbuh secara geometris membuat penambahan di akhir murah secara rata-rata (Persamaan (5.3)), sedangkan tumbuh satu slot setiap kali berongkos kuadratik (Persamaan (5.2)). List bilangan di Java memerlukan tipe pembungkus seperti `Integer`.
- **Masukan**: `Scanner` di Java membaca baris (`nextLine`) dan token bertipe (`nextInt`, `nextDouble`); waspadalah terhadap sisa pindah baris. `input()` di Python membaca satu baris; konversikan dengan `int()` atau `float()`. Alihkan sebuah berkas ke masukan standar dengan `<` agar program dapat dijalankan ulang secara persis sama.
- **Eksepsi** adalah objek yang menguraikan sebuah kegagalan. Eksepsi **dilemparkan** (`throw`, `raise`), **berpropagasi** kembali melalui panggilan-panggilan, dan **ditangkap** oleh `catch`/`except` pertama yang cocok. Bacalah *stack trace* Java dari atas ke bawah untuk mengetahui tipenya lalu carilah kelas Anda sendiri; bacalah *traceback* Python dari bawah ke atas.
- Eksepsi membentuk sebuah **hierarki kelas**; `catch` untuk sebuah kelas juga menangkap subkelas-subkelasnya, jadi urutkan klausa-klausanya dari yang spesifik ke yang umum. `finally` selalu dijalankan. Python menambahkan `else`.
- Di Java, subkelas-subkelas `RuntimeException` adalah eksepsi **tak terperiksa**, biasanya kesalahan pemrograman yang harus dibetulkan; eksepsi lain seperti `IOException` adalah eksepsi **terperiksa** dan harus **ditangkap atau dideklarasikan** dengan `throws`. Python tidak mengenal eksepsi terperiksa; dokumentasikan eksepsi di dalam *docstring*.
- **Eksepsi buatan sendiri** memperluas `Exception` (terperiksa) atau `RuntimeException` (tak terperiksa) di Java, dan `Exception` atau salah satu subkelasnya di Python; eksepsi itu membawa pesan, data tambahan, dan sebuah **penyebab** (`super(message, cause)`, `raise ... from ...`).
- **Berkas**: *stream* membawa bita, *reader* dan *writer* membawa karakter, dan penyanggaan membuat keduanya efisien. Selalu tutup apa yang Anda buka, dengan **`try`-with-resources** atau **`with`**. Nyatakan UTF-8. Susunlah teks panjang dengan `StringBuilder` atau `"".join`.
- **CSV**: satu baris tabel per baris teks, *field* dipisahkan koma, biasanya dengan sebuah *header*. Pakailah `split(",", -1)` di Java, modul `csv` di Python, dan validasi setiap *field*.
- **Profitina Laundry** kini memuat pesanan dari CSV, menolak setiap baris yang buruk beserta nomor baris dan alasannya, terus melaju melewatinya, berhenti dengan rapi bila berkasnya tidak ada atau asing, dan menyimpan berkas bersih yang dapat dimuat ulang tanpa kesalahan.

5.13 Soal Latihan

Soal Latihan 5.1. (a) Apa yang terjadi jika, sesudah membuat `String[] fruitArray = new String[5]`, kita menulis `fruitArray[5] = "durian";`? Mengapa? (b) Tulislah pernyataan yang membuat larik `int` berukuran sepuluh, di Java dan di Python. (c) Nilai-nilai apa yang dimuat `new boolean[4]`, dan apa yang dimuat `new String[2]`?

Soal Latihan 5.2. Tulislah sebuah metode yang mengembalikan buah kesepuluh dari sebuah `ArrayList<String>` (fungsi Python untuk sebuah `list`). Ingatlah bahwa mungkin saja tidak ada

sepuluh buah di dalam list itu, dan tangani kasus tersebut tanpa eksepsi apa pun. Ujilah pada list berisi 3 dan 10 buah.

Soal Latihan 5.3. Untuk setiap keadaan berikut, pilihlah larik atau list dan berikan alasannya dalam satu kalimat: (a) tujuh total harian dalam satu pekan; (b) pesanan-pesanan yang diterima hari ini; (c) papan *tic-tac-toe* 3 × 3; (d) para pelanggan yang menjawab sebuah survei; (e) 256 hitungan histogram *grayscale*; (f) baris-baris sebuah berkas teks yang panjangnya tidak diketahui.

Soal Latihan 5.4. Tanpa memakai `ArrayList` atau `append`, tulislah metode yang menambahkan sebuah nilai ke dalam larik dengan membuat larik baru yang satu slot lebih panjang. Tambahkan 1000 nilai dan hitunglah berapa kali penyalinan elemen dilakukan. Kemudian implementasikan strategi penggandaan dari Subbab 5.3.3, dengan menyimpan ukuran secara terpisah, dan hitunglah lagi. Bandingkan kedua hitungan itu dengan Persamaan (5.2) dan Persamaan (5.3).

Soal Latihan 5.5. Tanpa menjalankannya, nyatakan dengan tepat apa yang dicetak metode Java ini untuk `f(0)` dan untuk `f(1)`, dan apa yang dikembalikannya:

```
static int f(int x) {
    try {
        System.out.println("A");
        int y = 10 / x;
        System.out.println("B");
        return y;
    } catch (ArithmeticException e) {
        System.out.println("C");
        return -1;
    } finally {
        System.out.println("D");
    }
}
```

Soal Latihan 5.6. Golongkan setiap eksepsi Java berikut sebagai terperiksa atau tak terperiksa, dan katakan apakah eksepsi itu biasanya menandakan kesalahan pemrograman atau masalah dari luar: `NullPointerException`, `FileNotFoundException`, `NumberFormatException`, `IOException`, `ClassCastException`, `InputMismatchException`, `ArithmeticException`, `OrderFormatException` (Subbab 5.9).

Soal Latihan 5.7. Tulislah metode `getSomeNumbers(int first, int second)` yang melempar `IllegalArgumentException` dengan pesan “The numbers you’ve entered are not acceptable.” ketika `first + second >= 20`, dan jika tidak, mencetak “Your numbers are fine. Well done.” Panggillah metode itu dari `main` untuk pasangan (3,4), (15,5), (−30,45), dan (9,10), sambil menangkap eksepsinya. Tulislah versi Python-nya dengan `ValueError`.

Soal Latihan 5.8. Tulislah program yang membaca token-token yang dipisahkan spasi dari masukan standar sampai bertemu token `99`, melewati setiap token yang bukan bilangan bulat, dan akhirnya mencetak berapa bilangan bulat yang terbaca, berapa token yang dilewati, dan rata-rata bilangan-bilangan bulat itu dengan dua angka di belakang titik desimal. Program tidak boleh macet jika masukan berakhir tanpa `99` atau jika tidak ada bilangan bulat yang terbaca. Pakailah masukan `4 x 8 / 15 3.5 -2 / six 99 1000` (tiga baris, garis miring menandai pindah baris).

Soal Latihan 5.9. Temukan semua kesalahan dalam program ini dan katakan kesalahan mana yang dilaporkan kompilator:

```
public class Buggy {
    public static void main(String[] args) {
        String[] prices = {"7000", "5000", "ten thousand"};
        int total = 0;
        for (int i = 0; i <= prices.length; i++) {
            try {
                total += Integer.parseInt(prices[i]);
            } catch (Exception e) {
            } catch (NumberFormatException e) {
                System.out.println("bad price " + prices[i]);
            }
        }
        System.out.println("total = " + total);
    }
}
```

Soal Latihan 5.10. Profitina Laundry menjual kartu prabayar. Tulislah kelas `PrepaidCard` dengan seorang pemilik dan saldo dalam rupiah, serta metode `pay(long amount)` yang melempar `IllegalArgumentException` untuk jumlah yang tidak positif dan eksepsi *terperiksa* buatan sendiri `InsufficientBalanceException` ketika saldonya terlalu kecil, dengan membawa informasi berapa yang kurang. Pembayaran yang gagal harus membiarkan saldo tidak tersentuh. Ujilah dengan saldo 50,000 dan tagihan 24,500, 30,000, 20,000, dan −5. Tulislah juga versi Python-nya.

Soal Latihan 5.11. Dengan memakai `OrderCsv` (atau `order_csv`) dari Subbab 5.9, muatlah `orders.csv`, hitunglah cacah pesanan, kilogram, dan pendapatan per layanan, lalu tuliskan ke `summary.csv` dengan *header* `service,orders,kg,revenue`.

Soal Latihan 5.12. Versi pertama **Kode 5.23** menangani masukan yang buruk dengan menulis `s = new Scanner(System.in);` di dalam blok `catch` alih-alih `s.next();`. Jelaskan mengapa cara ini tampak berhasil di papan ketik tetapi gagal ketika masukannya dialihkan dari sebuah berkas.

Soal Latihan 5.13. Tulislah program `WordCount` (di Python `word_count.py`) yang menerima sebuah nama berkas sebagai satu-satunya argumen baris perintah dan mencetak cacah baris, kata, dan karakter di dalam berkas itu (tanpa menghitung pindah baris). Tanpa argumen, program mencetak pesan cara pakai; untuk berkas yang tidak ada, program mencetak “no such file:” diikuti namanya. Cobalah pada `notes.txt`.

5.14 Penyelesaian Soal Latihan

Penyelesaian 5.1. (a) Program melempar `ArrayIndexOutOfBoundsException: Index 5 out of bounds for length 5`. Larik sepanjang 5 mempunyai indeks 0 sampai 4 (**Gambar 5.1**); indeks 5 berarti slot keenam, dan larik Java tidak pernah tumbuh. (b) Java: `int[] tenInts = new int[10];` Python: `ten_ints = [0] * 10`. (c) `new boolean[4]` memuat empat `false` dan `new String[2]` memuat dua `null`: setiap larik Java yang baru diisi dengan nilai bawaan tipe elemennya.

Penyelesaian 5.2. Elemen kesepuluh berindeks 9, dan elemen itu hanya ada bila ukurannya sekurang-kurangnya 10, sehingga kita menguji ukurannya sebelum memanggil `get`. **Kode 5.55** dan **Kode 5.56** memberikan penyelesaiannya.

Kode 5.55 `TenthFruit.java`: buah kesepuluh, jika ada.

```java
import java.util.ArrayList;

public class TenthFruit {
    static String tenthFruit(ArrayList<String> fruit) {
        if (fruit.size() >= 10) {                // buah ke-10 berada di indeks 9
            return fruit.get(9);
        }
        return "(only " + fruit.size() + " fruit, so there is no 10th)";
    }

    public static void main(String[] args) {
        ArrayList<String> fruit = new ArrayList<>();
        String[] names = {"apel", "belimbing", "ceri", "duku", "durian",
                          "jambu", "jeruk", "kedondong", "kelengkeng", "mangga"};
        for (String name : names) {
            fruit.add(name);
            if (fruit.size() == 3 || fruit.size() == 10) {
                System.out.println(fruit.size() + " fruit: " + tenthFruit(fruit));
            }
        }
    }
}
```

```
3 fruit: (only 3 fruit, so there is no 10th)
10 fruit: mangga
```

Kode 5.56 `tenth_fruit.py`: fungsi yang sama di Python.

```python
def tenth_fruit(fruit):
    if len(fruit) >= 10:                    # buah ke-10 berada di indeks 9
        return fruit[9]
    return "(only " + str(len(fruit)) + " fruit, so there is no 10th)"


def main():
    fruit = []
    names = ["apel", "belimbing", "ceri", "duku", "durian",
             "jambu", "jeruk", "kedondong", "kelengkeng", "mangga"]
    for name in names:
        fruit.append(name)
        if len(fruit) == 3 or len(fruit) == 10:
            print(str(len(fruit)) + " fruit: " + tenth_fruit(fruit))


if __name__ == "__main__":
    main()
```

```
3 fruit: (only 3 fruit, so there is no 10th)
10 fruit: mangga
```

Penyelesaian 5.3. (a) Larik: selalu tepat tujuh nilai. (b) List: tidak ada yang tahu berapa banyak pesanan yang akan datang. (c) Larik dua dimensi: bentuknya tetap dan menyerupai kisi. (d) List: cacah responden tidak diketahui. (e) Larik (`int[256]`): cacah tingkat keabuan yang mungkin sudah tetap. (f) List: panjangnya tidak diketahui sampai seluruh berkas selesai dibaca.

Penyelesaian 5.4. **Kode 5.57** dan **Kode 5.58** mengimplementasikan kedua strategi dan menghitung setiap elemen yang disalin.

Kode 5.57 `GrowArray.java`: tumbuh satu slot lawan penggandaan.

```java
import java.util.Arrays;

public class GrowArray {
    static long copies = 0;                       // berapa slot yang sudah kita salin sejauh ini

    // Strategi 1: sediakan tempat untuk tepat satu nilai lagi setiap kali.
    static int[] appendOne(int[] a, int value) {
        int[] b = new int[a.length + 1];
        for (int i = 0; i < a.length; i++) {
            b[i] = a[i];
            copies++;
        }
        b[a.length] = value;
        return b;
    }

    public static void main(String[] args) {
        int n = 1000;
        int[] a = new int[0];
        for (int v = 1; v <= n; v++) {
            a = appendOne(a, v);
        }
        System.out.println("grow by one:  n = " + n + ", copies = " + copies);

        // Strategi 2: gandakan kapasitas saat penuh, dan ingat berapa yang terpakai.
        copies = 0;
        int[] data = new int[1];
        int size = 0;
        for (int v = 1; v <= n; v++) {
            if (size == data.length) {
                copies += size;
                data = Arrays.copyOf(data, 2 * data.length);
            }
            data[size] = v;
            size++;
        }
        System.out.println("doubling:     n = " + n + ", copies = " + copies
                + ", capacity = " + data.length);
    }
}
```

```
grow by one:  n = 1000, copies = 499500
doubling:     n = 1000, copies = 1023, capacity = 1024
```

Kode 5.58 `grow_array.py`: percobaan yang sama di Python.

```
copies = 0                                    # berapa slot yang sudah kita salin sejauh
ini


def append_one(a, value):
    """Strategi 1: sediakan tempat untuk tepat satu nilai lagi setiap kali."""
    global copies
    b = [0] * (len(a) + 1)
    for i in range(len(a)):
        b[i] = a[i]
        copies += 1
    b[len(a)] = value
    return b


def main():
    global copies
    n = 1000
    a = []
    for v in range(1, n + 1):
        a = append_one(a, v)
    print("grow by one:  n = " + str(n) + ", copies = " + str(copies))

    # Strategi 2: gandakan kapasitas saat penuh, dan ingat berapa yang terpakai.
    copies = 0
    data = [0] * 1
    size = 0
    for v in range(1, n + 1):
        if size == len(data):
            copies += size
            data = data + [0] * len(data)
        data[size] = v
        size += 1
    print("doubling:     n = " + str(n) + ", copies = " + str(copies)
          + ", capacity = " + str(len(data)))


if __name__ == "__main__":
    main()
```

```
grow by one:  n = 1000, copies = 499500
doubling:     n = 1000, copies = 1023, capacity = 1024
```

Tumbuh satu slot setiap kali menyalin $1000 \times 999/2 = 499{,}500$ elemen, persis seperti yang diramalkan Persamaan (5.2). Penggandaan menyalin $1 + 2 + \cdots + 512 = 1023 < 2 \times 1000$ elemen, seperti yang diramalkan Persamaan (5.3), dengan harga kapasitas 1024 untuk 1000 nilai. Untuk $n = 10^6$ strategi pertama akan menyalin sekitar 5×10^{11} elemen dan strategi kedua kurang dari 2×10^6.

Penyelesaian 5.5. Untuk `f(0)`: `A` tercetak; `10 / 0` melempar `ArithmeticException`, sehingga `B` terlewati; `catch` mencetak `C` dan bersiap mengembalikan -1; `finally` mencetak `D`; metode mengembalikan -1. Keluarannya `A C D` pada tiga baris. Untuk `f(1)`: `A`, lalu `y = 10`, lalu `B`; `return` menyiapkan 10; `finally` mencetak `D`; metode mengembalikan 10. Keluarannya `A B D`. Blok `finally` dijalankan pada kedua kasus, sesudah nilai yang akan dikembalikan dihitung dan sebelum nilai itu benar-benar dikembalikan (**Gambar 5.6**).

Penyelesaian 5.6. Tak terperiksa (memperluas `RuntimeException`), biasanya kesalahan pemrograman: `NullPointerException`, `NumberFormatException` (walaupun sering berasal dari masukan yang buruk, dan dalam hal itu harus ditangkap), `ClassCastException`, `InputMismatchException` (sekali lagi dari masukan), `ArithmeticException`. Terperiksa, masalah dari luar: `FileNotFoundException`, `IOException`, dan `OrderFormatException`, yang langsung

memperluas `Exception` (**Gambar 5.4**). Perhatikan bahwa "tak terperiksa" tidak berarti "tidak pernah ditangkap": eksepsi yang disebabkan masukan pengguna, seperti kedua kegagalan konversi tadi, hendaknya ditangkap di dekat masukannya.

Penyelesaian 5.7. Kode 5.59 dan **Kode 5.60**. Pasangan (−30,45) berjumlah 15 dan diterima; aturannya menyangkut jumlahnya, bukan nilai masing-masing.

Kode 5.59 `SomeNumbers.java`: melempar `IllegalArgumentException` dan menangkapnya di pemanggil.

```
public class SomeNumbers {
    public static void getSomeNumbers(int first, int second) {
        if (first + second >= 20) {
            throw new IllegalArgumentException("The numbers you've entered are not
acceptable.");
        }
        System.out.println("Your numbers are fine. Well done.");
    }

    public static void main(String[] args) {
        int[][] pairs = {{3, 4}, {15, 5}, {-30, 45}, {9, 10}};
        for (int[] p : pairs) {
            System.out.print(p[0] + " + " + p[1] + ": ");
            try {
                getSomeNumbers(p[0], p[1]);
            } catch (IllegalArgumentException e) {
                System.out.println("refused, " + e.getMessage());
            }
        }
    }
}
```

```
3 + 4: Your numbers are fine. Well done.
15 + 5: refused, The numbers you've entered are not acceptable.
-30 + 45: Your numbers are fine. Well done.
9 + 10: Your numbers are fine. Well done.
```

Kode 5.60 `some_numbers.py`: hal yang sama dengan `ValueError`.

```
def get_some_numbers(first, second):
    if first + second >= 20:
        raise ValueError("The numbers you've entered are not acceptable.")
    print("Your numbers are fine. Well done.")


def main():
    pairs = [(3, 4), (15, 5), (-30, 45), (9, 10)]
    for first, second in pairs:
        print(str(first) + " + " + str(second) + ": ", end="")
        try:
            get_some_numbers(first, second)
        except ValueError as e:
            print("refused, " + str(e))


if __name__ == "__main__":
    main()
```

```
3 + 4: Your numbers are fine. Well done.
15 + 5: refused, The numbers you've entered are not acceptable.
-30 + 45: Your numbers are fine. Well done.
9 + 10: Your numbers are fine. Well done.
```

Penyelesaian 5.8. Versi Java, **Kode 5.61**, memakai pola `hasNext`/`nextInt`/`next` dari **Kode 5.23**; versi Python, **Kode 5.62**, membaca seluruh masukan lalu memecahnya menjadi token-token. Keduanya melindungi pembagian dari cacah nol. Bilangan bulat yang terbaca adalah $4, 8, 15, -2$, sehingga rata-ratanya $25/4 = 6.25$; `x`, `3.5`, dan `six` dilewati, dan `1000` tidak pernah terbaca karena datang sesudah `99`.

Kode 5.61 `AverageInput.java`: perata-rataan bilangan bulat yang diketik secara tangguh.

```java
import java.util.InputMismatchException;
import java.util.Scanner;

public class AverageInput {
    public static void main(String[] args) {
        Scanner s = new Scanner(System.in);
        long sum = 0;
        int count = 0;
        int skipped = 0;
        boolean sawStop = false;
        while (s.hasNext()) {
            try {
                int v = s.nextInt();
                if (v == 99) {
                    sawStop = true;
                    break;
                }
                sum += v;
                count++;
            } catch (InputMismatchException e) {
                s.next();                          // buang token yang buruk
                skipped++;
            }
        }
        if (!sawStop) {
            System.out.println("(input ended without 99)");
        }
        System.out.println("numbers: " + count + ", skipped: " + skipped);
        if (count > 0) {
            System.out.printf("average: %.2f%n", (double) sum / count);
        } else {
            System.out.println("average: undefined, no numbers were read");
        }
    }
}
```

```
4 x 8
15 3.5 -2
six 99 1000
```

```
numbers: 4, skipped: 3
average: 6.25
```

Kode 5.62 `average_input.py`: hal yang sama di Python.

```python
import sys


def main():
    total = 0
    count = 0
    skipped = 0
    saw_stop = False
    tokens = sys.stdin.read().split()         # setiap token yang dipisahkan spasi
    for token in tokens:
        try:
            v = int(token)
        except ValueError:
            skipped += 1                      # buang token yang buruk
            continue
        if v == 99:
            saw_stop = True
            break
        total += v
        count += 1
    if not saw_stop:
        print("(input ended without 99)")
    print("numbers: " + str(count) + ", skipped: " + str(skipped))
    if count > 0:
        print(f"average: {total / count:.2f}")
    else:
        print("average: undefined, no numbers were read")


if __name__ == "__main__":
    main()
```

```
numbers: 4, skipped: 3
average: 6.25
```

Penyelesaian 5.9. Ada tiga kesalahan. (1) Kompilator melaporkan urutan `catch`:

```
Buggy.java:9: error: exception NumberFormatException has already been caught
            } catch (NumberFormatException e) {
              ^
1 error
```

`catch (Exception e)` yang umum datang lebih dahulu, sehingga klausa yang spesifik tidak akan pernah bisa berjalan; tukarlah keduanya. (2) Blok `catch` yang pertama kosong: sesudah urutannya dibetulkan pun, blok itu masih akan diam-diam menelan `ArrayIndexOutOfBoundsException` dari kesalahan ketiga. (3) Syarat perulangan `i <= prices.length` mengunjungi indeks 3, yang tidak ada; syarat itu harus `i < prices.length`. Perulangan yang sudah dibetulkan mencetak “bad price ten thousand” dan “total = 12000”.

Penyelesaian 5.10. Pemeriksaan saldo yang tidak mencukupi datang *sebelum* pengurangan, sehingga pembayaran yang gagal tidak mengubah apa pun. Eksepsinya terperiksa, sehingga kompilator memaksa `main` menanganinya.

Kode 5.63 `PrepaidCard.java`: eksepsi terperiksa buatan sendiri untuk saldo yang tidak mencukupi.

```
public class PrepaidCard {
    // Eksepsi terperiksa: pemanggil harus bersiap menghadapi kartu yang saldonya habis.
    static class InsufficientBalanceException extends Exception {
        private final long shortBy;

        InsufficientBalanceException(long balance, long amount) {
            super("balance Rp " + balance + " is short of Rp " + amount);
            this.shortBy = amount - balance;
        }

        long getShortBy() {
            return shortBy;
        }
    }

    private final String owner;
    private long balance;

    PrepaidCard(String owner, long balance) {
        this.owner = owner;
        this.balance = balance;
    }

    void pay(long amount) throws InsufficientBalanceException {
        if (amount <= 0) {
            throw new IllegalArgumentException("amount must be positive: " + amount);
        }
        if (amount > balance) {
            throw new InsufficientBalanceException(balance, amount);   // saldo tidak
tersentuh
        }
        balance -= amount;
    }

    public static void main(String[] args) {
        PrepaidCard card = new PrepaidCard("Sari", 50000);
        long[] bills = {24500, 30000, 20000, -5};
        for (long bill : bills) {
            try {
                card.pay(bill);
                System.out.println("paid " + bill + ", balance now " + card.balance);
            } catch (InsufficientBalanceException e) {
                System.out.println("declined: " + e.getMessage()
                        + " (short by " + e.getShortBy() + ")");
            } catch (IllegalArgumentException e) {
                System.out.println("rejected: " + e.getMessage());
            }
        }
        System.out.println(card.owner + " keeps Rp " + card.balance);
    }
}
```

Kode 5.63 (Lanjutan) *PrepaidCard.java: eksepsi terperiksa buatan sendiri untuk saldo yang tidak mencukupi.*

```
paid 24500, balance now 25500
declined: balance Rp 25500 is short of Rp 30000 (short by 4500)
paid 20000, balance now 5500
rejected: amount must be positive: -5
Sari keeps Rp 5500
```

Kode 5.64 `prepaid_card.py`: kartu yang sama di Python.

```
class InsufficientBalanceError(Exception):
    """Dilempar bila kartu tidak dapat menutup sebuah pembayaran."""

    def __init__(self, balance, amount):
        super().__init__("balance Rp " + str(balance) + " is short of Rp " + str(amount))
        self.short_by = amount - balance


class PrepaidCard:
    def __init__(self, owner, balance):
        self.owner = owner
        self.balance = balance

    def pay(self, amount):
        """Melempar ValueError untuk jumlah yang tidak positif, InsufficientBalanceError
jika saldo kurang."""
        if amount <= 0:
            raise ValueError("amount must be positive: " + str(amount))
        if amount > self.balance:
            raise InsufficientBalanceError(self.balance, amount)   # saldo tidak tersentuh
        self.balance -= amount


def main():
    card = PrepaidCard("Sari", 50000)
    for bill in [24500, 30000, 20000, -5]:
        try:
            card.pay(bill)
            print("paid " + str(bill) + ", balance now " + str(card.balance))
        except InsufficientBalanceError as e:
            print("declined: " + str(e) + " (short by " + str(e.short_by) + ")")
        except ValueError as e:
            print("rejected: " + str(e))
    print(card.owner + " keeps Rp " + str(card.balance))


if __name__ == "__main__":
    main()
```

```
paid 24500, balance now 25500
declined: balance Rp 25500 is short of Rp 30000 (short by 4500)
paid 20000, balance now 5500
rejected: amount must be positive: -5
Sari keeps Rp 5500
```

Kode 5.63 dan **Kode 5.64** sama-sama menolak tagihan 30,000 dengan saldo tetap 25,500, lalu menerima 20,000.

Penyelesaian 5.11. **Kode 5.65** memakai ulang pemuatnya dan menyimpan satu pencacah per layanan dalam tiga larik paralel yang diindeks seperti `Order.SERVICES`. *Multi-catch* `IOException | OrderFormatException` menangani kedua kegagalan pemuatan dengan satu pesan.

Kode 5.65 `ServiceSummary.java`: cacah pesanan, kilogram, dan pendapatan per layanan.

```java
import java.io.IOException;
import java.io.PrintWriter;
import java.io.FileWriter;
import java.nio.charset.StandardCharsets;
import java.util.ArrayList;
import java.util.List;

public class ServiceSummary {
    public static void main(String[] args) {
        List<Order> orders;
        try {
            orders = OrderCsv.load("orders.csv", new ArrayList<>());
        } catch (IOException | OrderFormatException e) {    // satu penangan, dua tipe
            System.out.println("Cannot load orders: " + e.getMessage());
            return;
        }
        String[] services = Order.SERVICES;
        int[] count = new int[services.length];              // satu pencacah per layanan
        double[] kg = new double[services.length];
        long[] revenue = new long[services.length];
        for (Order o : orders) {
            for (int i = 0; i < services.length; i++) {
                if (services[i].equals(o.getService())) {
                    count[i]++;
                    kg[i] += o.getWeightKg();
                    revenue[i] += o.price();
                }
            }
        }
        try (PrintWriter out = new PrintWriter(
                new FileWriter("summary.csv", StandardCharsets.UTF_8))) {
            out.println("service,orders,kg,revenue");
            for (int i = 0; i < services.length; i++) {
                out.println(services[i] + "," + count[i] + "," + kg[i] + "," + revenue[i]);
                System.out.println(services[i] + "," + count[i] + "," + kg[i] + "," +
revenue[i]);
            }
        } catch (IOException e) {
            System.out.println("Cannot write summary: " + e.getMessage());
        }
    }
}
```

```
WASH,2,15.5,150500
IRON,1,2.0,10000
WASH_IRON,1,5.0,75000
```

Kode 5.66 `service_summary.py`: ringkasan yang sama di Python.

```
import order_csv
from order import RATES
from order_errors import OrderFormatError


def main():
    try:
        orders = order_csv.load("orders.csv", [])
    except (OSError, OrderFormatError) as e:  # satu penangan, dua tipe
        print("Cannot load orders: " + str(e))
        return
    services = list(RATES)                    # WASH, IRON, WASH_IRON, dengan urutan itu
    count = [0] * len(services)               # satu pencacah per layanan
    kg = [0.0] * len(services)
    revenue = [0] * len(services)
    for o in orders:
        i = services.index(o.service)
        count[i] += 1
        kg[i] += o.weight_kg
        revenue[i] += o.price()
    try:
        with open("summary.csv", "w", encoding="utf-8") as out:
            out.write("service,orders,kg,revenue\n")
            for i in range(len(services)):
                row = services[i] + "," + str(count[i]) + "," + str(kg[i]) + "," +
str(revenue[i])
                out.write(row + "\n")
                print(row)
    except OSError as e:
        print("Cannot write summary: " + str(e))


if __name__ == "__main__":
    main()
```

```
WASH,2,15.5,150500
IRON,1,2.0,10000
WASH_IRON,1,5.0,75000
```

Keempat pesanan yang sah di `orders.csv` menghasilkan `WASH` 24,500 + 126,000 = 150,500, `IRON` 10,000, dan `WASH_IRON` 75,000. Baris-baris yang sama ditulis ke `summary.csv`. Pada **Kode 5.66**, `list(RATES)` memberikan kunci-kunci *dictionary* itu menurut urutan penyisipannya.

Penyelesaian 5.12. `Scanner` tidak membaca satu token setiap kali dari sistem operasi; ia membaca satu bongkah besar masukan ke dalam *buffer*-nya sendiri lalu menyerahkan token-token dari *buffer* itu. Ketika masukan datang dari berkas, `Scanner` yang pertama kemungkinan besar sudah membaca *seluruh* berkas ke dalam *buffer*-nya. `Scanner` baru yang dibangun di atas `System.in` yang sama mulai membaca dari tempat sistem operasi berhenti, yaitu akhir berkas, sehingga ia tidak menemukan apa-apa dan `nextInt()` pertamanya melempar `NoSuchElementException`; segala sesuatu yang masih berada di *buffer* lama hilang. Di papan ketik, masukan tiba satu baris setiap kali, sehingga *buffer* lama biasanya hanya memuat baris yang buruk itu dan kesalahannya luput dari perhatian. Perbaikan yang tangguh adalah mempertahankan satu `Scanner` dan membuang token yang buruk dengan `next()`.

Penyelesaian 5.13. **Kode 5.67** membuka berkas dengan `try`-with-resources dan melaporkan kedua kasus kegagalan secara terpisah. `split("\\s+")` memecah di setiap deretan spasi; barisnya dirapikan dengan `trim` lebih dahulu, dan baris kosong tidak dihitung mempunyai kata. `split()` tanpa argumen di Python mengerjakan semua itu sendiri (**Kode 5.68**).

Kode 5.67 `WordCount.java`: menghitung baris, kata, dan karakter sebuah berkas yang namanya diberikan di baris perintah.

```
import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.nio.charset.StandardCharsets;

public class WordCount {
    public static void main(String[] args) {
        if (args.length != 1) {
            System.out.println("usage: java WordCount <file>");
            return;
        }
        int lines = 0;
        int words = 0;
        int chars = 0;
        try (BufferedReader in = new BufferedReader(
                new FileReader(args[0], StandardCharsets.UTF_8))) {
            String line;
            while ((line = in.readLine()) != null) {
                lines++;
                chars += line.length();
                String trimmed = line.trim();
                if (!trimmed.isEmpty()) {
                    words += trimmed.split("\\s+").length;   // satu spasi atau lebih
                }
            }
        } catch (FileNotFoundException e) {
            System.out.println("no such file: " + args[0]);
            return;
        } catch (IOException e) {
            System.out.println("cannot read " + args[0] + ": " + e.getMessage());
            return;
        }
        System.out.println(lines + " lines, " + words + " words, "
                + chars + " characters (line breaks not counted) in " + args[0]);
    }
}
```

Menjalankan `java WordCount notes.txt`, lalu `java WordCount nope.txt`, lalu `java WordCount` menghasilkan:

```
3 lines, 17 words, 99 characters (line breaks not counted) in notes.txt
no such file: nope.txt
usage: java WordCount <file>
```

Kode 5.68 `word_count.py`: hitungan yang sama di Python.

```python
import sys


def main():
    if len(sys.argv) != 2:
        print("usage: python word_count.py <file>")
        return
    name = sys.argv[1]
    lines = words = chars = 0
    try:
        with open(name, encoding="utf-8") as f:
            for line in f:
                line = line.rstrip("\n")
                lines += 1
                chars += len(line)
                words += len(line.split())    # split() tanpa argumen: deretan spasi berapa
pun panjangnya
    except FileNotFoundError:
        print("no such file: " + name)
        return
    except OSError as e:
        print("cannot read " + name + ": " + str(e))
        return
    print(str(lines) + " lines, " + str(words) + " words, " + str(chars)
          + " characters (line breaks not counted) in " + name)


if __name__ == "__main__":
    main()
```

```
3 lines, 17 words, 99 characters (line breaks not counted) in notes.txt
no such file: nope.txt
usage: python word_count.py <file>
```

Bacaan lanjutan

The Java Tutorials [6], [7], [8] membahas larik, eksepsi, dan I/O dasar langkah demi langkah, dan dokumentasi API `ArrayList` [9] dan `Scanner` [10] layak dibaca utuh sekali. Aturan yang persis tentang eksepsi terperiksa terdapat dalam Bab 11 *Java Language Specification* [3]. Horstmann [11] memberikan pembahasan buku ajar yang menyeluruh atas semua Java di bab ini. Untuk Python, bab-bab tutorial resmi tentang struktur data [12] dan tentang *error* dan eksepsi [13], daftar eksepsi bawaan [1], serta dokumentasi modul `csv` [5] adalah sumber-sumber utamanya. Bloch [2] menghimpun kearifan yang telah diterima luas tentang pemakaian eksepsi yang baik, dan RFC 4180 [4] menguraikan format CSV.

Daftar Pustaka

[1] Python Software Foundation, “Built-in Exceptions,” *The Python Standard Library*. [Online]. Available: https://docs.python.org/3/library/exceptions.html

[2] J. Bloch, *Effective Java*, 3rd ed. Boston, MA, USA: Addison-Wesley, 2018, ch. 10, “Exceptions.”

[3] J. Gosling, B. Joy, G. Steele, G. Bracha, A. Buckley, D. Smith, and G. Bierman, *The Java Language Specification, Java SE 21 Edition*, ch. 11, “Exceptions.” Oracle, 2023.

[4] Y. Shafranovich, “Common Format and MIME Type for Comma-Separated Values (CSV) Files,” IETF RFC 4180, Oct. 2005.

[5] Python Software Foundation, “csv — CSV File Reading and Writing,” *The Python Standard Library*. [Online]. Available: https://docs.python.org/3/library/csv.html

[6] Oracle, "Arrays," *The Java Tutorials*. [Online]. Available: https://docs.oracle.com/javase/tutorial/java/nutsandbolts/arrays.html

[7] Oracle, "Lesson: Exceptions," *The Java Tutorials*. [Online]. Available: https://docs.oracle.com/javase/tutorial/essential/exceptions/

[8] Oracle, "Lesson: Basic I/O" and "The try-with-resources Statement," *The Java Tutorials*. [Online]. Available: https://docs.oracle.com/javase/tutorial/essential/io/

[9] Oracle, "Class ArrayList," *Java SE 21 & JDK 21 API Specification*, 2023. [Online]. Available: https://docs.oracle.com/en/java/javase/21/docs/api/java.base/java/util/ArrayList.html

[10] Oracle, "Class Scanner," *Java SE 21 & JDK 21 API Specification*, 2023. [Online]. Available: https://docs.oracle.com/en/java/javase/21/docs/api/java.base/java/util/Scanner.html

[11] C. S. Horstmann, *Core Java, Volume I: Fundamentals*, 12th ed. Hoboken, NJ, USA: Pearson, 2022.

[12] Python Software Foundation, "Data Structures," *The Python Tutorial*, ch. 5. [Online]. Available: https://docs.python.org/3/tutorial/datastructures.html

[13] Python Software Foundation, "Errors and Exceptions," *The Python Tutorial*, ch. 8. [Online]. Available: https://docs.python.org/3/tutorial/errors.html

Bab 6

Dokumentasi, Pengujian, dan Antarmuka

Setelah menyelesaikan bab ini, Anda akan mampu mendokumentasikan kelas dengan Javadoc dan *docstring* Python, lalu mengubah komentar itu menjadi halaman rujukan dengan `javadoc` dan `pydoc`; menguji sebuah kelas dari nol, mula-mula dengan program biasa yang memeriksa dirinya sendiri, kemudian dengan JUnit 6, `unittest`, dan pytest, memakai pola *arrange–act–assert*, kasus tepi, dan uji yang ditulis sebelum kodenya; menyelaraskan `equals`/`hashCode` serta `__eq__`/`__hash__` agar himpunan dan peta bekerja benar; memberi objek bentuk cetak yang mudah dibaca; melakukan *overloading* metode di Java dan menjelaskan mengapa Python mengerjakan hal yang sama dengan cara lain; dan, yang terpenting, mendefinisikan sebuah *interface* (antarmuka) — janji tentang apa yang dapat dilakukan suatu objek, tanpa sepatah kata pun tentang caranya — serta menulis kode yang hanya bergantung pada janji itu. Bab ini ditutup dengan mesin penghitung harga untuk Profitina Laundry: lima aturan harga berdiri di balik satu antarmuka, dan setiap aturan diuji tuntas dalam kedua bahasa.

6.1 Titik Berangkat Kita

Bab 5 mengajari program kita menyimpan banyak data sekaligus, membaca masukan, bertahan menghadapi data buruk dengan eksepsi, dan menyimpan hasil ke berkas. Program kita sudah cukup nyata untuk menjalankan sebuah toko kecil selama satu pagi — tetapi belum dapat dipercaya. Ajukan tiga pertanyaan yang kurang enak tentang kelas mana pun yang telah kita tulis.

Bagaimana seorang rekan tahu cara memakainya? Saat ini ia harus membaca kode sumbernya; itu mungkin untuk dua puluh baris, tetapi mustahil untuk ribuan kelas di pustaka Java. *Bagaimana kita tahu kelas itu benar?* Kita menjalankannya sekali lalu mengangguk; setelah perubahan berikutnya, tak seorang pun akan memeriksanya lagi. *Bagaimana mengubahnya tanpa merusak semua yang memakainya?* Jika lima puluh tempat memanggil metode penghitung harga secara langsung, mengubah cara menghitung harga berarti mendatangi lima puluh tempat.

Bab ini menjawab ketiga pertanyaan itu berurutan. **Dokumentasi** — komentar dengan format baku yang oleh perkakas diubah menjadi halaman rujukan — menjawab pertanyaan pertama. **Pengujian** — program kecil yang membandingkan apa yang dilakukan kode Anda dengan apa yang seharusnya dilakukan, setiap saat, dalam sedetik — menjawab yang kedua. **Antarmuka** — janji tertulis tentang *apa* yang dapat dilakukan sekelompok objek — menjawab yang ketiga, karena bagian lain program kemudian dapat bergantung pada janji itu dan tidak pernah pada rincian di baliknya.

Di sepanjang jalan kita bertemu tiga perkakas kecil yang menopang ketiganya: beberapa konstruktor untuk satu kelas, metode yang memutuskan apakah dua objek sama, dan beberapa versi dari satu nama metode. Masing-masing menyembunyikan jebakan, dan kita akan sengaja masuk ke setiap jebakan itu, dengan keluaran nyata, supaya Anda tidak pernah terperosok ke dalamnya tanpa sengaja.

> ✓ **WAWASAN KUNCI.** Dokumentasi, uji, dan antarmuka adalah tiga cara menuliskan sebuah *janji*. Komentar dokumentasi menyatakan janji itu dengan kata-kata untuk manusia. Uji menyatakannya dalam kode dan memeriksanya. Antarmuka menyatakannya dalam sistem tipe, sehingga kompilator (Java) atau pemeriksa tipe (Python) yang memeriksanya. Perangkat lunak yang baik menjaga ketiganya tetap selaras.

6.2 Menengok Kembali Sebuah Kelas

Sebelum mendokumentasikan atau menguji sebuah kelas, mari kita sepakati dulu bagian-bagiannya. Kelas yang lazim disusun dari atas ke bawah seperti berikut.

Paling atas terdapat **field** (medan): variabel milik setiap objek, yang dapat dilihat dan diubah oleh setiap metode kelas itu. Di Java kita menandainya `private`, dan di Python kita mengawali namanya dengan garis bawah, agar hanya kelas itu sendiri yang menyentuhnya.

Berikutnya **konstruktor**, yang berjalan ketika program menulis `new BankAccount(...)` di Java atau `BankAccount(...)` di Python, dan bertugas memberi setiap *field* nilai awal yang masuk akal. Kelas Java yang tidak mendeklarasikan konstruktor apa pun akan diberi konstruktor kosong tanpa parameter oleh kompilator.

Lalu **metode**: biasanya *getter* lebih dahulu, kemudian *setter*, dan di bagian bawah metode yang diwarisi setiap kelas dan biasanya diperbaiki — `toString`, yang menentukan rupa objek ketika dicetak, dan `equals`, yang menentukan kapan dua objek dianggap sama (Subbab 6.9 dan 6.10).

Urutan ini hanyalah kebiasaan, bukan aturan; kedua bahasa menerima anggota kelas dalam urutan apa pun.

6.2.1 Lebih dari satu cara lahir: beberapa konstruktor

Sebuah bank mengizinkan nasabah membuka rekening dengan dua cara. Ia dapat membukanya dengan setoran uang, dan bank memberinya batas cerukan (*overdraft*) 50 — saldonya boleh turun sampai 50 di bawah nol. Atau ia dapat membukanya kosong, dan cerukannya hanya 10. Rekeningnya tetap benda yang sama; yang berbeda hanyalah *nilai awalnya*, dan perbedaan itu bergantung pada apa yang dibawa nasabah.

Java mengungkapkan hal ini dengan **dua konstruktor** di dalam kelas yang sama. Keduanya bernama sama dengan kelasnya, sehingga harus berbeda daftar parameternya: yang satu menerima sebuah `double`, yang lain tidak menerima apa pun. Ketika program menulis `new BankAccount()`, kompilator memilih konstruktor tanpa parameter; ketika program menulis `new BankAccount(25)`, kompilator memilih yang menerima `double`. **Kode 6.1** memperlihatkan kelas selengkapnya. Abaikan sejenak komentar-komentar panjangnya — itulah pokok bahasan Subbab 6.3 — dan temukan kedua konstruktornya.

Kode 6.1 `BankAccount.java`: kelas dengan dua konstruktor dan komentar dokumentasi pada setiap anggotanya.

```java
/**
 * A bank account with a balance and an overdraft limit.
 *
 * <p>An account opened with money gets an overdraft of 50;
 * an account opened empty gets an overdraft of only 10.</p>
 */
public class BankAccount {
    private double balance;
    private int overdraft;

    /**
     * Opens an account with an opening deposit.
     *
     * @param openingAmount the money paid in on the first day
     */
    public BankAccount(double openingAmount) {
        balance = openingAmount;
        overdraft = 50;
    }

    /**
     * Opens an empty account. No amount is supplied,
     * so the smaller default overdraft applies.
     */
    public BankAccount() {
        balance = 0;
        overdraft = 10;
    }

    /**
     * Returns the current balance.
     *
     * @return the balance
     */
    public double getBalance() {
        return balance;
    }
```

Kode 6.1 (Lanjutan) *BankAccount.java: kelas dengan dua konstruktor dan komentar dokumentasi pada setiap anggotanya.*

```
    /**
     * Returns how far the balance may go below zero.
     *
     * @return the overdraft limit
     */
    public int getOverdraft() {
        return overdraft;
    }

    /**
     * Calculates the compound interest earned on an amount.
     *
     * @param initial the initial balance
     * @param years   the number of years
     * @param rate    the yearly interest rate, for example 0.06 for 6%
     * @return the compound interest on the balance
     * @throws IllegalArgumentException if {@code years} is negative
     */
    public double interestOn(double initial, int years, double rate) {
        if (years < 0) {
            throw new IllegalArgumentException("years must not be negative");
        }
        double total = initial * Math.pow(1 + rate, years);
        return total - initial;
    }

    /**
     * Describes the account for printing.
     *
     * @return the balance and the overdraft in one line
     */
    @Override
    public String toString() {
        return "BankAccount[balance=" + balance + ", overdraft=" + overdraft + "]";
    }
```

Kode 6.1 (Lanjutan) *BankAccount.java: kelas dengan dua konstruktor dan komentar dokumentasi pada setiap anggotanya.*

```
    /**
     * Opens two accounts and prints them.
     *
     * @param args not used
     */
    public static void main(String[] args) {
        BankAccount ba1 = new BankAccount();      // konstruktor tanpa argumen
        BankAccount ba2 = new BankAccount(25);    // konstruktor satu argumen
        System.out.println(ba1);
        System.out.println(ba2);
        double interest = ba2.interestOn(1_000_000, 2, 0.06);
        System.out.println("Interest on 1,000,000 at 6% for 2 years: " +
Math.round(interest));
    }
}
```

Setelah dikompilasi dengan `javac BankAccount.java` dan dijalankan dengan `java BankAccount`, program mencetak:

```
BankAccount[balance=0.0, overdraft=10]
BankAccount[balance=25.0, overdraft=50]
Interest on 1,000,000 at 6% for 2 years: 123600
```

Rekening pertama, `ba1`, dibangun oleh konstruktor tanpa parameter, sehingga saldonya `0.0` dan cerukannya `10`. Rekening kedua, `ba2`, dibangun oleh konstruktor berparameter satu: saldo `25.0`, cerukan `50`. Baris ketiga memanggil `interestOn`, yang kita jelaskan di Subbab 6.3.

Python hanya mengizinkan **satu** `__init__` per kelas. Jika Anda menulis yang kedua, yang kedua itu begitu saja menggantikan yang pertama (Subbab 6.11 memperlihatkan perilaku ini pada metode biasa). Python mencapai tujuan yang sama dengan **argumen bawaan** (*default argument*): parameter yang nilainya sudah ditulis di kepala fungsi dan dipakai setiap kali pemanggil tidak menyebutkannya. **Kode 6.2** memberi `opening_amount` nilai bawaan `None`, nilai Python yang berarti "tidak ada apa-apa di sini", lalu menanyakan kasus mana yang sedang terjadi.

Kode 6.2 `bank_account.py`: satu `__init__` dengan argumen bawaan mengerjakan tugas dua konstruktor Java.

```
"""A bank account with a balance and an overdraft limit.

Run this file to open two accounts, or run
``python -m doctest -v bank_account.py`` to check the examples
written in the docstrings.
"""


class BankAccount:
    """A bank account with a balance and an overdraft limit.

    An account opened with money gets an overdraft of 50;
    an account opened empty gets an overdraft of only 10.

    >>> BankAccount().overdraft
    10
    >>> BankAccount(25).overdraft
    50
    """

    def __init__(self, opening_amount=None):
        """Open an account, with or without an opening deposit.

        Args:
            opening_amount: the money paid in on the first day, or
                None (the default) to open an empty account.
        """
        if opening_amount is None:          # tidak ada jumlah yang diberikan: rekening
kosong
            self._balance = 0.0
            self._overdraft = 10
        else:
            self._balance = float(opening_amount)
            self._overdraft = 50

    @property
    def balance(self):
        """The current balance."""
        return self._balance

    @property
    def overdraft(self):
        """How far the balance may go below zero."""
        return self._overdraft
```

Kode 6.2 (Lanjutan) *bank_account.py: satu __init__ dengan argumen bawaan mengerjakan tugas dua konstruktor Java.*

```
    def interest_on(self, initial, years, rate):
        """Calculate the compound interest earned on an amount.

        Args:
            initial: the initial balance.
            years: the number of years.
            rate: the yearly interest rate, for example 0.06 for 6%.

        Returns:
            The compound interest on the balance.

        Raises:
            ValueError: if years is negative.

        >>> round(BankAccount().interest_on(1_000_000, 2, 0.06))
        123600
        """
        if years < 0:
            raise ValueError("years must not be negative")
        total = initial * (1 + rate) ** years
        return total - initial

    def __repr__(self):
        return f"BankAccount[balance={self._balance}, overdraft={self._overdraft}]"


if __name__ == "__main__":
    ba1 = BankAccount()        # tanpa argumen: nilai bawaan None yang dipakai
    ba2 = BankAccount(25)
    print(ba1)
    print(ba2)
    interest = ba2.interest_on(1_000_000, 2, 0.06)
    print("Interest on 1,000,000 at 6% for 2 years:", round(interest))
```

Dijalankan dengan `python3 bank_account.py`, program mencetak tiga baris yang persis sama dengan program Java:

```
BankAccount[balance=0.0, overdraft=10]
BankAccount[balance=25.0, overdraft=50]
Interest on 1,000,000 at 6% for 2 years: 123600
```

Mengapa `None` dan bukan `0`? Karena bagi bank ini "dibuka dengan uang nol" dan "dibuka tanpa apa-apa" adalah dua keadaan yang berbeda: `BankAccount(0)` adalah setoran yang kebetulan bernilai nol dan berhak atas cerukan yang lebih besar, persis seperti `new BankAccount(0)` di Java. Baris-baris `@property` menjadikan `balance` dan `overdraft` atribut yang hanya dapat dibaca, sebagaimana *field* privat dengan *getter* publik di Java.

> ⚠ **RENUNGKAN.** Di Java, kedua konstruktor mengulang pekerjaan yang sama: mengisi dua *field*. Dengan dua *field*, hal itu tidak berbahaya; dengan sepuluh, suatu hari salah satu konstruktor akan mengisi sembilan dan melupakan yang kesepuluh. Java mengizinkan sebuah konstruktor memanggil konstruktor lain dengan `this(...)` sebagai pernyataan pertamanya, sehingga semua pekerjaan sesungguhnya terkumpul di satu tempat. Soal Latihan 6.3 meminta Anda menulis ulang `BankAccount` dengan cara itu, dan jawaban Python — konstruktor alternatif yang ditulis sebagai metode kelas — muncul dalam penyelesaiannya.

6.3 Komentar Dokumentasi: Javadoc dan Docstring

Anda sudah biasa menulis komentar. Komentar seperti `// the no-argument constructor` pada **Kode 6.1** berbicara kepada orang yang *membaca kode*: ia menjelaskan sebuah baris.

Ada pembaca kedua yang sama sekali tidak membaca kode Anda. Rekan yang ingin *memakai* `BankAccount` perlu tahu apa yang dikerjakan setiap metode publik, apa yang diharapkannya, apa yang dikembalikannya, dan apa yang bisa salah — dan tidak perlu tahu apa pun tentang cara kerjanya di dalam. Itulah persisnya yang disampaikan halaman API Java tentang `String` atau `ArrayList`: Anda telah memakai kelas-kelas itu selama lima bab tanpa pernah melihat kode sumbernya.

Komentar untuk pembaca kedua ini disebut **komentar dokumentasi**, dan kedua bahasa memiliki format baku untuknya sehingga sebuah perkakas dapat menghimpunnya menjadi halaman rujukan.

> **DEFINISI.** *Komentar dokumentasi* menggambarkan *antarmuka* sebuah kelas atau metode — tujuannya, parameternya, hasilnya, dan kegagalannya — dalam format yang dapat diekstrak oleh perkakas dokumentasi. Di Java bentuknya adalah *komentar Javadoc*, yang diawali `/**`, diakhiri `*/`, dan diletakkan tepat sebelum kelas, *field*, konstruktor, atau metode yang digambarkannya. Di Python bentuknya adalah *docstring*: literal string yang ditulis sebagai pernyataan paling awal di dalam modul, kelas, atau fungsi yang digambarkannya.

Satu catatan tentang bahasa. Dalam kode program buku ini, komentar biasa (`//` dan `#`) diterjemahkan ke bahasa Indonesia, tetapi komentar Javadoc dan *docstring* sengaja dibiarkan dalam bahasa Inggris. Keduanya adalah dokumentasi API: teks itulah yang muncul dalam halaman buatan `javadoc`, dalam keluaran `pydoc`, dan dalam laporan uji yang dikutip di bab ini, dan bahasa Inggris adalah kelaziman internasional untuk dokumentasi API.

6.3.1 Javadoc, langkah demi langkah

Ambillah metode yang paling sederhana, sebuah *getter*:

```
public class Pet {
    private final String name = "Rover";

    /**
     * Gets the pet's name.
     *
     * @return the pet's name
     */
    public String getName() {
        return name;
    }
}
```

Komentar itu dibuka dengan garis miring dan **dua** tanda bintang, `/**`, dan ditutup dengan `*/`. Tanda bintang di awal baris-baris di antaranya hanyalah hiasan; perkakas akan membuangnya. Kalimat pertama, "Gets the pet's name.", adalah *ringkasan*: kalimat ini muncul di tabel metode di bagian atas halaman yang dihasilkan, jadi ia harus mengatakan dalam beberapa kata apa yang dikerjakan metode itu. Setelah satu baris kosong datang **tag blok**, kata berawalan `@` yang menandai satu jenis keterangan tertentu. `@return` menggambarkan nilai yang dikembalikan metode.

Metode yang berparameter menggambarkan setiap parameternya, berurutan, masing-masing dengan satu tag `@param`, dalam bentuk `@param <namaParameter> <keterangan>`. Namanya harus

dieja persis seperti di dalam kode — perkakasnya memeriksa hal itu. Metode `interestOn` pada **Kode 6.1** memiliki tiga parameter, satu hasil, dan satu kemungkinan kegagalan, sehingga komentarnya memuat tiga tag `@param`, satu `@return`, dan satu `@throws`:

```
/**
 * Calculates the compound interest earned on an amount.
 *
 * @param initial the initial balance
 * @param years   the number of years
 * @param rate    the yearly interest rate, for example 0.06 for 6%
 * @return the compound interest on the balance
 * @throws IllegalArgumentException if {@code years} is negative
 */
```

Metode ini menghitung bunga *majemuk*: setiap tahun bunga ditambahkan ke saldo, dan bunga tahun berikutnya dihitung dari jumlah yang lebih besar itu. Setelah satu tahun, jumlah awal P dengan suku bunga tahunan r tumbuh menjadi $P(1+r)$; setelah dua tahun menjadi $P(1+r)(1+r) = P(1+r)^2$; setelah n tahun menjadi $P(1+r)^n$. Bunganya adalah apa yang ditambahkan pada jumlah awal:

$$I = P(1+r)^n - P \tag{6.1}$$

Persamaan (6.1) persis sama dengan dua baris di tubuh `interestOn`: `total = initial * Math.pow(1 + rate, years)` menghitung $P(1+r)^n$, dan `total - initial` mengurangkan P. Dengan $P = 1\,000\,000$, $r = 0.06$, dan $n = 2$ kita memperoleh $1\,000\,000 \times 1.06^2 = 1\,000\,000 \times 1.1236 = 1\,123\,600$, sehingga $I = 123\,600$ — angka yang dicetak kedua program.

Tag sebaris `{@code years}` di dalam baris `@throws` meminta perkakas mencetak `years` dengan fon kode. Tag sebaris hidup *di dalam* kalimat dan diapit kurung kurawal; tag blok mengawali sebuah baris. **Tabel 6.1** mendaftar tag yang paling sering Anda pakai, beserta bagian *docstring* Python yang setara.

Tabel 6.1 Tag Javadoc yang paling sering dipakai, dan tempat keterangan yang sama dalam *docstring* Python.

Tag Javadoc	Yang didokumentasikan	*Docstring* Python (gaya Google)
kalimat pertama	ringkasan satu baris	baris pertama
`@param name text`	satu parameter, berurutan	satu entri di bawah `Args:`
`@return text`	nilai yang dikembalikan (tiadakan untuk `void`)	`Returns:`
`@throws Type text`	sebuah eksepsi dan kapan terjadinya	`Raises:`
`{@code text}`	fon kode di dalam kalimat	tanda petik terbalik, `` `text` ``
`{@link Class#method}`	tautan ke butir lain yang dapat diklik	namanya, misalnya `Dog.age`
`@see item`	"lihat juga" di akhir halaman	bagian `See Also:`
`@since version`	versi yang menambahkan butir itu	catatan di dalam teks
`@deprecated text`	"jangan dipakai lagi; pakailah …"	catatan `.. deprecated::`

Semua yang dibutuhkan pemanggil harus ada di dalam komentar; tidak ada satu pun tentang isi metode. "Memakai `Math.pow`" adalah rincian implementasi. "Suku bunga berupa pecahan, jadi 6% ditulis 0.06" adalah bagian dari kontrak, karena pemanggil yang memberikan `6` akan mendapati bank yang luar biasa dermawan.

> ⚠ **JEBAKAN.** Komentar Javadoc harus diakhiri dengan `*/`. Andaikan, karena mengetik terburu-buru, Anda mengakhiri komentar `interestOn` dengan `/` saja. Kompilator tidak mengeluh — ia menganggap segala sesuatu sampai `*/` *berikutnya* sebagai bagian komentar. Jika metode berikutnya memiliki komentar Javadoc, seluruh `interestOn`, kepala dan tubuhnya, lenyap ke dalam komentar. Kami mencobanya dengan kelas yang berisi `interestOn` diikuti metode terdokumentasi `twice`. `javac` mengompilasinya tanpa sepatah kata pun; ketika kelas hasil kompilasi itu ditanya isinya (`javap Interest`), jawabannya

```
Compiled from "Interest.java"
public class Interest {
  public Interest();
  public double twice(double);
}
```

> dan program yang memanggil metode itu kemudian ditolak dengan

```
UseInterest.java:3: error: cannot find symbol
        System.out.println(new Interest().interestOn(1000, 2, 0.06));
                                         ^
  symbol:   method interestOn(int,int,double)
  location: class Interest
1 error
```

> Metode itu hilang karena Anda sendiri menjadikannya komentar. Penyunting dengan pewarnaan sintaksis menampilkan seluruh metode dalam warna komentar — satu alasan lagi untuk memakainya.

6.3.2 Docstring, langkah demi langkah

Docstring Python mengikuti konvensi PEP 257 [1]. *Docstring* ditulis dengan tiga tanda petik ganda, `"""`, agar dapat membentang beberapa baris. Baris pertamanya adalah ringkasan satu kalimat yang diakhiri titik dan ditulis sebagai perintah (“Open an account”, “Return the balance”). Jika masih ada yang perlu dikatakan, menyusul satu baris kosong, lalu rinciannya.

PEP 257 hanya menetapkan kerangkanya; isi *docstring* yang panjang mengikuti salah satu dari beberapa gaya populer. Buku ini memakai *gaya Google*, yang dapat Anda lihat di **Kode 6.2**: bagian-bagian berjudul `Args:`, `Returns:`, dan `Raises:`, masing-masing diikuti entri yang menjorok. Gaya ini enak dibaca sebagai teks biasa, dan dipahami oleh setiap perkakas dokumentasi Python.

Python menyimpan setiap *docstring* saat program berjalan. Ketik `help(BankAccount.interest_on)` di *prompt* Python dan *docstring*-nya tercetak; atribut `BankAccount.interest_on.__doc__` memuat teks yang sama sebagai string biasa. Komentar Java, sebaliknya, dibuang oleh kompilator; hanya perkakas `javadoc` yang membacanya.

6.3.3 Contoh yang memeriksa dirinya sendiri: doctest

Perhatikan lagi *docstring* kelas pada **Kode 6.2**. Di dalamnya ada baris-baris yang tampak seperti sesi Python:

```
>>> BankAccount().overdraft
10
```

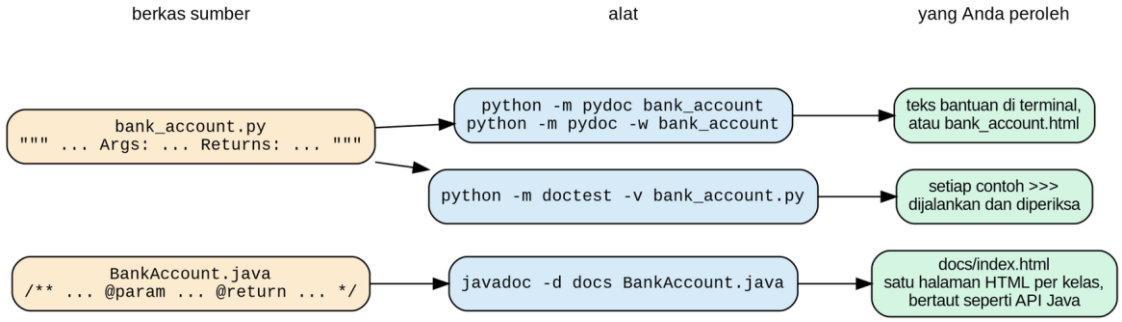
Modul standar `doctest` [2] menemukan setiap pasangan seperti itu — baris yang diawali `>>>` dan keluaran yang seharusnya mengikutinya — menjalankan kodenya, lalu membandingkan. Perintah `python3 -m doctest -v bank_account.py` menjalankan ketiga contoh di berkas itu; baris-baris terakhirnya berbunyi

```
   2 tests in bank_account.BankAccount
   1 tests in bank_account.BankAccount.interest_on
3 tests in 7 items.
3 passed and 0 failed.
Test passed.
```

Doctest adalah contoh dokumentasi yang tidak mungkin usang tanpa ada yang menyadarinya. Ia juga sekilas pandang pertama kita pada pengujian: kode yang menjalankan kode lain dan membandingkan hasilnya dengan harapan. Java tidak memiliki padanan langsung; sejak Java 18, tag `{@snippet}` memungkinkan contoh kode yang diperiksa disertakan dalam Javadoc, tetapi rumah yang tepat untuk pemeriksaan di Java adalah kelas uji, yang dibangun di Subbab 6.7.

6.4 Membangkitkan dan Membaca Dokumentasi

Komentar dokumentasi baru berbuah ketika sebuah perkakas mengubahnya menjadi halaman. **Gambar 6.1** memperlihatkan tiga jalur yang dipakai bab ini: `javadoc` untuk Java, `pydoc` untuk Python, dan `doctest`, yang membaca *docstring* yang sama tetapi *menjalankan* contoh-contohnya alih-alih mencetaknya.



Gambar 6.1 Dari komentar dokumentasi menjadi dokumentasi: `javadoc` membangun halaman HTML yang saling bertaut, `pydoc` mencetak teks bantuan atau menulis halaman HTML, dan `doctest` menjalankan contoh yang ditulis di dalam *docstring*.

6.4.1 javadoc

`javadoc` terpasang bersama setiap JDK, bersebelahan dengan `javac` dan `java`, dan manual rujukannya [3] mendaftar setiap opsi dan tag. Di terminal yang dibuka pada folder tempat `BankAccount.java` berada, ketikkan

```
javadoc -d docs BankAccount.java
```

Opsi `-d docs` menyebutkan folder yang menerima halaman-halamannya. (Untuk mendokumentasikan semua kelas di folder itu, tulis `javadoc -d docs *.java`.) Perkakas itu melaporkan apa yang dikerjakannya:

```
Loading source file BankAccount.java...
Constructing Javadoc information...
Creating destination directory: "docs/"
Building index for all the packages and classes...
Standard Doclet version 21.0.10+7-Ubuntu-124.04
Building tree for all the packages and classes...
Generating docs/BankAccount.html...
Generating docs/package-summary.html...
Generating docs/package-tree.html...
Generating docs/overview-tree.html...
Building index for all classes...
Generating docs/allclasses-index.html...
Generating docs/allpackages-index.html...
Generating docs/index-all.html...
Generating docs/search.html...
Generating docs/index.html...
Generating docs/help-doc.html...
```

Buka `docs/index.html` di peramban dan Anda akan menemukan halaman yang persis seperti API Java resmi: keterangan kelas di atas, *Constructor Summary* dengan kedua konstruktor, tabel *Method Summary* yang kolom kanannya memuat kalimat ringkasan setiap metode, dan di bawahnya satu entri rinci per metode dengan parameter, nilai kembalian, dan eksepsinya. (Baris versi akan menunjukkan JDK Anda sendiri.)

`javadoc` juga *memeriksa* komentar. Berikut metode yang komentarnya salah mengeja satu parameter dan melupakan satu lagi:

```java
/** Interest helpers. */
public class Interest2 {
    /** Creates the helper. */
    public Interest2() {
    }

    /**
     * Calculates interest after a certain number of years.
     *
     * @param initial the initial balance
     * @param year the number of years
     * @return the compound interest on the balance
     */
    public double interestOn(double initial, int years, double rate) {
        double total = initial * Math.pow(1 + rate, years);
        return total - initial;
    }
}
```

dan inilah kata `javadoc -quiet -d out Interest2.java` tentangnya:

```
Interest2.java:11: error: @param name not found
     * @param year the number of years
             ^
Interest2.java:14: warning: no @param for years
    public double interestOn(double initial, int years, double rate) {
                  ^
Interest2.java:14: warning: no @param for rate
    public double interestOn(double initial, int years, double rate) {
                  ^
1 error
2 warnings
```

Itulah *uji* dokumentasi, gratis. Jadikan kebiasaan: dokumentasikan setiap kelas, konstruktor, dan metode publik, jalankan `javadoc`, dan perbaiki setiap *error* dan *warning* sampai perkakas itu

berjalan tanpa suara. `BankAccount` ditulis dengan cara itu, itulah sebabnya jalannya di atas tidak menampilkan satu peringatan pun.

6.4.2 pydoc

Modul `pydoc` Python membaca *docstring*. `python3 -m pydoc bank_account` mencetak dokumentasi modul itu di terminal (itulah yang ditampilkan `help()`), dan `python3 -m pydoc -w bank_account` menuliskannya ke `bank_account.html`. Versi terminal untuk **Kode 6.2** dimulai seperti ini (teks lengkapnya sekitar enam puluh baris):

```
Help on module bank_account:

NAME
    bank_account - A bank account with a balance and an overdraft limit.

DESCRIPTION
    Run this file to open two accounts, or run
    ``python -m doctest -v bank_account.py`` to check the examples
    written in the docstrings.

CLASSES
    builtins.object
        BankAccount

    class BankAccount(builtins.object)
     |  BankAccount(opening_amount=None)
     |
     |  A bank account with a balance and an overdraft limit.
```

Lebih ke bawah, `pydoc` mendaftar `interest_on` dengan bagian `Args:`, `Returns:`, dan `Raises:` persis seperti yang ditulis, dan mendaftar `balance` serta `overdraft` di bawah "Readonly properties" — ia mengenali dekorator `@property` dengan sendirinya. Proyek Python yang lebih besar memakai Sphinx, yang membangun seluruh situs web dari *docstring* yang sama; *docstring* yang Anda tulis hari ini tidak perlu diubah untuk bekerja di sana.

6.4.3 Membaca API: kelas `Random`

Membangkitkan dokumentasi baru separuh keterampilan; separuh lainnya adalah *membacanya*. Dokumentasi API Java resmi persis seperti yang dihasilkan `javadoc` untuk kita, hanya saja untuk ribuan kelas.

Andaikan program laundry harus memutuskan secara acak mesin cuci mana di antara empat mesin yang menerima muatan berikutnya (cara sederhana untuk meratakan keausan). Java menawarkan `Math.random()`, yang mengembalikan `double` antara 0 (inklusif) dan 1 (eksklusif), dan kelas `java.util.Random`, yang halaman API-nya mendaftar metode untuk berbagai jenis keacakan: `nextDouble()`, `nextInt()`, `nextInt(bound)`, `nextBoolean()`, dan lain-lain. Tabel ringkasannya memberi tahu bahwa `nextInt(10)` mengembalikan nilai dari 0 (inklusif) sampai 10 (eksklusif) — jadi nomor mesin 1 sampai 4 adalah `nextInt(4) + 1`. **Kode 6.3** memakai metode-metode ini.

Kode 6.3 `RandomDemo.java`: metode-metode utama `java.util.Random`, dengan benih tetap.

```java
import java.util.Random;

/**
 * Shows the main methods of {@code java.util.Random}.
 */
public class RandomDemo {
    /**
     * Draws a few random values and assigns eight loads to four machines.
     *
     * @param args not used
     */
    public static void main(String[] args) {
        Random rand = new Random(2026);   // benih tetap: bilangan yang sama setiap kali dijalankan
        double randomNumber = rand.nextDouble();      // 0.0 (inklusif) sampai 1.0 (eksklusif)
        int randomInt = rand.nextInt();               // int apa saja
        int upperBoundedInt = rand.nextInt(10);       // 0 (inklusif) sampai 10 (eksklusif)
        boolean coin = rand.nextBoolean();
        System.out.println("nextDouble() = " + randomNumber);
        System.out.println("nextInt()    = " + randomInt);
        System.out.println("nextInt(10)  = " + upperBoundedInt);
        System.out.println("nextBoolean()= " + coin);

        System.out.print("Machines for 8 loads:");
        for (int load = 1; load <= 8; load++) {
            int machine = rand.nextInt(4) + 1;        // 1, 2, 3, atau 4
            System.out.print(" " + machine);
        }
        System.out.println();
    }
}
```

```
nextDouble() = 0.6186327814734252
nextInt()    = -647640901
nextInt(10)  = 0
nextBoolean()= true
Machines for 8 loads: 1 3 3 4 1 1 4 4
```

Dokumentasi Python untuk modul `random` menawarkan `random()`, `randrange(stop)`, `randint(a, b)` (yang, berbeda dengan `randrange`, *menyertakan* batas atasnya *b*), `choice(sequence)`, dan `getrandbits(k)`. **Kode 6.4** adalah program yang sama.

Kode 6.4 `random_demo.py`: gagasan yang sama dengan modul `random` Python.

```python
"""Shows the main functions of Python's random module."""
import random

rand = random.Random(2026)          # benih tetap: bilangan yang sama setiap kali dijalankan
random_number = rand.random()       # 0.0 (inklusif) sampai 1.0 (eksklusif)
any_int = rand.getrandbits(32)      # 32 bit acak, seperti nextInt() di Java
upper_bounded_int = rand.randrange(10)   # 0 (inklusif) sampai 10 (eksklusif)
coin = rand.choice([True, False])
print("random()        =", random_number)
print("getrandbits(32) =", any_int)
print("randrange(10)   =", upper_bounded_int)
print("choice(...)     =", coin)

machines = [rand.randint(1, 4) for load in range(8)]   # randint mencakup kedua ujung
print("Machines for 8 loads:", *machines)
```

```
random()        = 0.11911988496396309
getrandbits(32) = 2158288730
randrange(10)   = 8
choice(...)     = True
Machines for 8 loads: 2 4 4 4 2 1 1 1
```

Bilangan-bilangannya berbeda antara kedua bahasa — masing-masing memakai pembangkitnya sendiri — tetapi setiap program mencetak bilangan yang *sama* setiap kali dijalankan, di komputer mana pun. Itulah kerja **benih** (*seed*), angka `2026` yang diberikan ketika pembangkit dibuat. Pembangkit yang dimulai dari benih yang sama menghasilkan barisan yang sama. Tanpa benih (`new Random()`, `random.Random()`), barisannya berubah setiap kali dijalankan. Untuk laundry sungguhan Anda menginginkan yang kedua; untuk sebuah *uji* Anda menginginkan yang pertama, karena uji yang masukannya berubah setiap kali dijalankan tidak dapat memberi tahu apakah hasil baru itu *bug* atau kebetulan. Pegang gagasan ini: itulah aturan pertama subbab berikutnya.

6.5 Pengujian dari Nol

Sebuah program *benar* jika ia melakukan apa yang dijanjikan dokumentasinya untuk setiap masukan yang diterimanya. Kita tidak mungkin mencoba setiap masukan — `interestOn` menerima lebih banyak kombinasi `double` daripada jumlah atom di sebuah laundry — maka kita mengerjakan hal terbaik berikutnya: memilih segelintir masukan dengan cerdik, menghitung dengan tangan jawaban yang semestinya, lalu membiarkan komputer membandingkan. Itulah sebuah **uji**.

> **DEFINISI.** *Uji unit* (*unit test*) adalah potongan kode kecil yang menjalankan satu bagian kecil program (sebuah *unit*: biasanya satu metode dari satu kelas) pada masukan terpilih, membandingkan hasil sebenarnya dengan hasil *yang diharapkan* yang telah dihitung sebelumnya, dan melaporkan *lulus* jika keduanya cocok dan *gagal* jika tidak. *Rangkaian uji* (*test suite*) adalah kumpulan uji unit yang dapat dijalankan bersama dengan satu perintah.

Mengapa repot, padahal kita bisa menjalankan program dan melihat hasilnya? Ada tiga alasan, dari yang kurang penting ke yang paling penting.

1. **Melihat itu tidak andal.** Manusia yang membaca empat puluh baris keluaran akan terlewat bahwa baris ke-23 berbunyi `27000`, bukan `28000`; perbandingan di dalam kode tidak pernah terlewat.
2. **Melihat tidak diulang.** Rangkaian uji berjalan lagi dalam sedetik, setelah setiap perubahan, dan menyebutkan dengan tepat janji mana yang dilanggar.
3. **Uji mengubah cara Anda merancang.** Kelas yang sulit diuji — karena membaca papan ketik di tengah perhitungan atau memilih bilangan acak tanpa benih — hampir selalu kelas yang pembagiannya buruk. Uji mendorong Anda ke arah metode-metode kecil dengan masukan dan keluaran yang jelas, dan antarmuka, seperti diperlihatkan akhir bab ini, membuat pengujian mudah justru karena alasan itu.

6.5.1 Kelas yang diuji

Kita mulai dengan kelas yang cukup kecil untuk digenggam satu tangan. Seekor `Dog` memiliki nama dan umur; umur anjing bertambah satu pada hari ulang tahunnya; umur negatif tidak masuk akal dan wajib ditolak. **Kode 6.5** dan **Kode 6.6** adalah kedua versinya, didokumentasikan seperti diajarkan Subbab 6.3.

Kode 6.5 `Dog.java`: kelas yang akan kita uji. Kelas ini tidak memiliki `main`; program-program uji di bawah yang memakainya.

```java
/**
 * A dog with a name and an age in whole years.
 */
public class Dog {
    private final String name;
    private int age;

    /**
     * Creates a dog.
     *
     * @param name the dog's name
     * @param age  the dog's age in years
     * @throws IllegalArgumentException if {@code age} is negative
     */
    public Dog(String name, int age) {
        if (age < 0) {
            throw new IllegalArgumentException("age must not be negative: " + age);
        }
        this.name = name;
        this.age = age;
    }

    /**
     * Gets the dog's name.
     *
     * @return the dog's name
     */
    public String getName() {
        return name;
    }

    /**
     * Gets the dog's age.
     *
     * @return the age in years
     */
    public int getAge() {
        return age;
    }
```

Kode 6.5 (Lanjutan) *Dog.java: kelas yang akan kita uji. Kelas ini tidak memiliki main; program-program uji di bawah yang memakainya.*

```java
    /**
     * Makes the dog one year older (call it on the dog's birthday).
     */
    public void incrementAge() {
        age++;
    }
}
```

Kode 6.6 `dog.py`: kelas yang sama dalam Python, dengan *property* yang hanya dapat dibaca.

```
"""A dog with a name and an age in whole years."""


class Dog:
    """A dog with a name and an age in whole years."""

    def __init__(self, name, age):
        """Create a dog.

        Args:
            name: the dog's name.
            age: the dog's age in years.

        Raises:
            ValueError: if age is negative.
        """
        if age < 0:
            raise ValueError(f"age must not be negative: {age}")
        self._name = name
        self._age = age

    @property
    def name(self):
        """The dog's name."""
        return self._name

    @property
    def age(self):
        """The dog's age in years."""
        return self._age

    def increment_age(self):
        """Make the dog one year older (call it on the dog's birthday)."""
        self._age += 1
```

Kedua berkas tidak mencetak apa pun dengan sendirinya. Itu wajar untuk *kelas pustaka*: tugasnya dipakai oleh kode lain, dan kode lain pertama yang memakainya adalah ujinya.

6.5.2 Uji tanpa pustaka apa pun

Anda tidak memerlukan kerangka kerja pengujian apa pun untuk menguji. Yang dibutuhkan sebuah uji hanyalah cara untuk berkata "saya mengharapkan ini, saya mendapat itu, apakah keduanya cocok?" **Kode 6.7** menuliskannya sebagai metode lima baris, `check`, lalu memakainya empat kali.

Kode 6.7 `DogCheck.java`: menguji `Dog` dengan metode `check` buatan sendiri, tanpa pustaka uji.

```
/**
 * Tests {@link Dog} without any test library: every check prints PASS or FAIL.
 */
public class DogCheck {
    private static int passed = 0;
    private static int failed = 0;

    /**
     * Compares what we expected with what actually happened, and reports it.
     *
     * @param what     a short description of the check
     * @param expected the value a correct Dog must produce
     * @param actual   the value this Dog did produce
     */
    static void check(String what, Object expected, Object actual) {
        if (expected.equals(actual)) {
            passed++;
            System.out.println("PASS  " + what);
        } else {
            failed++;
            System.out.println("FAIL  " + what + ": expected " + expected + " but was " +
actual);
        }
    }

    /**
     * Runs every check, then prints a one-line verdict.
     *
     * @param args not used
     */
    public static void main(String[] args) {
        // Siapkan: bangun objek yang diuji
        Dog rover = new Dog("Rover", 5);
        // Lakukan dan pastikan, satu perilaku setiap kali
        check("age is 5", 5, rover.getAge());
        rover.incrementAge();
        check("age is now 6", 6, rover.getAge());
        check("name is Rover", "Rover", rover.getName());

        // Kasus tepi: umur negatif wajib ditolak
        try {
            new Dog("Rex", -1);
            check("negative age is rejected", "exception", "no exception");
        } catch (IllegalArgumentException e) {
            check("negative age is rejected", "exception", "exception");
        }
```

Kode 6.7 (Lanjutan) *DogCheck.java: menguji Dog dengan metode check buatan sendiri, tanpa pustaka uji.*

```
        System.out.println(passed + " passed, " + failed + " failed");
        if (failed > 0) {
            System.exit(1);   // kode keluar bukan nol memberi tahu skrip bahwa ada yang
rusak
        }
    }
}
```

Kompilasi kedua kelas lalu jalankan pemeriksanya (`javac Dog.java DogCheck.java`, kemudian `java DogCheck`):

```
PASS  age is 5
PASS  age is now 6
PASS  name is Rover
PASS  negative age is rejected
4 passed, 0 failed
```

Bacalah `main` dari atas ke bawah dan Anda akan melihat pola yang diikuti setiap uji yang baik: *bangun* apa yang kita perlukan (Rover, umur 5), *lakukan* sesuatu (tanyakan umurnya, tambahkan satu tahun), dan *periksa* hasilnya terhadap angka yang dihitung *sebelum* apa pun dijalankan. Pemeriksaan terakhir adalah **kasus tepi** (*edge case*) — nilai di perbatasan dari apa yang diterima kelas — dan ia menguji sebuah *kegagalan*: uji itu lulus hanya jika `new Dog("Rex", -1)` melempar eksepsi. Rangkaian uji yang hanya mencoba kasus-kasus menyenangkan membiarkan kode yang paling berbahaya tidak terjamah.

Baris-baris terakhir juga penting: setelah mencetak putusannya, program *keluar dengan status bukan nol* jika ada yang gagal, karena perkakas *build*, penyunting, dan server yang memeriksa setiap perubahan membaca status itu.

Kode 6.8 adalah pemeriksa yang sama dalam Python.

Kode 6.8 `dog_check.py`: program pemeriksa-diri yang sama dalam Python.

```python
"""Test Dog without any test library: every check prints PASS or FAIL."""
import sys

from dog import Dog

passed = 0
failed = 0


def check(what, expected, actual):
    """Compare what we expected with what actually happened, and report it."""
    global passed, failed
    if expected == actual:
        passed += 1
        print("PASS ", what)
    else:
        failed += 1
        print("FAIL ", what + f": expected {expected!r} but was {actual!r}")


# Siapkan: bangun objek yang diuji
rover = Dog("Rover", 5)
# Lakukan dan pastikan, satu perilaku setiap kali
check("age is 5", 5, rover.age)
rover.increment_age()
check("age is now 6", 6, rover.age)
check("name is Rover", "Rover", rover.name)

# Kasus tepi: umur negatif wajib ditolak
try:
    Dog("Rex", -1)
    check("negative age is rejected", "exception", "no exception")
except ValueError:
    check("negative age is rejected", "exception", "exception")

print(f"{passed} passed, {failed} failed")
if failed > 0:
    sys.exit(1)       # kode keluar bukan nol memberi tahu skrip bahwa ada yang rusak
```

```
PASS  age is 5
PASS  age is now 6
PASS  name is Rover
PASS  negative age is rejected
4 passed, 0 failed
```

Sekarang mari kita rusak anjingnya dengan sengaja. Ubah `age++` menjadi `age += 2` di `Dog.java` (atau `self._age += 1` menjadi `self._age += 2` di `dog.py`), simpan, lalu jalankan pemeriksanya lagi. Kedua bahasa melaporkan hal yang sama:

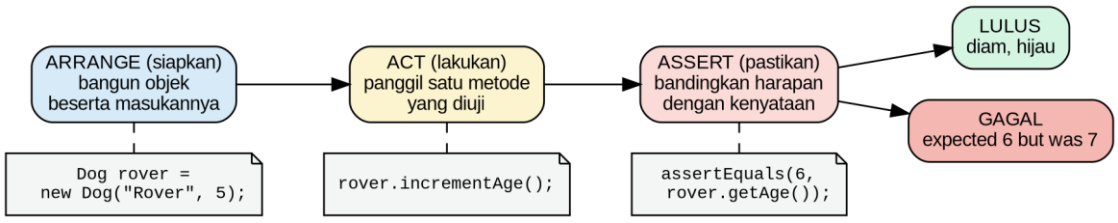
```
PASS  age is 5
FAIL  age is now 6: expected 6 but was 7
PASS  name is Rover
PASS  negative age is rejected
3 passed, 1 failed
```

dan status keluarnya 1. Pesan itu menyebutkan janji mana yang dilanggar, apa yang diharapkan, dan apa yang sebenarnya terjadi; sebagian besar nilai sebuah uji terletak pada pesan itu. Kembalikan perubahannya sebelum Anda melanjutkan.

> ⚠ **RENUNGKAN.** Python juga memiliki pernyataan bawaan `assert`: `assert rover.age == 6, "Age is now 6"` melempar `AssertionError` jika kondisinya salah. Pernyataan itu baik dipakai di dalam uji, dan pytest (Subbab 6.7) dibangun di atasnya. Namun jangan pernah memakai `assert` untuk memeriksa *data pengguna* di dalam program itu sendiri: menjalankan Python dengan opsi `-O` menghapus setiap pernyataan `assert`, dan pemeriksaan Anda lenyap diam-diam. Kata kunci `assert` di Java bahkan lebih lemah — ia *mati* kecuali program dijalankan dengan `java -ea`. Logika program memakai `if` dan eksepsi; `assert` untuk uji dan untuk pernyataan tentang kode Anda sendiri yang "tidak mungkin" salah.

6.6 Anatomi Sebuah Uji Unit

Setiap uji di **Kode 6.7**, dan setiap uji di sisa buku ini, memiliki tiga bagian yang sama, seperti tampak pada **Gambar 6.2**.



Gambar 6.2 *Arrange, act, assert*: bangun objeknya, panggil satu metode yang diuji, bandingkan nilai yang diharapkan dengan nilai sebenarnya.

Arrange (siapkan). Bangun objek yang diuji dan semua yang dibutuhkannya: anjingnya, pesanannya, rekeningnya. Buat seminimal mungkin; uji yang membangun dua puluh objek untuk memeriksa satu angka sulit dibaca dan rusak karena alasan yang tidak ada hubungannya dengan hal yang diujinya.

Act (lakukan). Panggil *satu* metode yang Anda uji. Satu uji, satu perilaku. Jika uji yang memeriksa lima perilaku gagal, Anda belum tahu yang mana dari kelima perilaku itu yang rusak.

Assert (pastikan). Bandingkan nilai yang diharapkan dengan nilai sebenarnya. Nilai yang diharapkan ditulis sebagai *literal* — `6`, bukan `5 + 1`, apalagi salinan rumus yang dipakai metode itu, karena uji yang menghitung ulang jawaban dengan rumus yang sama akan dengan senang hati menyetujui *bug* yang sama.

Ketiga bagian itu kadang disebut *given, when, then*: *diberikan* anjing berumur 5, *ketika* ia berulang tahun, *maka* umurnya 6. Ucapkan uji Anda dalam bentuk itu sebelum menuliskannya; jika kalimatnya janggal, ujinya mungkin juga janggal.

6.6.1 Membandingkan bilangan pecahan

Bilangan bulat dibandingkan secara tepat. Bilangan titik mengambang — `double` di Java, `float` di Python — biasanya tidak, karena kebanyakan pecahan desimal tidak dapat disimpan secara tepat dalam biner. Bilangan 0.1 disimpan sebagai pecahan biner yang sedikit sekali berbeda dari sepersepuluh, dan galat-galat kecil itu menumpuk. **Kode 6.9** dan **Kode 6.10** memperlihatkan kejutan klasiknya.

Kode 6.9 `DoubleEquality.java`: mengapa `==` adalah uji yang keliru untuk dua `double`.

```java
/**
 * Why two doubles should almost never be compared with ==.
 */
public class DoubleEquality {
    /**
     * Returns true when two doubles differ by at most epsilon.
     *
     * @param expected the value we hoped for
     * @param actual   the value we computed
     * @param epsilon  the largest difference we will tolerate
     * @return whether the two values are close enough
     */
    static boolean close(double expected, double actual, double epsilon) {
        return Math.abs(expected - actual) <= epsilon;
    }

    /**
     * Prints the classic 0.1 + 0.2 surprise and two ways around it.
     *
     * @param args not used
     */
    public static void main(String[] args) {
        double a = 0.1 + 0.2;
        System.out.println("0.1 + 0.2          = " + a);
        System.out.println("a == 0.3           : " + (a == 0.3));
        System.out.println("close(0.3, a, 1e-9): " + close(0.3, a, 1e-9));

        long priceA = 10_000;   // uang disimpan sebagai rupiah bulat dalam long
        long priceB = 20_000;
        System.out.println("10000 + 20000 == 30000: " + (priceA + priceB == 30_000));
    }
}
```

```
0.1 + 0.2          = 0.30000000000000004
a == 0.3           : false
close(0.3, a, 1e-9): true
10000 + 20000 == 30000: true
```

Kode 6.10 `double_equality.py`: percobaan yang sama, ditambah `math.isclose`.

```python
"""Why two floats should almost never be compared with ==."""
import math


def close(expected, actual, epsilon):
    """Return True when two floats differ by at most epsilon."""
    return abs(expected - actual) <= epsilon


a = 0.1 + 0.2
print("0.1 + 0.2          =", a)
print("a == 0.3           :", a == 0.3)
print("close(0.3, a, 1e-9):", close(0.3, a, 1e-9))
print("math.isclose(0.3, a):", math.isclose(0.3, a))

price_a = 10_000        # uang disimpan sebagai rupiah bulat dalam int
price_b = 20_000
print("10000 + 20000 == 30000:", price_a + price_b == 30_000)
```

```
0.1 + 0.2          = 0.30000000000000004
a == 0.3           : False
close(0.3, a, 1e-9): True
math.isclose(0.3, a): True
10000 + 20000 == 30000: True
```

Obatnya adalah menerima jawaban apa pun yang *cukup dekat* dengan yang diharapkan. Kita memilih toleransi kecil ε (huruf Yunani epsilon) dan menyatakan bahwa nilai yang diharapkan e dan nilai sebenarnya a cocok apabila

$$|e - a| \le \varepsilon \tag{6.2}$$

dengan $|e - a|$ nilai mutlak selisih keduanya — seberapa jauh keduanya terpisah, tanpa peduli mana yang lebih besar. Persamaan (6.2) persis sama dengan metode `close` di kedua program. Di sini $|0.3 - 0.30000000000000004| = 4 \times 10^{-17}$, jauh di bawah $\varepsilon = 10^{-9}$, sehingga uji menyatakan cocok. Setiap pustaka uji sudah memiliki perbandingan ini: `assertEquals(expected, actual, delta)` di JUnit, `assertAlmostEqual(expected, actual, delta=...)` di `unittest`, dan `actual == pytest.approx(expected)` di pytest.

Pilih ε yang cukup besar untuk menelan derau pembulatan dan cukup kecil untuk menangkap kesalahan sungguhan; untuk rupiah, 10^{-6} sudah lebih dari cukup. Lebih baik lagi untuk uang: **jangan pakai titik mengambang sama sekali**. Simpan jumlah uang sebagai rupiah bulat dalam `long` atau `int` Python, maka setiap perbandingan kembali tepat, seperti dilakukan mesin harga di Subbab 6.16.

6.7 JUnit 6 dan unittest

Metode `check` buatan tangan memang bekerja, tetapi setiap proyek akan menulis versinya sendiri yang sedikit berbeda. **Kerangka kerja uji** (*test framework*) mengerjakan tugas itu sekali, dengan baik, untuk semua orang. Sebuah kerangka kerja menyediakan tiga hal: cara menandai sebuah metode sebagai uji, pustaka metode asersi dengan pesan kegagalan yang baik, dan sebuah *runner* — program yang menemukan semua uji, menjalankan masing-masing pada objek yang segar, lalu melapor. Di Java kerangka kerja bakunya adalah **JUnit 6** [4] (mesinnya bernama *Jupiter*). Python menyertakan `unittest` [5] dalam pustaka standarnya, dan kebanyakan proyek Python juga memakai **pytest** [6].

6.7.1 Kelas uji JUnit 6

Kode 6.11 menguji `Dog` dengan JUnit 6, versi terkininya [4]. Model pemrogramannya, yang disebut *Jupiter*, adalah model yang diperkenalkan JUnit 5, sehingga komentar di dalam listing dan banyak tutorial JUnit 5 di web tetap berlaku kata demi kata.

Kode 6.11 `DogTest.java`: uji-uji `Dog` yang ditulis untuk JUnit 6.

```
import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertThrows;

import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Test;

/**
 * JUnit 5 tests for {@link Dog}.
 */
class DogTest {
    private Dog rover;

    @BeforeEach
    void setUp() {
        rover = new Dog("Rover", 5);   // anjing baru sebelum setiap uji
    }

    @Test
    @DisplayName("a new dog reports the age it was given")
    void getAgeReturnsInitialAge() {
        assertEquals(5, rover.getAge(), "Age is 5");
    }

    @Test
    @DisplayName("incrementAge adds exactly one year")
    void incrementAgeAddsOneYear() {
        rover.incrementAge();
        assertEquals(6, rover.getAge(), "Age is now 6");
    }

    @Test
    @DisplayName("a negative age is rejected")
    void negativeAgeIsRejected() {
        assertThrows(IllegalArgumentException.class, () -> new Dog("Rex", -1));
    }
}
```

Telusurilah baris demi baris.

- Baris `import static ... Assertions.assertEquals` memungkinkan kita menulis `assertEquals(...)` alih-alih `Assertions.assertEquals(...)`. Impor lainnya mendatangkan tiga **anotasi** yang kita pakai. Anotasi adalah kata berawalan `@`, ditulis di depan sebuah metode, yang memberi tahu sebuah perkakas sesuatu tentang metode itu tanpa mengubah apa yang dikerjakannya. (Anda pernah melihat satu: `@Override`.)
- **Tidak ada metode `main`.** Kita tidak menjalankan kelas uji sendiri; *runner* JUnit yang menjalankannya, dan ia mencari metode bertanda `@Test`. JUnit 6 bahkan tidak mensyaratkan kelas atau metodenya `public`.
- `@BeforeEach` menandai `setUp`, yang dijalankan JUnit *sebelum setiap metode uji*. Dengan demikian setiap uji mendapat Rover berumur lima tahun yang baru. Ini penting: `incrementAgeAddsOneYear` mengubah anjingnya, dan seandainya uji-uji itu berbagi satu anjing, hasil `getAgeReturnsInitialAge` akan bergantung pada uji mana yang kebetulan dijalankan lebih dulu. **Uji harus saling bebas dan tidak bergantung pada urutan.**

- `@DisplayName` memberi uji sebuah kalimat yang mudah dibaca untuk laporan. Tanpanya, laporan menampilkan nama metode — itulah sebabnya nama metode pun ditulis seperti kalimat kecil.
- `assertEquals(expected, actual, message)` adalah langkah *assert* dari *arrange–act–assert*. Letakkan **nilai yang diharapkan lebih dulu**: ketika uji gagal, JUnit mencetak "expected: <6> but was: <7>", dan jika argumennya tertukar, pesannya berbohong.
- `assertThrows(IllegalArgumentException.class, () -> new Dog("Rex", -1))` lulus hanya jika menjalankan kode itu melempar eksepsi jenis tersebut. Bentuk `() -> new Dog("Rex", -1)` yang tampak aneh itu adalah **ekspresi lambda**: sepotong kecil kode yang dibungkus, belum dijalankan, dan diserahkan kepada `assertThrows`, yang menjalankannya di dalam `try` miliknya sendiri. Untuk sementara, bacalah panah itu sebagai "ini kode untuk kamu jalankan nanti"; Bab 10 menjelaskan lambda dengan tuntas.

> ✓ **WAWASAN KUNCI.** JUnit adalah dua hal sekaligus: *pustaka* untuk menulis uji (`@Test`, `assertEquals`) dan *aplikasi* yang menjalankannya. Itulah sebabnya kelas uji tidak memiliki `main`: `main`-nya milik *runner*.

Banyak proyek yang ada masih memakai JUnit 4, dengan gagasan yang sama di bawah nama berbeda: `org.junit.Test`, `@Before` alih-alih `@BeforeEach`, dan **pesan di depan** dalam `assertEquals("Age is 5", 5, rover.getAge())`. Bacalah keduanya; tulislah uji baru dengan JUnit 6.

6.7.2 Menjalankan JUnit 6 di VS Code dan di terminal

JUnit bukan bagian dari JDK; ia adalah *pustaka*, sebuah berkas `.jar` (arsip zip berisi kelas-kelas terkompilasi). Pilihan paling sederhana bagi pemula adalah berkas tunggal `junit-platform-console-standalone-6.1.3.jar` (versi 6.x mana pun bekerja sama), yang memuat JUnit 6 beserta *runner*-nya. Unduh dari Maven Central lalu simpan di folder `lib` di samping berkas-berkas `.java` Anda.

Di VS Code (*Extension Pack for Java* dari Bab 1 sudah menyertakan *Test Runner for Java*), buka folder yang berisi `Dog.java`, `DogTest.java`, dan `lib`; ekstensi itu menambahkan setiap `.jar` di `lib` ke proyek (*Java Projects* → *Referenced Libraries*). Segitiga hijau muncul di samping kelas dan di samping setiap metode `@Test`; klik segitiga itu, atau buka tampilan *Testing* (ikon labu kimia) lalu klik *Run Tests*. Setiap uji mendapat centang hijau atau silang merah, dan mengeklik silang menampilkan pesan kegagalan beserta barisnya. Jika VS Code menawarkan "Enable Java Tests", terima lalu pilih *JUnit Jupiter*; ia mengunduh *jar* yang sama ke `lib`.

Di terminal, di folder yang sama (di Windows, PowerShell menerima perintah yang sama):

```
javac -cp "lib/*" -d out Dog.java DogTest.java
java -jar lib/junit-platform-console-standalone-6.1.3.jar execute -cp out --select-class
DogTest
```

Baris pertama mengompilasi kedua kelas ke `out` dengan *jar* itu di *class path* (`-cp`); baris kedua menyalakan *runner* konsol JUnit pada kelas-kelas terkompilasi itu. Ia mencetak pohon uji dengan centang atau silang di samping setiap nama tampilan, serta ringkasan jumlah uji *successful* dan *failed*. Sebuah kegagalan menampilkan nama tampilan, pesan `Age is now 6 ==> expected: <6> but was: <7>`, serta berkas dan baris asersinya.

> ⚠ **RENUNGKAN.** Program-program di bab ini dikompilasi dengan `javac` 21 dan dijalankan di mesin yang dipakai untuk menyiapkan buku ini, yang tidak memiliki akses internet dan karena itu tidak memiliki *jar* JUnit. Kedua kelas JUnit (`DogTest.java` di sini dan `PriceRuleTest.java` di Subbab 6.16) dikompilasi dan dijalankan terhadap pengganti kecil

> yang meniru beberapa anotasi dan asersi JUnit 6 yang dipakainya, dan semua ujinya lulus; putusan yang dicetak di buku ini berasal dari pemeriksa biasa `DogCheck` dan `PriceRuleCheck`, yang menjalankan pemeriksaan yang sama tanpa pustaka apa pun. Itu juga sebuah pelajaran: kerangka kerja adalah kemudahan, bukan keharusan. Ketika JUnit tidak tersedia, metode `check` dan status keluar bukan nol tetap melindungi Anda.

6.7.3 unittest

Pustaka standar Python memiliki kerangka kerjanya sendiri, `unittest`, yang meniru JUnit awal tahun 2000-an. **Kode 6.12** adalah `DogTest` sekali lagi.

Kode 6.12 `test_dog.py`: uji-uji `Dog` yang ditulis untuk `unittest`.

```
"""unittest tests for Dog."""
import unittest

from dog import Dog


class DogTest(unittest.TestCase):
    def setUp(self):
        self.rover = Dog("Rover", 5)     # anjing baru sebelum setiap uji

    def test_age_returns_initial_age(self):
        """a new dog reports the age it was given"""
        self.assertEqual(5, self.rover.age, "Age is 5")

    def test_increment_age_adds_one_year(self):
        """increment_age adds exactly one year"""
        self.rover.increment_age()
        self.assertEqual(6, self.rover.age, "Age is now 6")

    def test_negative_age_is_rejected(self):
        """a negative age is rejected"""
        with self.assertRaises(ValueError):
            Dog("Rex", -1)


if __name__ == "__main__":
    unittest.main()
```

Bentuknya bentuk Java. Kelas uji mewarisi `unittest.TestCase`. Setiap metode yang namanya diawali `test` adalah sebuah uji — namanya *itulah* penandanya, menggantikan `@Test`. `setUp` berperan sebagai `@BeforeEach`. Baris pertama *docstring* sebuah uji berperan sebagai `@DisplayName`. `self.assertEqual(expected, actual, message)` dan `with self.assertRaises(ValueError):` mengerjakan apa yang dikerjakan `assertEquals` dan `assertThrows` di JUnit; blok `with` menggantikan lambda.

Jalankan dari folder yang memuat kedua berkas dengan `python3 -m unittest -v test_dog` (`-v` meminta satu baris per uji):

```
test_age_returns_initial_age (test_dog.DogTest.test_age_returns_initial_age)
a new dog reports the age it was given ... ok
test_increment_age_adds_one_year (test_dog.DogTest.test_increment_age_adds_one_year)
increment_age adds exactly one year ... ok
test_negative_age_is_rejected (test_dog.DogTest.test_negative_age_is_rejected)
a negative age is rejected ... ok

----------------------------------------------------------------------
Ran 3 tests in 0.000s

OK
```

`python3 -m unittest` tanpa nama berkas akan *menemukan* setiap berkas bernama `test*.py` di folder itu dan menjalankan semuanya. Di VS Code, buka tampilan *Testing*, klik *Configure Python Tests*, pilih `unittest`, folder `.`, dan pola `test*.py`; ikon labu kimia lalu menampilkan centang hijau yang sama seperti untuk Java.

Dan inilah kegagalan yang sama seperti tadi — `self._age += 2` ditanam di `dog.py` — sebagaimana dilaporkan `unittest` (nama folder yang panjang di baris `File` dipersingkat di sini):

```
.F.
======================================================================
FAIL: test_increment_age_adds_one_year (test_dog.DogTest.test_increment_age_adds_one_year)
increment_age adds exactly one year
----------------------------------------------------------------------
Traceback (most recent call last):
  File ".../test_dog.py", line 18, in test_increment_age_adds_one_year
    self.assertEqual(6, self.rover.age, "Age is now 6")
AssertionError: 6 != 7 : Age is now 6

----------------------------------------------------------------------
Ran 3 tests in 0.000s

FAILED (failures=1)
```

Baris pertama, `.F.`, adalah bilah kemajuan ringkas: satu karakter per uji, titik untuk yang lulus dan `F` untuk yang gagal.

6.7.4 pytest

pytest (pasang sekali dengan `pip install pytest`) membuang segala upacara. Sebuah uji adalah fungsi apa pun yang namanya diawali `test_`, di dalam berkas yang namanya diawali `test_`; asersinya adalah pernyataan `assert` milik Python sendiri; tidak perlu kelas. Uji-uji harga di Subbab 6.16 ditulis dengan cara itu, dan memperlihatkan keunggulan terbaik pytest: ketika sebuah `assert` gagal, pytest membongkar ekspresinya dan mencetak nilai setiap bagiannya. pytest juga menjalankan kelas `unittest` tanpa perubahan, sehingga `python3 -m pytest` di sebuah folder menjalankan kedua jenis uji. **Tabel 6.2** menjajarkan ketiga kerangka kerja itu.

Tabel 6.2 Gagasan pengujian yang sama dalam JUnit 6, `unittest`, dan pytest.

Tujuan	JUnit 6 (Java)	Unittest (Python)	Pytest (Python)
menandai uji	`@Test`	nama metode `test...` dalam sebuah `TestCase`	nama fungsi `test_...`
sebelum setiap uji	`@BeforeEach`	`setUp(self)`	sebuah fungsi *fixture*
sama	`assertEquals(e, a)`	`assertEqual(e, a)`	`assert a == e`
cukup dekat	`assertEquals(e, a, delta)`	`assertAlmostEqual(e, a, delta=d)`	`assert a == pytest.approx(e)`
benar / salah	`assertTrue(c)`, `assertFalse(c)`	`assertTrue(c)`, `assertFalse(c)`	`assert c`, `assert not c`
melempar eksepsi	`assertThrows(T.class, () -> ...)`	`with self.assertRaises(T):`	`with pytest.raises(T):`
menjalankan	VS Code, atau peluncur konsol	`python -m unittest`	`python -m pytest`

6.8 Memilih Kasus Uji, dan Menulisnya Lebih Dulu

Sebuah rangkaian uji hanya sebaik masukan yang dicobanya. Mencoba `3.2` kilogram sepuluh kali tidak membuktikan lebih banyak daripada mencobanya sekali. Penguji yang baik memilih masukan secara sistematis, dengan dua teknik klasik dan satu kebiasaan [7].

6.8.1 Kelas ekuivalensi dan batas

Bagilah masukan yang mungkin ke dalam **kelas ekuivalensi**: kelompok-kelompok yang semestinya diperlakukan sama oleh kode, sehingga satu wakil dari setiap kelompok sama baiknya dengan anggota lain. Lalu ujilah **batas-batas** di antara kelompok itu, karena di sanalah pemrogram menulis `<` padahal maksudnya `<=`.

Ambil harga per kilogram laundry, yang akan dibangun di Subbab 6.16: cuci lipat berharga 7 000 rupiah per kilogram, setiap kilogram yang *sudah dimulai* dihitung (3,2 kg ditagih sebagai 4 kg), dan tak seorang pun membayar kurang dari 3 kg. Berat-berat itu jatuh ke dalam empat kelas, dan **Tabel 6.3** memilih ujinya.

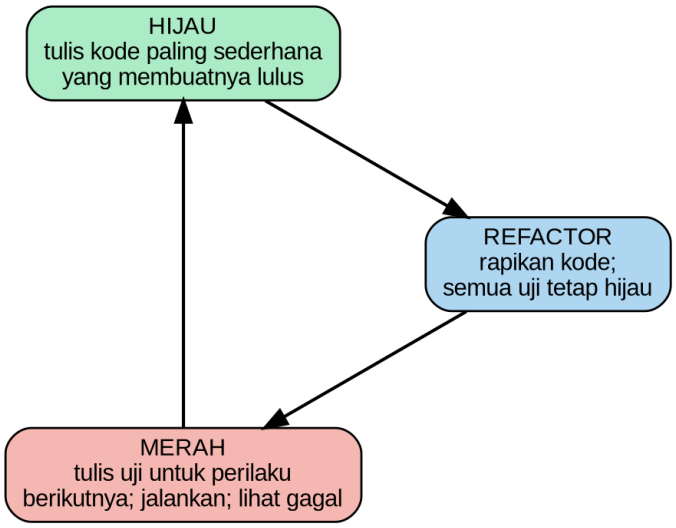
Tabel 6.3 Kelas ekuivalensi dan nilai batas untuk harga per kilogram dengan minimum 3 kg.

Kelas masukan	Wakil	Nilai batas	Diharapkan
tanpa berat (tidak sah)	0	0	eksepsi
sampai minimum	2.0	3.0	21 000 (3 kg)
sedikit lewat minimum	3.2	3.01	28 000 (4 kg)
jauh di atasnya	5.5	—	42 000 (6 kg)

Cara berpikir yang sama berlaku untuk semua yang Anda uji. Untuk larik, ujilah elemen pertama, elemen terakhir, dan larik kosong — galat indeks bersarang di ujung-ujung. Untuk setiap `if`, pastikan ada uji yang mengambil cabang *benar* dan ada yang mengambil cabang *salah*. Untuk setiap eksepsi yang dijanjikan dokumentasi sebuah metode, tulislah uji yang memancingnya. Untuk `double`, bandingkan dengan toleransi (Persamaan (6.2)).

6.8.2 Uji lebih dulu

Kebiasaannya adalah bagian yang mengejutkan: **tulislah uji sebelum kodenya**. Praktik ini disebut *test-driven development* (TDD, pengembangan berpacu uji) [8], dan berjalan dalam putaran pendek, seperti tampak pada **Gambar 6.3**.



Gambar 6.3 Siklus uji-lebih-dulu: uji yang gagal (merah), kode paling sederhana yang meluluskannya (hijau), lalu perapian dengan uji sebagai jaring pengaman (*refactor*).

Merah. Tulis satu uji untuk perilaku kecil berikutnya yang Anda inginkan, jalankan, dan *saksikan ia gagal*. Kegagalan itu membuktikan bahwa ujinya memang bisa gagal — uji yang sudah lulus sebelum kodenya ada tidak menguji apa-apa.

Hijau. Tulis kode paling sederhana yang membuat uji itu lulus. Bukan yang paling cerdik; yang paling sederhana.

Refactor (perapian). Sekarang perbaiki kodenya — ganti nama, buang duplikasi, sederhanakan — sambil menjalankan uji setelah setiap perubahan. Uji tetap hijau, sehingga Anda tahu perbaikan itu tidak mengubah hal yang penting.

Menulis uji lebih dulu memaksa Anda memutuskan apa yang harus *dikerjakan* metode, dari sisi pemanggil, sebelum perhatian Anda tersita oleh caranya. Soal Latihan 6.6 menempuh satu siklus lengkap untuk kartu loyalitas.

6.9 Kapan Dua Objek Dianggap Sama?

Sari datang ke laundry pada hari Senin, dan kasir mencatatnya sebagai “Sari”, telepon `0812-555-0101`. Pada hari Kamis, kasir lain mengetik ” Sari “, `0812 555 0101`. Apakah itu pelanggan yang sama? Bagi manusia, jelas ya. Bagi program, jawabannya sepenuhnya bergantung pada apa yang dikatakan kelasnya — dan jika kelas tidak mengatakan apa-apa, jawabannya *tidak*, dan poin loyalitas Sari terbelah di antara dua pelanggan.

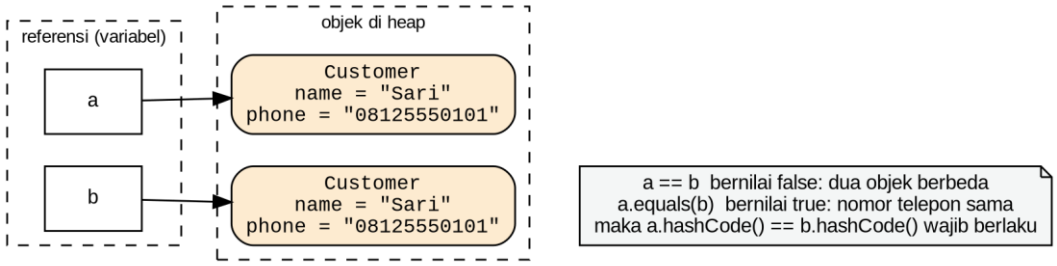
6.9.1 Identitas dan kesamaan

Setiap bahasa yang mengenal objek memiliki dua pertanyaan berbeda yang terdengar mirip.

> **DEFINISI.** Dua referensi *identik* jika keduanya menunjuk objek yang benar-benar sama di memori. Dua objek *sama* (*equal*) jika keduanya mewakili nilai yang sama, menurut aturan yang ditetapkan kelasnya. Di Java, `a == b` menguji identitas untuk objek dan `a.equals(b)`

> menguji kesamaan. Di Python, `a is b` menguji identitas dan `a == b` menguji kesamaan (dengan memanggil `a.__eq__(b)`).

Gambar 6.4 menggambarkan keadaannya: dua variabel, dua objek terpisah, isi yang sama persis.



Gambar 6.4 Dua referensi ke dua objek berbeda yang isinya sama: tidak identik, tetapi sama jika kelasnya berkata demikian — dan kalau begitu, kode *hash* keduanya juga harus sama.

Apa yang dikatakan sebuah kelas jika kita tidak menulis apa-apa? Setiap kelas Java mewarisi `equals` dari kelas `Object`, dan `Object.equals` tidak lain adalah `==`: sebuah objek hanya sama dengan dirinya sendiri [9]. `__eq__` bawaan Python berperilaku serupa. **Kode 6.13** memperlihatkan akibatnya, lalu memperlihatkan jebakan kedua: kelas yang mendefinisikan `equals` tetapi melupakan `hashCode`.

Kode 6.13 `IdentityDemo.java`: kelas tanpa `equals`, dan kelas dengan `equals` tetapi tanpa `hashCode`.

```
import java.util.ArrayList;
import java.util.HashSet;
import java.util.List;
import java.util.Set;

/** A customer class that overrides nothing: it inherits equals, hashCode and toString from
Object. */
class PlainCustomer {
    final String name;
    final String phone;

    PlainCustomer(String name, String phone) {
        this.name = name;
        this.phone = phone;
    }
}

/** A customer class that overrides equals but forgets hashCode. */
class HalfCustomer {
    final String name;
    final String phone;

    HalfCustomer(String name, String phone) {
        this.name = name;
        this.phone = phone;
    }

    @Override
    public boolean equals(Object other) {
        if (!(other instanceof HalfCustomer)) {
            return false;
        }
        HalfCustomer that = (HalfCustomer) other;
        return phone.equals(that.phone);
    }
}
```

Kode 6.13 (Lanjutan) *IdentityDemo.java: kelas tanpa equals, dan kelas dengan equals tetapi tanpa hashCode.*

```
/**
 * What happens to equality when a class does not define it properly.
 */
public class IdentityDemo {
    /**
     * Hides the memory-dependent hexadecimal part of a default toString.
     *
     * @param o any object
     * @return its toString with the hexadecimal hash replaced by a placeholder
     */
    static String masked(Object o) {
        return o.toString().replaceAll("@[0-9a-f]+", "@<hash>");
    }

    /**
     * Compares two customers typed in twice from the same data.
     *
     * @param args not used
     */
    public static void main(String[] args) {
        PlainCustomer a = new PlainCustomer("Sari", "0812-555-0101");
        PlainCustomer b = new PlainCustomer("Sari", "0812-555-0101");
        System.out.println("PlainCustomer");
        System.out.println("  a == b      : " + (a == b));
        System.out.println("  a.equals(b) : " + a.equals(b));
        System.out.println("  a prints as : " + masked(a));
        Set<PlainCustomer> plainSet = new HashSet<>();
        plainSet.add(a);
        plainSet.add(b);
        System.out.println("  set size    : " + plainSet.size());

        HalfCustomer c = new HalfCustomer("Sari", "0812-555-0101");
        HalfCustomer d = new HalfCustomer("Sari", "0812-555-0101");
        System.out.println("HalfCustomer (equals without hashCode)");
        System.out.println("  c.equals(d) : " + c.equals(d));
        List<HalfCustomer> list = new ArrayList<>();
        list.add(c);
        System.out.println("  list contains d: " + list.contains(d));   // List hanya
memakai equals
        Set<HalfCustomer> halfSet = new HashSet<>();
        halfSet.add(c);
        System.out.println("  set contains d : " + halfSet.contains(d)); // HashSet memakai
hashCode lebih dahulu
        halfSet.add(d);
        System.out.println("  set size       : " + halfSet.size());
    }
}
```

```
PlainCustomer
  a == b      : false
  a.equals(b) : false
  a prints as : PlainCustomer@<hash>
  set size    : 2
HalfCustomer (equals without hashCode)
  c.equals(d) : true
  list contains d: true
  set contains d : false
  set size       : 2
```

Blok pertama adalah yang kita khawatirkan: dua objek `PlainCustomer` yang diketik dari data yang sama tidak identik dan tidak pula sama, dan sebuah `HashSet` menampung keduanya. `PlainCustomer@<hash>` adalah `toString` bawaan dari `Object`: nama kelas dan sebuah bilangan

heksadesimal yang berubah dari satu jalan ke jalan berikutnya (program mencetak `<hash>` agar keluaran di sini sama dengan keluaran Anda).

Blok kedua lebih buruk, karena tampaknya sudah beres. `HalfCustomer` menimpa `equals`, `c.equals(d)` bernilai `true`, dan sebuah `ArrayList` menemukan `d`. Namun `HashSet` menyatakan `d` tidak ada di dalamnya, dan setelah `d` ditambahkan, himpunan itu berisi dua elemen.

Penyebabnya adalah cara kerja himpunan *hash*. Ia tidak membandingkan elemen baru dengan setiap elemen yang dimuatnya; ia lebih dulu meminta sebuah **kode *hash***, sebuah `int` yang dihitung dari objek, lalu memakainya untuk memilih satu *ember* (*bucket*) kecil. Hanya elemen-elemen di ember itu yang dibandingkan dengan `equals`. `HalfCustomer` mewarisi `hashCode` dari `Object`, yang memberi objek berbeda bilangan berbeda, sehingga `c` dan `d` jatuh ke ember berbeda dan `equals` tidak pernah dipanggil. `javac -Xlint:all IdentityDemo.java` memperingatkan persis hal ini:

```
IdentityDemo.java:18: warning: [overrides] Class HalfCustomer overrides equals, but neither
it nor any superclass overrides hashCode method
class HalfCustomer {
^
1 warning
```

Aturan yang membuat himpunan dan peta *hash* bekerja itu pendek. Untuk dua objek sembarang a dan b dan fungsi *hash* h,

$$a \text{ sama dengan } b \;\Rightarrow\; h(a) = h(b) \tag{6.3}$$

Panah ⇒ dibaca “mengakibatkan”: *jika* dua objek sama, *maka* kode *hash* keduanya harus sama. Persamaan (6.3) tidak berlaku ke arah sebaliknya — dua objek berbeda boleh berbagi kode *hash* (sebuah *tabrakan*), dan itu hanya sedikit memperlambat, karena `equals` memilah keduanya di dalam ember. Aturan itu baru dilanggar jika objek-objek yang sama mendapat kode berbeda, dan saat itulah himpunan dan peta berhenti bekerja. Akibat praktisnya: **setiap kali Anda menimpa `equals`, timpa juga `hashCode`, dengan *field* yang sama.**

Python lebih ketat dalam hal ini, dan itu justru lebih ramah. **Kode 6.14** mengulang percobaannya.

Kode 6.14 `identity_demo.py`: dua kesalahan yang sama dalam Python; kesalahan kedua tertangkap.

```
"""What happens to equality when a class does not define it properly."""
import re


class PlainCustomer:
    """A customer class that defines nothing: it inherits object's behavior."""

    def __init__(self, name, phone):
        self.name = name
        self.phone = phone


class HalfCustomer:
    """A customer class that defines __eq__ but not __hash__."""

    def __init__(self, name, phone):
        self.name = name
        self.phone = phone

    def __eq__(self, other):
        if not isinstance(other, HalfCustomer):
            return NotImplemented
        return self.phone == other.phone


def masked(o):
    """Hide the memory address in a default repr."""
    return re.sub(r"0x[0-9a-fA-F]+", "0x<address>", repr(o))


a = PlainCustomer("Sari", "0812-555-0101")
b = PlainCustomer("Sari", "0812-555-0101")
print("PlainCustomer")
print("  a is b      :", a is b)
print("  a == b      :", a == b)
print("  a prints as :", masked(a))
print("  set size    :", len({a, b}))
```

Kode 6.14 (Lanjutan) *identity_demo.py: dua kesalahan yang sama dalam Python; kesalahan kedua tertangkap.*

```
c = HalfCustomer("Sari", "0812-555-0101")
d = HalfCustomer("Sari", "0812-555-0101")
print("HalfCustomer (__eq__ without __hash__)")
print("  c == d      :", c == d)
print("  d in [c]    :", d in [c])          # list hanya memakai ==
print("  __hash__ is :", HalfCustomer.__hash__)
try:
    {c, d}                                  # set memerlukan hash()
except TypeError as e:
    print("  set fails   :", e)
```

```
PlainCustomer
  a is b      : False
  a == b      : False
  a prints as : <__main__.PlainCustomer object at 0x<address>>
  set size    : 2
HalfCustomer (__eq__ without __hash__)
  c == d      : True
  d in [c]    : True
  __hash__ is : None
  set fails   : unhashable type: 'HalfCustomer'
```

Jika sebuah kelas Python mendefinisikan `__eq__` tetapi tidak `__hash__`, Python menjadikan `__hash__` bernilai `None`, sehingga objek-objeknya **tidak dapat di-*hash***: objek itu sama sekali tidak

dapat dimasukkan ke himpunan atau dipakai sebagai kunci kamus. Anda mendapat `TypeError` yang jelas saat itu juga, bukan jawaban salah yang diam-diam. (Versi Python yang lebih baru menyusun pesannya sedikit berbeda; maknanya sama.)

6.9.2 Menulis equals dan hashCode dengan benar

Kode 6.15 adalah kelas `Customer` yang ditulis dengan benar. Seorang pelanggan dikenali dari nomor teleponnya, jadi konstruktor lebih dulu *menormalkan* nomor itu — hanya menyimpan angkanya — lalu `equals` dan `hashCode` memakai telepon yang sudah dinormalkan dan tidak memakai yang lain. Nama sengaja diabaikan: “Sari” dan “Sari W.” dengan telepon yang sama adalah orang yang sama.

Kode 6.15 `Customer.java`: kesamaan berdasarkan nomor telepon, `hashCode` yang serasi, dan `toString` yang mudah dibaca.

```
import java.util.HashMap;
import java.util.HashSet;
import java.util.Map;
import java.util.Set;

/**
 * A laundry customer, identified by phone number.
 *
 * <p>Two {@code Customer} objects are equal when their phone numbers
 * have the same digits, however the numbers were typed.</p>
 */
public final class Customer {
    private final String name;
    private final String phone;   // angka saja, mis. "08125550101"

    /**
     * Creates a customer.
     *
     * @param name  the customer's name as spoken at the counter
     * @param phone the phone number in any format, e.g. "0812-555-0101"
     */
    public Customer(String name, String phone) {
        this.name = name.strip();
        this.phone = phone.replaceAll("[^0-9]", "");   // simpan angkanya saja
    }

    /**
     * Gets the name.
     *
     * @return the name
     */
    public String getName() {
        return name;
    }

    /**
     * Gets the phone number.
     *
     * @return the phone number, digits only
     */
    public String getPhone() {
        return phone;
    }
```

Kode 6.15 (Lanjutan) *Customer.java: kesamaan berdasarkan nomor telepon, hashCode yang serasi, dan toString yang mudah dibaca.*

```
    /**
     * Two customers are equal when they have the same phone number.
     *
     * @param other the object to compare with
     * @return true if {@code other} is a Customer with the same phone number
     */
    @Override
    public boolean equals(Object other) {
        if (this == other) {
            return true;                        // objek yang persis sama
        }
        if (!(other instanceof Customer)) {
            return false;                       // null, atau kelas lain
        }
        Customer that = (Customer) other;
        return phone.equals(that.phone);
    }

    /**
     * Equal customers must have equal hash codes, so hash the phone number only.
     *
     * @return a hash code built from the phone number
     */
    @Override
    public int hashCode() {
        return phone.hashCode();
    }

    /**
     * Describes the customer for logs and debugging.
     *
     * @return for example {@code Customer[name=Sari, phone=08125550101]}
     */
    @Override
    public String toString() {
        return "Customer[name=" + name + ", phone=" + phone + "]";
    }
```

Kode 6.15 (Lanjutan) *Customer.java: kesamaan berdasarkan nomor telepon, hashCode yang serasi, dan toString yang mudah dibaca.*

```
    /**
     * Shows equals, hashCode and toString at work.
     *
     * @param args not used
     */
    public static void main(String[] args) {
        Customer a = new Customer("Sari", "0812-555-0101");
        Customer b = new Customer(" Sari ", "0812 555 0101");
        Customer budi = new Customer("Budi", "0813-222-0202");
        System.out.println(a);
        System.out.println(b);
        System.out.println("a == b      : " + (a == b));
        System.out.println("a.equals(b) : " + a.equals(b));
        System.out.println("same hash   : " + (a.hashCode() == b.hashCode()));
        System.out.println("a.equals(budi): " + a.equals(budi));
        System.out.println("a.equals(null): " + a.equals(null));

        Set<Customer> customers = new HashSet<>();
        customers.add(a);
        customers.add(b);
        customers.add(budi);
        System.out.println("distinct customers: " + customers.size());

        Map<Customer, Integer> visits = new HashMap<>();
        Customer[] counter = {a, budi, b, a};   // empat kunjungan yang diketik di kasir
        for (Customer c : counter) {
            visits.put(c, visits.getOrDefault(c, 0) + 1);
        }
        System.out.println("visits by Sari: " + visits.get(new Customer("Sari",
"08125550101")));
        System.out.println("visits by Budi: " + visits.get(budi));
    }
}
```

```
Customer[name=Sari, phone=08125550101]
Customer[name=Sari, phone=08125550101]
a == b      : false
a.equals(b) : true
same hash   : true
a.equals(budi): false
a.equals(null): false
distinct customers: 2
visits by Sari: 3
visits by Budi: 1
```

Bacalah `equals` baris demi baris, karena empat langkah yang sama muncul di hampir setiap `equals` yang akan Anda tulis.

1. `if (this == other) return true;` — sebuah objek selalu sama dengan dirinya sendiri. Ini juga jalan keluar cepat untuk kasus yang umum.
2. `if (!(other instanceof Customer)) return false;` — parameternya bertipe `Object`, jadi apa saja bisa datang: sebuah `String`, sebuah `Dog`, atau `null`. `instanceof` menjawab "apakah objek ini sebuah `Customer` (atau subkelasnya)?", dan menjawab `false` untuk `null`, sehingga satu baris ini menangani kedua kasus. Jangan pernah menulis `equals(Customer other)`: itu tidak *menimpa* (*override*) `Object.equals(Object)`, melainkan *membebani* (*overload*) nama itu (Subbab 6.11), dan koleksi akan tetap memanggil versi lama. Anotasi `@Override` menangkap kesalahan persis ini saat kompilasi.

3. `Customer that = (Customer) other;` — setelah kita tahu objek itu sebuah `Customer`, sebuah *cast* meminta kompilator memperlakukannya demikian, sehingga kita boleh membaca `that.phone`. Subbab 6.14.1 membahas *cast* lebih jauh.
4. `return phone.equals(that.phone);` — bandingkan *field* yang menentukan kesamaan. *Field* berupa objek dibandingkan dengan `equals`, tidak pernah dengan `==`: dua objek `String` dengan karakter yang sama itu sama, tetapi belum tentu identik.

`hashCode` mengembalikan `phone.hashCode()` — *hash* dari persis *field* yang dibandingkan `equals`. Jika kesamaan memakai dua *field*, `Objects.hash(field1, field2)` menggabungkan keduanya (Soal Latihan 6.7 mengerjakannya).

Keluarannya menegaskan setiap janji: `a == b` bernilai `false` tetapi `a.equals(b)` bernilai `true`; kode *hash* keduanya cocok; himpunan berisi dua pelanggan berbeda, bukan tiga; dan penghitung kunjungan, sebuah `HashMap` dari pelanggan ke jumlah kunjungan, menghitung Sari tiga kali meskipun ia diketik dengan dua cara.

Aturan yang diwajibkan Java untuk setiap `equals` [9], [10] layak dinyatakan sekali, untuk sembarang referensi bukan-null `x`, `y`, dan `z`:

> **DEFINISI (kontrak equals).** *Refleksif*: `x.equals(x)` bernilai benar. *Simetris*: `x.equals(y)` benar tepat ketika `y.equals(x)` benar. *Transitif*: jika `x.equals(y)` dan `y.equals(z)`, maka `x.equals(z)`. *Konsisten*: pemanggilan berulang memberi jawaban yang sama selama objeknya tidak berubah. *Null*: `x.equals(null)` bernilai salah. Dan, dari Persamaan (6.3): objek-objek yang sama memiliki kode *hash* yang sama.

Versi Python, **Kode 6.16**, mengikuti rencana yang sama.

Kode 6.16 `customer.py`: `__eq__`, `__hash__` yang serasi, serta `__repr__` dan `__str__`.

```
"""A laundry customer, identified by phone number."""
import re


class Customer:
    """A laundry customer, identified by phone number.

    Two Customer objects are equal when their phone numbers have the
    same digits, however the numbers were typed.
    """

    def __init__(self, name, phone):
        """Create a customer.

        Args:
            name: the customer's name as spoken at the counter.
            phone: the phone number in any format, e.g. "0812-555-0101".
        """
        self._name = name.strip()
        self._phone = re.sub(r"[^0-9]", "", phone)   # simpan angkanya saja

    @property
    def name(self):
        """The name."""
        return self._name

    @property
    def phone(self):
        """The phone number, digits only."""
        return self._phone

    def __eq__(self, other):
        """Two customers are equal when they have the same phone number."""
        if self is other:
            return True                     # objek yang persis sama
        if not isinstance(other, Customer):
            return NotImplemented           # biarkan Python mencoba sisi lain, lalu
menjawab False
        return self._phone == other._phone

    def __hash__(self):
        """Equal customers must have equal hashes, so hash the phone number only."""
        return hash(self._phone)
```

Kode 6.16 (Lanjutan) *customer.py: __eq__, __hash__ yang serasi, serta __repr__ dan __str__.*

```
    def __repr__(self):
        """The unambiguous form, for developers: looks like the constructor call."""
        return f"Customer(name={self._name!r}, phone={self._phone!r})"

    def __str__(self):
        """The friendly form, for receipts and messages."""
        return f"{self._name} ({self._phone})"


if __name__ == "__main__":
    a = Customer("Sari", "0812-555-0101")
    b = Customer(" Sari ", "0812 555 0101")
    budi = Customer("Budi", "0813-222-0202")
    print(a)                    # print() memakai __str__
    print(repr(b))              # repr() memakai __repr__
    print([a, budi])            # list menampilkan butirnya dengan __repr__
    print("a is b      :", a is b)
    print("a == b      :", a == b)
    print("same hash   :", hash(a) == hash(b))
    print("a == budi   :", a == budi)
    print("a == None   :", a == None)

    customers = {a, b, budi}
    print("distinct customers:", len(customers))

    visits = {}
    for c in [a, budi, b, a]:   # empat kunjungan yang diketik di kasir
        visits[c] = visits.get(c, 0) + 1
    print("visits by Sari:", visits[Customer("Sari", "08125550101")])
    print("visits by Budi:", visits[budi])
```

```
Sari (08125550101)
Customer(name='Sari', phone='08125550101')
[Customer(name='Sari', phone='08125550101'), Customer(name='Budi', phone='08132220202')]
a is b      : False
a == b      : True
same hash   : True
a == budi   : False
a == None   : False
distinct customers: 2
visits by Sari: 3
visits by Budi: 1
```

Ada dua rincian Python yang berbeda dari Java. Pertama, ketika objek lainnya bukan `Customer`, `__eq__` mengembalikan nilai khusus `NotImplemented`, bukan `False`. Nilai itu berarti "saya tidak tahu cara membandingkan dengan benda itu"; Python lalu memberi objek lainnya kesempatan menjawab, dan jika objek itu juga tidak bisa, hasil `==` adalah `False` — itulah sebabnya `a == None` mencetak `False`. Kedua, `__hash__` meng-*hash field* yang sama dengan yang dibandingkan `__eq__`, sehingga Persamaan (6.3) tetap benar.

> ✓ **WAWASAN KUNCI.** Anda jarang perlu menulis metode-metode ini dengan tangan untuk kelas yang *hanya berisi data*. *Record* Java — `record CustomerRecord(String name, String phone) {}` — mendapat `equals`, `hashCode`, dan `toString` yang ditulis oleh kompilator, dan `@dataclass(frozen=True)` Python mendapat `__eq__`, `__hash__`, dan `__repr__`. Namun keduanya membandingkan *setiap field*, sehingga "Sari" dan "Sari W." akan dianggap dua orang berbeda. Ketika kesamaan berarti sesuatu yang lebih sempit daripada "semua *field* sama" — seperti pada `Customer` — Anda menulis metodenya sendiri. Soal Latihan 6.8 membandingkan kedua pendekatan.

6.10 Menggambarkan Sebuah Objek: toString, str, dan repr

Ketika Java membutuhkan sebuah objek sebagai teks — dalam `System.out.println(x)`, dalam `"Customer: " + x`, atau di *debugger* — ia memanggil `x.toString()`. Versi yang diwarisi dari `Object` nyaris tidak berkata apa-apa kepada manusia, seperti diperlihatkan `IdentityDemo`, jadi timpalah metode itu di setiap kelas yang objeknya mungkin Anda cetak. Bentuk lazimnya, yang dipakai *record* dan ditiru **Kode 6.15**, adalah `ClassName[field=value, field=value]`.

Python membagi tugas itu menjadi dua, dan **Kode 6.16** menulis keduanya:

- `__repr__` adalah bentuk yang **tidak ambigu, untuk pengembang**. Idealnya ia tampak seperti ekspresi yang akan membangun kembali objek itu, misalnya `Customer(name='Sari', phone='08125550101')`. *Prompt* interaktif, *debugger*, `repr(x)`, dan list yang mencetak butir-butirnya semuanya memakai bentuk ini. (`!r` di dalam *f-string* meminta `repr` dari *field*-nya; dari situlah tanda petik di sekitar `'Sari'` berasal.)
- `__str__` adalah bentuk yang **ramah, untuk pengguna**, dipakai oleh `print(x)` dan `str(x)`: `Sari (08125550101)`, cocok untuk nota.

Jika sebuah kelas hanya mendefinisikan `__repr__`, `str()` akan memakainya sebagai cadangan. Pada keluaran **Kode 6.16**, `print(a)` memakai `__str__`, `print(repr(b))` memakai `__repr__`, dan mencetak sebuah *list* memakai `__repr__` untuk setiap elemennya.

Java hanya memiliki `toString`, jadi tulislah bentuk untuk pengembang di sana dan sebuah metode bernama tersendiri, seperti `receiptLine()`, untuk format yang ditujukan kepada pelanggan, sebagaimana dilakukan program laundry di Subbab 6.16.

> ⚠ **JEBAKAN.** Jangan pernah memasukkan rahasia ke dalam `toString` atau `__repr__`. Metode-metode ini berakhir di berkas log, laporan galat, dan jendela *debugger*. Nomor telepon lengkap seorang pelanggan sudah di ambang batas; kata sandi, token akses, atau nomor kartu identitas lengkap tidak boleh muncul di sana sama sekali. Menurut undang-undang pelindungan data pribadi Indonesia (UU PDP 27/2022), pencatatan data pribadi yang sembarangan adalah persoalan hukum yang nyata, bukan sekadar soal gaya.

6.11 Banyak Versi Satu Metode: Overloading

Inilah gangguan kecil yang nyata. Kita membutuhkan rata-rata saldo dari larik rekening bank, dan di tempat lain rata-rata luas dari larik negara. **Kode 6.17** menulis keduanya.

Kode 6.17 `AverageTwice.java`: dua metode dengan nama yang sama dan tubuh yang nyaris sama.

```java
/**
 * Two nearly identical average methods: the duplication that interfaces remove.
 */
public class AverageTwice {
    /** A bank account; nested here so its name cannot clash with other files. */
    static class Account {
        private final String id;
        private final String owner;
        private final double balance;

        Account(String id, String owner, double balance) {
            this.id = id;
            this.owner = owner;
            this.balance = balance;
        }

        double getBalance() {
            return balance;
        }
    }

    /** A country with an area in square kilometers. */
    static class Country {
        private final String name;
        private final double area;

        Country(String name, double area) {
            this.name = name;
            this.area = area;
        }

        double getArea() {
            return area;
        }
    }
```

Kode 6.17 (Lanjutan) *AverageTwice.java: dua metode dengan nama yang sama dan tubuh yang nyaris sama.*

```
    /**
     * Computes the average balance of some accounts.
     *
     * @param objects the accounts
     * @return the average balance, or 0 for an empty array
     */
    public static double average(Account[] objects) {
        if (objects.length == 0) {
            return 0;
        }
        double sum = 0.0;
        for (Account obj : objects) {       // menelusuri setiap objek di dalam larik
            sum = sum + obj.getBalance();
        }
        return sum / objects.length;
    }

    /**
     * Computes the average area of some countries.
     *
     * @param objects the countries
     * @return the average area, or 0 for an empty array
     */
    public static double average(Country[] objects) {
        if (objects.length == 0) {
            return 0;
        }
        double sum = 0.0;
        for (Country obj : objects) {       // gelung yang sama, getter yang berbeda
            sum = sum + obj.getArea();
        }
        return sum / objects.length;
    }

    /**
     * Calls both versions of average.
     *
     * @param args not used
     */
    public static void main(String[] args) {
        Account[] accounts = {
            new Account("01", "Aleksandra", 100),
            new Account("02", "Natasha", 150),
            new Account("03", "Sergei", 125)
        };
        Country[] countries = {
```

Kode 6.17 (Lanjutan) *AverageTwice.java: dua metode dengan nama yang sama dan tubuh yang nyaris sama.*

```
            new Country("Indonesia", 1_904_569),
            new Country("Malaysia", 330_803),
            new Country("Singapore", 734)
        };
        System.out.println("Average balance is: " + average(accounts));
        System.out.println("Average area is: " + average(countries));
    }
}
```

```
Average balance is: 125.0
Average area is: 745368.6666666666
```

(`Account` dan `Country` ditulis *di dalam* `AverageTwice` sebagai deklarasi `static class` agar namanya tidak bentrok dengan versi yang lebih baik di Subbab 6.12. Untuk sementara, bacalah kelas statis bersarang sebagai kelas pembantu yang tinggal di dalam berkas kelas lain.)

Satu hal di sini baik: kedua metode bernama `average`, dan Java tidak kesulitan membedakannya. Inilah **overloading metode** (pembebanan metode).

> **DEFINISI.** Sebuah nama metode *di-overload* jika sebuah kelas mendeklarasikan beberapa metode dengan nama itu dan *daftar parameter* yang berbeda — jumlah parameter yang berbeda, atau tipe yang berbeda pada suatu posisi. Nama bersama tipe-tipe parameternya adalah *signature* (tanda tangan) metode itu, dan *signature* harus unik di dalam satu kelas. Kompilator memilih versi yang dipanggil dengan melihat tipe argumen di tempat pemanggilan.

Anda sebenarnya sudah pernah melakukan *overloading*: kedua konstruktor `BankAccount` adalah *overload*, dan `System.out.println` memiliki sepuluh versi, untuk `int`, `double`, `String`, `Object`, dan seterusnya. Tipe kembalian **bukan** bagian dari *signature*: dua metode yang hanya berbeda pada apa yang dikembalikannya tidak dapat tinggal dalam satu kelas, karena pada pemanggilan seperti `price(3);` kompilator tidak dapat memilih. `javac` menolak:

```
Oops.java:3: error: method price(int) is already defined in class Oops
    static long price(int kg) { return kg * 7000L; }
                ^
1 error
```

Hal yang buruk pada **Kode 6.17** adalah duplikasinya: kedua tubuh metode sama persis kecuali satu tipe dan satu nama *getter*. *Overloading* memungkinkan kita *menamai* keduanya sama; ia sama sekali tidak memungkinkan kita *menulisnya* sekali saja. Subbab 6.12 membereskannya dengan sebuah antarmuka. Sebelum itu, mari kita lihat dengan tepat bagaimana Java memilih di antara *overload*, lalu bagaimana Python, yang sama sekali tidak dapat melakukan *overloading*, menyelesaikan pekerjaan yang sama.

6.11.1 Cara Java memilih sebuah overload

Kode 6.18 mendeklarasikan tiga versi `describe` dan memanggilnya dengan enam argumen berbeda.

Kode 6.18 `OverloadDemo.java`: versi mana dari metode yang di-*overload* yang dipilih setiap pemanggilan?

```java
/**
 * Method overloading: one name, several parameter lists.
 */
public class OverloadDemo {
    static String describe(int pieces) {
        return "int version    -> " + pieces + " pieces";
    }

    static String describe(double kilos) {
        return "double version -> " + kilos + " kg";
    }

    static String describe(String customer, int pieces) {
        return "(String, int)  -> " + customer + " brings " + pieces + " pieces";
    }

    /**
     * Calls describe with arguments of different types and shows which version runs.
     *
     * @param args not used
     */
    public static void main(String[] args) {
        System.out.println("describe(3)         : " + describe(3));
        System.out.println("describe(3.5)       : " + describe(3.5));
        System.out.println("describe(\"Sari\", 4) : " + describe("Sari", 4));
        byte small = 2;
        System.out.println("describe(small)     : " + describe(small));  // byte diperlebar
menjadi int
        System.out.println("describe(3L)        : " + describe(3L));     // long diperlebar
menjadi double!
        System.out.println("describe('A')       : " + describe('A'));    // char diperlebar
menjadi int!
    }
}
```

```
describe(3)         : int version    -> 3 pieces
describe(3.5)       : double version -> 3.5 kg
describe("Sari", 4) : (String, int)  -> Sari brings 4 pieces
describe(small)     : int version    -> 2 pieces
describe(3L)        : double version -> 3.0 kg
describe('A')       : int version    -> 65 pieces
```

Tiga baris pertama tidak mengejutkan: `int` menuju `describe(int)`, `double` menuju `describe(double)`, dan dua argumen menuju versi berparameter dua. Tiga baris terakhir memperlihatkan aturan yang sebenarnya diikuti kompilator [11]. Jika tidak ada versi yang cocok *persis* dengan tipe argumen, kompilator mencari versi yang dapat dicapai dengan **pelebaran** (*widening*) argumen — mengubahnya menjadi tipe yang dapat menampung setiap nilai tipe asalnya. `byte` melebar menjadi `int`, sehingga `describe(small)` memilih versi `int`. `long` tidak dapat menjadi `int` (mungkin tidak muat) tetapi *dapat* melebar menjadi `double`, sehingga `describe(3L)` diam-diam memilih versi kilogram dan mencetak `3.0 kg`. Dan `char`, bagi Java, adalah bilangan kecil tak bertanda — kode karakternya — sehingga `describe('A')` melebarkan `'A'` menjadi `int` 65 dan melaporkan 65 potong. Baru jika pelebaran gagal, kompilator mencoba *boxing* (mengubah `int` menjadi `Integer`), dan baru setelah itu ia mencoba metode dengan jumlah argumen yang berubah-ubah. Jika dua versi sama baiknya, pemanggilan itu ambigu dan tidak dapat dikompilasi.

Pelajarannya bukan menghafal fase-fase itu, melainkan menghindari rancangan yang bergantung padanya: lakukan *overloading* hanya jika versi-versinya mengerjakan *hal yang sama* untuk jenis masukan yang berbeda. `describePieces` dan `describeKilos` akan membuat kedua kejutan itu mustahil terjadi.

6.11.2 Python tidak mengenal overloading — dan tidak memerlukannya

Python mencari metode hanya berdasarkan *nama*. Pernyataan `def` sekadar memberikan sebuah fungsi kepada sebuah nama, sehingga `def average` kedua menggantikan yang pertama, persis seperti `x = ...` kedua menggantikan nilai pertama `x`. **Kode 6.19** mencoba menyalin **Kode 6.17** secara harfiah.

Kode 6.19 `average_twice.py`: di Python, `def average` kedua menggantikan yang pertama.

```
"""Python cannot keep two functions with the same name: the second replaces the first."""


class Account:
    def __init__(self, id, owner, balance):
        self.id = id
        self.owner = owner
        self.balance = balance


class Country:
    def __init__(self, name, area):
        self.name = name
        self.area = area


def average(objects):
    """Average balance of some accounts."""
    if len(objects) == 0:
        return 0
    total = 0.0
    for obj in objects:
        total = total + obj.balance
    return total / len(objects)


def average(objects):          # nama yang sama lagi: definisi ini MENGGANTIKAN yang pertama
    """Average area of some countries."""
    if len(objects) == 0:
        return 0
    total = 0.0
    for obj in objects:
        total = total + obj.area
    return total / len(objects)
```

Kode 6.19 (Lanjutan) *average_twice.py: di Python, def average kedua menggantikan yang pertama.*

```
accounts = [Account("01", "Aleksandra", 100),
            Account("02", "Natasha", 150),
            Account("03", "Sergei", 125)]
countries = [Country("Indonesia", 1_904_569),
             Country("Malaysia", 330_803),
             Country("Singapore", 734)]
print("Average area is:", average(countries))
try:
    print("Average balance is:", average(accounts))
except AttributeError as e:
    print("Average balance failed:", e)
```

```
Average area is: 745368.6666666666
Average balance failed: 'Account' object has no attribute 'area'
```

Negara-negara berhasil, rekening-rekening gagal: hanya fungsi kedua yang ada, dan tidak ada yang dilaporkan sampai jenis objek yang salah bertemu dengannya. (Pemeriksa tipe mypy memang menangkapnya, dengan `Name "average" already defined on line 17`.)

Python menangani kegunaan *overloading* dengan tiga perkakas lain, semuanya ada di **Kode 6.20**.

Kode 6.20 `overload_demo.py`: argumen bawaan, `*args`, dan `functools.singledispatch`.

```
"""Python's answers to overloading: default arguments, *args and singledispatch."""
from functools import singledispatch


# 1. Argumen bawaan: satu fungsi, parameter opsional
def describe_order(pieces, customer=None):
    if customer is None:
        return f"{pieces} pieces"
    return f"{customer} brings {pieces} pieces"


# 2. *args: berapa pun argumen posisional, dikumpulkan ke dalam tuple
def total(*prices):
    return sum(prices)


# 3. singledispatch: pilih implementasi menurut tipe argumen PERTAMA
@singledispatch
def describe(value):
    return f"general version -> {value!r}"


@describe.register
def _(value: int):
    return f"int version     -> {value} pieces"


@describe.register
def _(value: float):
    return f"float version   -> {value} kg"


print("describe_order(3)        :", describe_order(3))
print("describe_order(4, 'Sari'):", describe_order(4, "Sari"))
print("total()                  :", total())
print("total(7000, 21000, 35000):", total(7000, 21000, 35000))
print("describe(3)              :", describe(3))
print("describe(3.5)            :", describe(3.5))
print("describe('Sari')         :", describe("Sari"))
print("describe(True)           :", describe(True))    # bool adalah subkelas int!
```

```
describe_order(3)        : 3 pieces
describe_order(4, 'Sari'): Sari brings 4 pieces
total()                  : 0
total(7000, 21000, 35000): 63000
describe(3)              : int version     -> 3 pieces
describe(3.5)            : float version   -> 3.5 kg
describe('Sari')         : general version -> 'Sari'
describe(True)           : int version     -> True pieces
```

Argumen bawaan menangani kasus yang paling umum, ketika versi-versinya berbeda karena ada yang dihilangkan, seperti `BankAccount` di **Kode 6.2**. `*args` menangani "berapa pun banyaknya argumen": tanda bintang mengumpulkan semua argumen posisional ke dalam sebuah *tuple*, sehingga `total()` tanpa argumen dan `total(7000, 21000, 35000)` dengan tiga argumen adalah fungsi yang sama. Dan ketika perilaku memang bergantung pada *tipe* argumen, `functools.singledispatch` [12] menyediakan *overloading* berdasarkan tipe argumen pertama:

`@singledispatch` menandai versi umum, dan `@describe.register` menambahkan satu versi untuk setiap tipe yang disebut dalam anotasi parameternya. Perhatikan baris terakhir: `describe(True)` memilih versi `int`, karena di Python `bool` adalah subkelas `int`. Itu jebakan yang sama dengan `char` Java yang menjadi 65, dalam bahasa lain. **Tabel 6.4** merangkumnya.

Tabel 6.4 *Overloading* di Java dan perkakas Python yang mengerjakan tugas yang sama.

Kebutuhan	Java	Python
parameter opsional	*overload* yang saling memanggil	argumen bawaan
beberapa konstruktor	konstruktor yang di-*overload*, `this(...)`	satu `__init__` dengan nilai bawaan, atau *factory* `@classmethod`
berapa pun banyaknya argumen	*varargs*, `int... values`	`*args` (dan `**kwargs` untuk argumen bernama)
kode berbeda per tipe argumen	*overloading*, dipilih saat kompilasi	`functools.singledispatch`, dipilih saat program berjalan
kode yang sama untuk banyak tipe	antarmuka (Subbab 6.12)	*protocol* atau ABC (Subbab 6.15)

Baris terakhir itulah yang paling penting. Baik *overloading* maupun `singledispatch` tidak menghapus duplikasi `average`. Untuk itu kita memerlukan cara untuk berkata "objek apa pun yang dapat memberiku sebuah bilangan" — dan itulah antarmuka.

6.12 Antarmuka

Perhatikan sekali lagi kedua metode `average`. Apa yang sebenarnya dibutuhkan perulangannya dari setiap objek? Bukan bahwa objek itu rekening; bukan pula bahwa ia negara. Perulangan itu membutuhkan tepat satu hal: *sebuah bilangan untuk dijumlahkan*. Seandainya kita dapat menulis tipe yang berarti "objek yang dapat memberiku bilangannya", satu `average` akan melayani rekening, negara, dan apa pun yang kita ciptakan tahun depan.

Tipe itu adalah sebuah **antarmuka** (*interface*).

> **DEFINISI.** *Antarmuka* adalah daftar bernama kepala-kepala metode — nama, tipe parameter, dan tipe kembalian — tanpa tubuh. Antarmuka menggambarkan *apa* yang dapat dilakukan sebuah objek dan tidak berkata apa pun tentang *bagaimana*. Sebuah kelas *mengimplementasikan* antarmuka dengan menyatakannya (`implements Measurable` di Java) dan dengan menyediakan tubuh untuk setiap metode yang didaftar antarmuka itu. Antarmuka kemudian dapat dipakai sebagai *tipe*: variabel, parameter, atau larik bertipe itu dapat menampung objek apa pun dari kelas mana pun yang mengimplementasikannya.

Di Java, antarmuka dideklarasikan dengan kata kunci `interface`. **Kode 6.21** adalah `Measurable` selengkapnya.

Kode 6.21 `Measurable.java`: antarmuka dengan satu metode dan tanpa kode.

```
/**
 * Anything that can be reduced to one number for averaging.
 */
public interface Measurable {
    /**
     * Returns the number that represents this object when averaging.
     *
     * @return the measure
     */
    double getMeasure();      // tanpa badan: setiap kelas pengimplementasi wajib
menyediakannya
}
```

Metode itu memiliki kepala dan titik koma di tempat tubuhnya semestinya berada. Setiap metode yang dideklarasikan seperti ini di dalam antarmuka otomatis bersifat `public` dan `abstract` — *abstract* berarti "dideklarasikan di sini, diimplementasikan di tempat lain" — sehingga kedua kata kunci itu biasanya tidak ditulis. Antarmuka biasanya tinggal di berkasnya sendiri, dinamai sesuai namanya, persis seperti kelas publik.

Kode 6.22 membuat dua kelas mengimplementasikannya dan menulis `average` satu kali saja.

Kode 6.22 `MeasurableDemo.java`: dua kelas mengimplementasikan `Measurable`, dan satu metode `average` melayani keduanya.

```java
/** A bank account, measured by its balance. */
class Account implements Measurable {
    private final String id;
    private final String owner;
    private final double balance;

    Account(String id, String owner, double balance) {
        this.id = id;
        this.owner = owner;
        this.balance = balance;
    }

    @Override
    public double getMeasure() {
        return balance;
    }
}

/** A country, measured by its area. */
class Country implements Measurable {
    private final String name;
    private final double area;

    Country(String name, double area) {
        this.name = name;
        this.area = area;
    }

    @Override
    public double getMeasure() {
        return area;
    }
}

/**
 * One average method for every Measurable class.
 */
public class MeasurableDemo {
    /**
     * Computes the average measure of some objects.
     *
     * @param objects anything that implements Measurable
     * @return the average measure, or 0 for an empty array
     */
    public static double average(Measurable[] objects) {
        if (objects.length == 0) {
```

Kode 6.22 (Lanjutan) *MeasurableDemo.java: dua kelas mengimplementasikan Measurable, dan satu metode average melayani keduanya.*

```
            return 0;
        }
        double sum = 0;
        for (Measurable obj : objects) {     // obj bisa berupa Account atau Country
            sum = sum + obj.getMeasure();
        }
        return sum / objects.length;
    }

    /**
     * Averages accounts two ways, then countries.
     *
     * @param args not used
     */
    public static void main(String[] args) {
        // Cara pertama: larik yang tipe elemennya antarmuka
        Measurable[] accounts = new Measurable[3];
        accounts[0] = new Account("01", "Aleksandra", 100);
        accounts[1] = new Account("02", "Natasha", 150);
        accounts[2] = new Account("03", "Sergei", 125);
        System.out.println("Average balance is: " + average(accounts));

        // Cara kedua: larik dari kelas itu sendiri
        Account b1 = new Account("01", "Aleksandra", 100);
        Account b2 = new Account("02", "Natasha", 150);
        Account[] accounts2 = {b1, b2};
        System.out.println("Average balance is: " + average(accounts2));

        Measurable[] countries = {
            new Country("Indonesia", 1_904_569),
            new Country("Malaysia", 330_803),
            new Country("Singapore", 734)
        };
        System.out.println("Average area is: " + average(countries));

        Measurable y = new Account("01", "Aleksandra", 100);  // tipe antarmuka, objek
kelas
        System.out.println("y.getMeasure() = " + y.getMeasure());
    }
}
```

Kompilasi bersama antarmukanya (`javac Measurable.java MeasurableDemo.java`) lalu jalankan `java MeasurableDemo`:

```
Average balance is: 125.0
Average balance is: 125.0
Average area is: 745368.6666666666
y.getMeasure() = 100.0
```

Hasilnya sama dengan **Kode 6.17**, tetapi programnya dibangun dengan cara lain. Mari telusuri bagian-bagian barunya.

`class Account implements Measurable`. Ini sebuah janji, dan kompilator menagihnya: `Account` wajib memuat metode publik `double getMeasure()`, yang mengembalikan saldo; `Country` mengembalikan luasnya. Sebuah kelas boleh mengimplementasikan *beberapa* antarmuka, dipisahkan koma — salah satu keunggulan utama antarmuka.

Bagaimana jika sebuah kelas mengingkari janji? Hapus metode `getMeasure` dari `Country` lalu kompilasi lagi:

```
MeasurableDemo.java:20: error: Country is not abstract and does not override abstract
method getMeasure() in Measurable
class Country implements Measurable {
^
1 error
```

Kesalahan itu tertangkap sebelum program sempat berjalan.

`average(Measurable[] objects)`**.** Tipe parameternya adalah antarmuka, sehingga di dalam perulangan satu-satunya yang boleh kita panggil pada `obj` adalah `getMeasure()` — hanya itu yang dibutuhkan metode ini. Ketika `obj` menunjuk sebuah `Account`, versi milik `Account` yang berjalan; ketika menunjuk sebuah `Country`, versi milik `Country`. Tubuh mana yang berjalan diputuskan *saat program berjalan* oleh kelas objek yang sebenarnya: **pengiriman dinamis** (*dynamic dispatch*), jantung *polimorfisme*, yang dikembangkan Bab 7. Aritmetikanya adalah rata-rata biasa:

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i \tag{6.4}$$

dengan x_i ukuran objek ke-i dan n banyaknya objek. Persamaan (6.4) tidak membutuhkan apa pun selain n bilangan itu, dan justru karena itulah antarmukanya tidak membutuhkan apa pun selain `getMeasure`. Untuk ketiga rekening, $\bar{x} = (100 + 150 + 125)/3 = 375/3 = 125$.

Dua cara memanggilnya. Program itu mengirimkan `Measurable[]` yang berisi rekening dan juga `Account[]` biasa; Java menerima yang kedua karena setiap `Account` *adalah* sebuah `Measurable`.

Tidak ada objek dari antarmuka. Sebuah variabel boleh bertipe antarmuka — `Measurable y` — dan menunjuk objek dari kelas yang mengimplementasikannya, tetapi Anda tidak pernah dapat membuat objek *dari antarmuka itu sendiri*:

```
NewIface.java:4: error: Measurable is abstract; cannot be instantiated
        Measurable x = new Measurable();
                       ^
1 error
```

`new Measurable()` akan menjadi objek yang menjanjikan `getMeasure` tanpa tubuh. Antarmuka adalah kontrak, bukan benda.

> ✓ **WAWASAN KUNCI.** Antarmuka membelah program menjadi dua sisi yang tidak lagi perlu saling mengenal. *Pemanggil* (`average`) hanya mengenal antarmuka. *Pengimplementasi* (`Account`, `Country`) hanya mengenal antarmuka. Sisi mana pun dapat berubah — pengimplementasi baru, pemanggil yang ditulis ulang — tanpa menyentuh sisi lainnya, selama keduanya memegang kontrak.

Versi Python, **Kode 6.23**, memakai sebuah **protocol**, cara Python menulis antarmuka yang diperiksa berdasarkan *bentuk*. Kita kembali ke *protocol* di Subbab 6.15; untuk sementara bacalah kelas `Measurable(Protocol)` sebagai "objek apa pun yang memiliki metode `get_measure()` yang mengembalikan *float*".

Kode 6.23 `measurable_demo.py`: *protocol* `Measurable`; perhatikan bahwa `Account` dan `Country` tidak pernah menyebutnya.

```
"""One average function for everything that has a measure, using a Protocol."""
from collections.abc import Sequence
from typing import Protocol, runtime_checkable


@runtime_checkable
class Measurable(Protocol):
    """Anything that can be reduced to one number for averaging."""

    def get_measure(self) -> float:
        ...


class Account:            # perhatikan: tidak ada "Measurable" di mana pun dalam kelas ini
    def __init__(self, id, owner, balance):
        self.id = id
        self.owner = owner
        self.balance = balance

    def get_measure(self) -> float:
        return self.balance


class Country:
    def __init__(self, name, area):
        self.name = name
        self.area = area

    def get_measure(self) -> float:
        return self.area


def average(objects: Sequence[Measurable]) -> float:
    """Average measure of the objects, or 0 for an empty list."""
    if len(objects) == 0:
        return 0
    total = 0.0
    for obj in objects:
        total = total + obj.get_measure()
    return total / len(objects)
```

Kode 6.23 (Lanjutan) *measurable_demo.py: protocol Measurable; perhatikan bahwa Account dan Country tidak pernah menyebutnya.*

```
if __name__ == "__main__":
    accounts = [Account("01", "Aleksandra", 100),
                Account("02", "Natasha", 150),
                Account("03", "Sergei", 125)]
    print("Average balance is:", average(accounts))
    countries = [Country("Indonesia", 1_904_569),
                 Country("Malaysia", 330_803),
                 Country("Singapore", 734)]
    print("Average area is:", average(countries))
    print("Account is Measurable:", isinstance(accounts[0], Measurable))
    print("a str is Measurable  :", isinstance("hello", Measurable))
```

```
Average balance is: 125.0
Average area is: 745368.6666666666
Account is Measurable: True
a str is Measurable  : False
```

Tanda `...` di metode *protocol* adalah cara Python menulis “tidak ada tubuh di sini”. `Account` dan `Country` tidak menyebut `Measurable` di mana pun, tetapi `isinstance(accounts[0],`

`Measurable)` menjawab `True`: objek itu memiliki metode `get_measure`, jadi ia cocok. String tidak memiliki metode itu, jadi tidak cocok. Anotasi `Sequence[Measurable]` pada `average` berarti "list, *tuple*, atau barisan lain berisi objek `Measurable`"; Python sendiri mengabaikan anotasi saat berjalan, tetapi perkakas membacanya, seperti diperlihatkan Subbab 6.15.

6.12.1 Mengapa antarmuka penting: kisah plug-in

Antarmuka bukan sekadar cara menghemat beberapa baris yang berulang. Bayangkan sebuah perusahaan yang filter penyunting gambarnya adalah rahasia dagang, tetapi ingin orang lain menulis *plug-in* untuk penyuntingnya. Perusahaan itu menerbitkan sebuah antarmuka, misalnya `interface ImageTool { String name(); Image apply(Image input); }`, beserta halaman `javadoc`-nya. Siapa pun boleh menulis kelas yang mengimplementasikan `ImageTool` dan menyerahkan kelas terkompilasinya; penyunting menaruh `name()` di menunya dan memanggil `apply` ketika pengguna mengeklik. Kedua pihak tidak pernah melihat kode sumber pihak lain: antarmuka adalah satu-satunya yang mereka miliki bersama. Ekstensi peramban, *plug-in* penyunting kode, dan *driver* pencetak semuanya mengikuti bentuk ini.

6.13 Menukar Representasi di Balik Antarmuka

Berikut versi yang lebih kecil dari kisah yang sama, yang dapat Anda jalankan. Sebuah program harus menyimpan tabel kunci bilangan bulat — sebuah "basis data" mungil — dengan dua operasi: menambahkan kunci dan mencari kunci. Ada dua cara yang masuk akal untuk menyimpan kunci-kunci itu. `LinearDB` menyimpannya dalam list yang tidak terurut dan mencari dengan memeriksa setiap kunci bergiliran. `TreeDB` menyimpannya dalam *pohon pencarian biner*: setiap kunci duduk di sebuah simpul, kunci yang lebih kecil ke kiri, yang lebih besar ke kanan, sehingga pencarian mengikuti satu lintasan dari puncak alih-alih memeriksa semuanya. Mana yang lebih baik bergantung pada datanya, sehingga program klien ingin memutuskannya ketika mulai berjalan.

6.13.1 Tanpa antarmuka

Kode 6.24 adalah klien yang ditulis tanpa antarmuka. Kedua kelasnya berada di paket `dbrep`, jadi berkasnya terletak di folder `dbrep`; kompilasi dari folder *di atasnya* dengan `javac dbrep/DBClient.java` dan jalankan dengan `java dbrep.DBClient`.

Kode 6.24 `dbrep/DBClient.java`: klien yang menyimpan satu variabel per representasi dan menguji sebuah penanda (*flag*) di setiap operasi.

```java
package dbrep;

import java.util.ArrayList;
import java.util.List;

/** Keeps the keys in an unsorted list; a search looks at every key. */
class LinearDB {
    private final List<Integer> keys = new ArrayList<>();

    void addKey(int k) {
        System.out.println("LinearDB: adding key " + k);
        keys.add(k);
    }

    boolean search(int k) {
        System.out.println("LinearDB: searching for key " + k);
        for (int key : keys) {
            if (key == k) {
                return true;
            }
        }
        return false;
    }
}

/** Keeps the keys in a binary search tree; a search follows one path down. */
class TreeDB {
    private static class Node {
        final int key;
        Node left;
        Node right;

        Node(int key) {
            this.key = key;
        }
    }

    private Node root;
```

Kode 6.24 (Lanjutan) *dbrep/DBClient.java: klien yang menyimpan satu variabel per representasi dan menguji sebuah penanda (flag) di setiap operasi.*

```
    void addKey(int k) {
        System.out.println("TreeDB: adding key " + k);
        Node fresh = new Node(k);
        if (root == null) {
            root = fresh;
            return;
        }
        Node node = root;
        while (true) {
            if (k == node.key) {
                return;                          // sudah ada
            } else if (k < node.key) {
                if (node.left == null) {
                    node.left = fresh;
                    return;
                }
                node = node.left;
            } else {
                if (node.right == null) {
                    node.right = fresh;
                    return;
                }
                node = node.right;
            }
        }
    }

    boolean search(int k) {
        System.out.println("TreeDB: searching for key " + k);
        Node node = root;
        while (node != null) {
            if (k == node.key) {
                return true;
            }
            node = (k < node.key) ? node.left : node.right;
        }
        return false;
    }
}
```

Kode 6.24 (Lanjutan) *dbrep/DBClient.java: klien yang menyimpan satu variabel per representasi dan menguji sebuah penanda (flag) di setiap operasi.*

```
/**
 * A client that must test, at every operation, which representation it chose.
 */
public class DBClient {
    static LinearDB ldb;
    static TreeDB tdb;
    static boolean useLinearDB = true;

    public static void main(String[] args) {
        // Tentukan representasi yang dipakai (bawaannya pohon)
        String decision = (args.length > 0) ? args[0] : "Tree Rep";
        if (decision.equals("Linear Rep")) {
            ldb = new LinearDB();
        } else {                                    // selain itu, pohon yang dipakai
            useLinearDB = false;
            tdb = new TreeDB();
        }
        // Tambahkan tiga kunci
        int[] keys = {13, 9, 21};
        for (int k : keys) {
            if (useLinearDB) {
                ldb.addKey(k);
            } else {
                tdb.addKey(k);
            }
        }
        // Cari dua kunci
        int k2 = 9;
        boolean result = useLinearDB ? ldb.search(k2) : tdb.search(k2);
        System.out.println("found " + k2 + ": " + result);
        int k3 = 4;
        result = useLinearDB ? ldb.search(k3) : tdb.search(k3);
        System.out.println("found " + k3 + ": " + result);
    }
}
```

Secara bawaan ia memilih pohon:

```
TreeDB: adding key 13
TreeDB: adding key 9
TreeDB: adding key 21
TreeDB: searching for key 9
found 9: true
TreeDB: searching for key 4
found 4: false
```

dan dengan argumen `"Linear Rep"` (`java dbrep.DBClient "Linear Rep"`) ia memilih list:

```
LinearDB: adding key 13
LinearDB: adding key 9
LinearDB: adding key 21
LinearDB: searching for key 9
found 9: true
LinearDB: searching for key 4
found 4: false
```

(Cara kerja `TreeDB` di dalamnya adalah pokok bahasan mata kuliah struktur data; yang penting di sini hanyalah bahwa ia menawarkan dua operasi yang sama dengan `LinearDB`.)

Program itu bekerja, tetapi sudah tidak menyenangkan. `main` menyimpan *dua* variabel, `ldb` dan `tdb`, yang salah satunya selalu `null`, dan sebuah penanda, `useLinearDB`, yang menyatakan mana yang sungguhan. Setiap operasi basis data dibungkus dengan pengujian penanda itu. Basis data

sungguhan memiliki dua puluh operasi, dan pada hari seseorang menambahkan representasi ketiga, setiap tempat itu harus menumbuhkan cabang ketiga — dan salah satunya akan terlupa. **Kode 6.25** adalah rancangan yang sama dalam Python, dengan bau yang sama.

Kode 6.25 `db_client_v1.py`: klien penguji penanda yang sama dalam Python.

```
"""A client that must test, at every operation, which representation it chose."""
import sys


class LinearDB:
    """Keeps the keys in an unsorted list; a search looks at every key."""

    def __init__(self):
        self._keys = []

    def add_key(self, k):
        print("LinearDB: adding key", k)
        self._keys.append(k)

    def search(self, k):
        print("LinearDB: searching for key", k)
        for key in self._keys:
            if key == k:
                return True
        return False


class _Node:
    def __init__(self, key):
        self.key = key
        self.left = None
        self.right = None


class TreeDB:
    """Keeps the keys in a binary search tree; a search follows one path down."""

    def __init__(self):
        self._root = None

    def add_key(self, k):
        print("TreeDB: adding key", k)
        fresh = _Node(k)
        if self._root is None:
            self._root = fresh
            return
        node = self._root
        while True:
            if k == node.key:
                return                          # sudah ada
            elif k < node.key:
```

Kode 6.25 (Lanjutan) *db_client_v1.py: klien penguji penanda yang sama dalam Python.*

```
                if node.left is None:
                    node.left = fresh
                    return
                node = node.left
            else:
                if node.right is None:
                    node.right = fresh
                    return
                node = node.right

    def search(self, k):
        print("TreeDB: searching for key", k)
        node = self._root
        while node is not None:
            if k == node.key:
                return True
            node = node.left if k < node.key else node.right
        return False


ldb = None
tdb = None
use_linear_db = True

# Tentukan representasi yang dipakai (bawaannya pohon)
decision = sys.argv[1] if len(sys.argv) > 1 else "Tree Rep"
if decision == "Linear Rep":
    ldb = LinearDB()
else:                                          # selain itu, pohon yang dipakai
    use_linear_db = False
    tdb = TreeDB()
# Tambahkan tiga kunci
for k in [13, 9, 21]:
    if use_linear_db:
        ldb.add_key(k)
    else:
        tdb.add_key(k)
# Cari dua kunci
k2 = 9
result = ldb.search(k2) if use_linear_db else tdb.search(k2)
print(f"found {k2}: {str(result).lower()}")
k3 = 4
result = ldb.search(k3) if use_linear_db else tdb.search(k3)
print(f"found {k3}: {str(result).lower()}")
```

```
TreeDB: adding key 13
TreeDB: adding key 9
TreeDB: adding key 21
TreeDB: searching for key 9
found 9: true
TreeDB: searching for key 4
found 4: false
```

6.13.2 Dengan antarmuka

Operasi yang ditawarkan kedua representasi adalah `addKey` dan `search`, maka kita menuliskannya sebagai antarmuka, `DBops`, dan membuat kedua kelas mengimplementasikannya: tambahkan `implements DBops` pada setiap kepala kelas dan deklarasikan kedua metode itu `public` (sebuah kelas tidak boleh mengimplementasikan metode antarmuka dengan visibilitas yang lebih sempit). Tidak ada hal lain yang berubah pada kedua kelas.

Kliennya berubah banyak. **Kode 6.26** adalah hasilnya.

Kode 6.26 `dbiface/DBClient.java`: satu variabel bertipe antarmuka, tanpa penanda, dan tanpa pengujian penanda.

```
package dbiface;

import java.util.ArrayList;
import java.util.List;

/** The operations every representation of the table must offer. */
interface DBops {
    void addKey(int k);

    boolean search(int k);

    /** Adds several keys; written once here, inherited by every implementation. */
    default void addSeveral(int[] keys) {
        for (int k : keys) {
            addKey(k);
        }
    }

    /** Tells whether at least one of the keys is present. */
    default boolean findOneOf(int[] keys) {
        for (int k : keys) {
            if (search(k)) {
                return true;
            }
        }
        return false;
    }
}

/** Keeps the keys in an unsorted list; a search looks at every key. */
class LinearDB implements DBops {
    private final List<Integer> keys = new ArrayList<>();

    @Override
    public void addKey(int k) {
        System.out.println("LinearDB: adding key " + k);
        keys.add(k);
    }
```

Kode 6.26 (Lanjutan) *dbiface/DBClient.java: satu variabel bertipe antarmuka, tanpa penanda, dan tanpa pengujian penanda.*

```
    @Override
    public boolean search(int k) {
        System.out.println("LinearDB: searching for key " + k);
        for (int key : keys) {
            if (key == k) {
                return true;
            }
        }
        return false;
    }
}

/** Keeps the keys in a binary search tree; a search follows one path down. */
class TreeDB implements DBops {
    private static class Node {
        final int key;
        Node left;
        Node right;

        Node(int key) {
            this.key = key;
        }
    }

    private Node root;

    @Override
    public void addKey(int k) {
        System.out.println("TreeDB: adding key " + k);
        Node fresh = new Node(k);
        if (root == null) {
            root = fresh;
            return;
        }
        Node node = root;
        while (true) {
            if (k == node.key) {
                return;                             // sudah ada
            } else if (k < node.key) {
                if (node.left == null) {
                    node.left = fresh;
                    return;
                }
                node = node.left;
            } else {
                if (node.right == null) {
```

Kode 6.26 (Lanjutan) *dbiface/DBClient.java: satu variabel bertipe antarmuka, tanpa penanda, dan tanpa pengujian penanda.*

```
                    node.right = fresh;
                    return;
                }
                node = node.right;
            }
        }
    }

    @Override
    public boolean search(int k) {
        System.out.println("TreeDB: searching for key " + k);
        Node node = root;
        while (node != null) {
            if (k == node.key) {
                return true;
            }
            node = (k < node.key) ? node.left : node.right;
        }
        return false;
    }
}

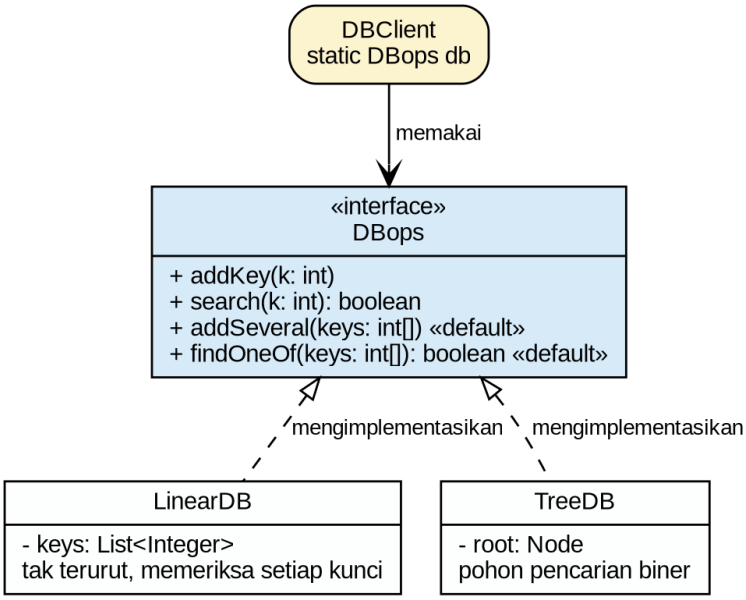
/**
 * The same client, written against the interface: no tests, no second variable.
 */
public class DBClient {
    static DBops db;

    public static void main(String[] args) {
        // Tentukan representasi yang dipakai (bawaannya pohon)
        String decision = (args.length > 0) ? args[0] : "Tree Rep";
        if (decision.equals("Linear Rep")) {
            db = new LinearDB();
        } else {
            db = new TreeDB();
        }
        db.addSeveral(new int[] {13, 9, 21});
        System.out.println("found 9: " + db.search(9));
        System.out.println("found 4: " + db.search(4));
        System.out.println("found one of {4, 21}: " + db.findOneOf(new int[] {4, 21}));
    }
}
```

```
TreeDB: adding key 13
TreeDB: adding key 9
TreeDB: adding key 21
TreeDB: searching for key 9
found 9: true
TreeDB: searching for key 4
found 4: false
TreeDB: searching for key 4
TreeDB: searching for key 21
found one of {4, 21}: true
```

Kini `main` memiliki *satu* variabel, `db`, bertipe antarmuka `DBops`. Keputusan antara `LinearDB` dan `TreeDB` diambil sekali, di bagian atas, dan tidak pernah disebut lagi; setiap operasi adalah pemanggilan biasa pada `db`, dan pengiriman dinamis menjalankan kode yang tepat. Representasi ketiga cukup berupa satu kelas baru dan satu baris tambahan dalam keputusan itu.

Gambar 6.5 menggambar struktur barunya dalam UML, bahasa diagram baku untuk kelas. Kotak dengan «interface» di atas namanya adalah antarmuka; garis putus-putus dengan segitiga kosong berarti “mengimplementasikan”; panah biasa berarti “memakai”.



Gambar 6.5 Klien hanya bergantung pada antarmuka `DBops`; `LinearDB` dan `TreeDB` mengimplementasikannya dan dapat dipertukarkan tanpa mengubah klien.

6.13.3 Hierarki tipe: subtipe dan supertipe

Setelah `LinearDB` dan `TreeDB` mengimplementasikan `DBops`, ketiga tipe membentuk sebuah *hierarki* kecil: `DBops` adalah **supertipe** dari setiap kelas, dan setiap kelas adalah **subtipe** dari `DBops`. Setiap objek subtipe dapat dipandang sebagai objek supertipe. Dalam penugasan seperti `db = new TreeDB();`, atau ketika sebuah `Account[]` dikirim ke tempat yang mengharapkan `Measurable[]`, Java mengubah dari subtipe ke supertipe *secara otomatis*: tidak ada yang bisa salah, karena `TreeDB` menawarkan semua yang dijanjikan `DBops`. Subbab 6.14.1 melihat arah sebaliknya, yang tidak otomatis.

6.13.4 Berbagi kode: metode default

Sekarang andaikan kedua representasi membutuhkan dua operasi lagi: `addSeveral(int[] keys)`, yang menambahkan satu larik kunci sekaligus, dan `findOneOf(int[] keys)`, yang menyatakan apakah setidaknya satu dari beberapa kunci ada. Keduanya mudah ditulis dengan `addKey` dan `search` — satu perulangan atas larik — dan perulangan itu akan sama persis di `LinearDB` dan `TreeDB`. Haruskah kita menyalinnya ke kedua kelas?

Sejak Java 8, antarmuka boleh memuat **metode default**: metode bertubuh, bertanda `default`, yang diwarisi setiap kelas pengimplementasi kecuali kelas itu menyediakan versinya sendiri. Di **Kode 6.26**, `addSeveral` dan `findOneOf` adalah metode default, ditulis sekali dari kedua metode abstrak dan diwarisi oleh kedua kelas; baris-baris terakhir keluarannya memperlihatkan `findOneOf` memanggil `search` untuk `4` lalu `21`. Antarmuka juga boleh memuat metode `static`, yang menjadi milik antarmuka itu sendiri; Subbab 6.16 memakai satu.

Metode default hanya dapat memakai metode milik antarmuka itu sendiri, karena antarmuka tidak memiliki *field*: ia menggambarkan perilaku, bukan keadaan. Ketika implementasi-implementasi juga harus berbagi *field*, kelas abstrak adalah perkakasnya (Bab 7); **Tabel 6.6** di bawah menimbang keduanya.

Versi Python memakai sebuah **kelas dasar abstrak** (*abstract base class*), yang dapat memuat kode bersama secara langsung. **Kode 6.27** memperlihatkannya.

Kode 6.27 `db_client_v2.py`: antarmuka sebagai kelas dasar abstrak, dengan dua metode bersama.

```
"""The same client, written against an abstract base class."""
import sys
from abc import ABC, abstractmethod


class DBops(ABC):
    """The operations every representation of the table must offer."""

    @abstractmethod
    def add_key(self, k):
        ...

    @abstractmethod
    def search(self, k):
        ...

    def add_several(self, keys):
        """Add several keys; written once here, inherited by every subclass."""
        for k in keys:
            self.add_key(k)

    def find_one_of(self, keys):
        """Tell whether at least one of the keys is present."""
        for k in keys:
            if self.search(k):
                return True
        return False


class LinearDB(DBops):
    """Keeps the keys in an unsorted list; a search looks at every key."""

    def __init__(self):
        self._keys = []

    def add_key(self, k):
        print("LinearDB: adding key", k)
        self._keys.append(k)

    def search(self, k):
        print("LinearDB: searching for key", k)
        for key in self._keys:
            if key == k:
                return True
        return False
```

Kode 6.27 (Lanjutan) *db_client_v2.py: antarmuka sebagai kelas dasar abstrak, dengan dua metode bersama.*

```
class _Node:
    def __init__(self, key):
        self.key = key
        self.left = None
        self.right = None


class TreeDB(DBops):
    """Keeps the keys in a binary search tree; a search follows one path down."""

    def __init__(self):
        self._root = None

    def add_key(self, k):
        print("TreeDB: adding key", k)
        fresh = _Node(k)
        if self._root is None:
            self._root = fresh
            return
        node = self._root
        while True:
            if k == node.key:
                return                          # sudah ada
            elif k < node.key:
                if node.left is None:
                    node.left = fresh
                    return
                node = node.left
            else:
                if node.right is None:
                    node.right = fresh
                    return
                node = node.right

    def search(self, k):
        print("TreeDB: searching for key", k)
        node = self._root
        while node is not None:
            if k == node.key:
                return True
            node = node.left if k < node.key else node.right
        return False
```

Kode 6.27 (Lanjutan) *db_client_v2.py: antarmuka sebagai kelas dasar abstrak, dengan dua metode bersama.*

```
# Tentukan representasi yang dipakai (bawaannya pohon)
decision = sys.argv[1] if len(sys.argv) > 1 else "Tree Rep"
db: DBops = LinearDB() if decision == "Linear Rep" else TreeDB()
db.add_several([13, 9, 21])
print("found 9:", str(db.search(9)).lower())
print("found 4:", str(db.search(4)).lower())
print("found one of {4, 21}:", str(db.find_one_of([4, 21])).lower())
try:
    DBops()
except TypeError as e:
    print("DBops() fails:", e)
```

```
TreeDB: adding key 13
TreeDB: adding key 9
TreeDB: adding key 21
TreeDB: searching for key 9
found 9: true
TreeDB: searching for key 4
found 4: false
TreeDB: searching for key 4
TreeDB: searching for key 21
found one of {4, 21}: true
DBops() fails: Can't instantiate abstract class DBops with abstract methods add_key, search
```

`DBops` mewarisi `ABC`, dan kedua operasinya memakai dekorator `@abstractmethod`. Metode bersama `add_several` dan `find_one_of` adalah metode biasa yang ditulis di kelas dasar; `LinearDB(DBops)` dan `TreeDB(DBops)` mewarisinya. Baris terakhir membuktikan bahwa kelas abstrak tidak dapat dibuat objeknya: `DBops()` melempar `TypeError` yang menyebutkan metode-metode yang belum ada, sama seperti Java menolak `new Measurable()`. (Python 3.12 dan sesudahnya menyusun pesannya "without an implementation for abstract methods".)

6.14 Antarmuka yang Sudah Anda Pakai

Pustaka Java penuh dengan antarmuka, dan Anda sudah memakai sebagian di antaranya tanpa menyadarinya. **Tabel 6.5** mendaftar antarmuka yang pertama kali ditemui pemula, beserta padanannya di Python. Paket `java.lang` (dimuat otomatis) dan `java.util` (koleksi) memuat sebagian besarnya; daftar lengkapnya ada di dokumentasi API setiap paket.

Tabel 6.5 Antarmuka dari pustaka Java yang muncul paling awal, dan apa yang memainkan peran yang sama di Python.

Antarmuka	Paket	Menjanjikan	Padanan Python
`Comparable<T>`	`java.lang`	urutan alami: `compareTo(T other)`	`__lt__` (dan `functools.total_ordering`)
`Comparator<T>`	`java.util`	urutan yang dipilih dari luar: `compare(a, b)`	fungsi `key=` untuk `sorted`
`Iterable<T>`	`java.lang`	dapat dipakai dalam perulangan *for-each*: `iterator()`	`__iter__`, `collections.abc.Iterable`
`CharSequence`	`java.lang`	barisan `char` yang dapat dibaca	`str`, `collections.abc.Sequence`
`AutoCloseable`	`java.lang`	melepaskan sumber daya: `close()`, untuk *try-with-resources*	`__enter__`/`__exit__`, pernyataan `with`
`Runnable`	`java.lang`	tugas yang dapat dijalankan sebuah *thread*: `run()`	*callable* apa pun, misalnya fungsi
`Collection<E>`, `List<E>`, `Set<E>`, `Map<K,V>`	`java.util`	operasi-operasi koleksi	`collections.abc.Collection`, `Sequence`, `Set`, `Mapping`
`ActionListener`	`java.awt.event`	bereaksi pada tombol: `actionPerformed(e)`	fungsi *callback* (`command=` di Tkinter)

Anda menulis `List<Order> orders = new ArrayList<>();` di Bab 5 — `List` adalah antarmuka dan `ArrayList` salah satu implementasinya, itulah sebabnya kode yang sama bekerja dengan `LinkedList`. Ketika perulangan *for-each* berjalan atas sebuah list, ia mengandalkan `Iterable`. Ketika *try-with-resources* menutup berkas Anda di Bab 5, ia mengandalkan `AutoCloseable`. Sisa

subbab ini melihat dua antarmuka lebih dekat: `Comparable`, yang menjadi sandaran pengurutan, dan `ActionListener`, yang menjadi sandaran setiap tombol.

6.14.1 Comparable, dan cast menuruni hierarki

Sebagian besar koleksi dapat diurutkan, dan pengurutan adalah rangkaian panjang perbandingan: "apakah elemen ini harus datang sebelum elemen itu?" Agar `Arrays.sort` dapat mengurutkan objek dari kelas *Anda*, ia harus dapat mengajukan pertanyaan itu kepada objek-objek tersebut, dan ia bertanya melalui antarmuka `Comparable`. Metode tunggalnya, `int compareTo(T other)`, menjawab dengan tanda sebuah bilangan bulat:

$$\operatorname{sign}(\texttt{a.compareTo(b)}) = \begin{cases} -1 & \text{jika } a \text{ datang sebelum } b \\ 0 & \text{jika urutan keduanya setara} \\ +1 & \text{jika } a \text{ datang sesudah } b \end{cases} \tag{6.5}$$

Hanya *tanda*-nya yang dijanjikan — Persamaan (6.5) berbicara tentang tanda, bukan nilai. `"Ace".compareTo("Zonk")` bernilai -25 (selisih kode `'A'` dan `'Z'`), bukan -1. Ingatlah hal ini; seluruh kisah di bawah bertumpu padanya.

Andaikan setiap nama memiliki nama belakang dan nama depan, dan nama-nama harus diurutkan menurut nama belakang, lalu menurut nama depan. **Kode 6.28** memuat dua versi kelas itu berdampingan. `SwitchName` ditulis dengan gaya lama: ia mengimplementasikan `Comparable` *mentah* (*raw*) — tanpa `<T>` — yang metodenya menerima `Object` biasa, dan ia memutuskan dengan `switch` atas hasil perbandingan nama belakang. `Name` adalah versi modern yang benar.

Kode 6.28 `NameSortDemo.java`: `compareTo` yang benar, dan satu yang percaya bahwa `String.compareTo` hanya mengembalikan -1, 0, atau 1.

```java
import java.util.Arrays;

/**
 * A name compared with a switch that expects compareTo to return only -1, 0 or 1.
 * It uses the raw (pre-generics) Comparable, so it must cast its argument.
 */
@SuppressWarnings("rawtypes")
class SwitchName implements Comparable {
    private final String lastName;
    private final String firstName;

    SwitchName(String lastName, String firstName) {
        this.lastName = lastName;
        this.firstName = firstName;
    }

    @Override
    public int compareTo(Object other) {
        SwitchName o = (SwitchName) other;           // down-cast: Object -> SwitchName
        int result = Integer.MIN_VALUE;
        switch (lastName.compareTo(o.lastName)) {
            case -1:
                result = -1;
                break;
            case 1:
                result = 1;
                break;
            case 0:
                result = firstName.compareTo(o.firstName);
                break;
        }
        return result;                               // MIN_VALUE untuk setiap selisih
lainnya!
    }

    @Override
    public String toString() {
        return "[" + lastName + ", " + firstName + "]";
    }
}
```

Kode 6.28 (Lanjutan) *NameSortDemo.java: compareTo yang benar, dan satu yang percaya bahwa String.compareTo hanya mengembalikan -1, 0, atau 1.*

```
/**
 * A name ordered by last name, then by first name: the correct version.
 */
class Name implements Comparable<Name> {
    private final String lastName;
    private final String firstName;

    Name(String lastName, String firstName) {
        this.lastName = lastName;
        this.firstName = firstName;
    }

    @Override
    public int compareTo(Name other) {               // tidak perlu cast: other sudah
bertipe Name
        int byLast = lastName.compareTo(other.lastName);
        if (byLast != 0) {
            return byLast;                           // bilangan negatif atau positif mana
pun boleh
        }
        return firstName.compareTo(other.firstName);
    }

    @Override
    public String toString() {
        return "[" + lastName + ", " + firstName + "]";
    }
}

/**
 * Sorts the same names with both versions.
 */
public class NameSortDemo {
    public static void main(String[] args) {
        Name[] names = {
            new Name("Zonk", "Godot"),
            new Name("Bogdanov", "Alexei"),
            new Name("Ace", "Portgas D."),
            new Name("Bogdanov", "Aleksandra")
        };
        System.out.println("Original order:");
        for (Name n : names) {
            System.out.println(n);
        }
        Arrays.sort(names);                          // memakai Name.compareTo
        System.out.println("\nAfter sorting:");
        for (Name n : names) {
            System.out.println(n);
```

Kode 6.28 (Lanjutan) *NameSortDemo.java: compareTo yang benar, dan satu yang percaya bahwa String.compareTo hanya mengembalikan -1, 0, atau 1.*

```
        }

        SwitchName[] risky = {
            new SwitchName("Bogdanov", "Alexei"),
            new SwitchName("Zonk", "Godot"),
            new SwitchName("Ace", "Portgas D."),
            new SwitchName("Bogdanov", "Aleksandra")
        };
        Arrays.sort(risky);                          // memakai SwitchName.compareTo
        System.out.println("\nSwitch version after sorting:");
        for (SwitchName n : risky) {
            System.out.println(n);
        }
    }
}
```

```
Original order:
[Zonk, Godot]
[Bogdanov, Alexei]
[Ace, Portgas D.]
[Bogdanov, Aleksandra]

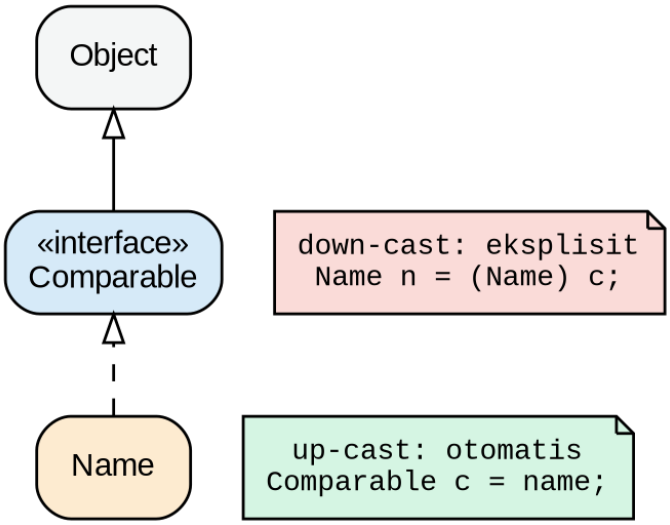
After sorting:
[Ace, Portgas D.]
[Bogdanov, Aleksandra]
[Bogdanov, Alexei]
[Zonk, Godot]

Switch version after sorting:
[Ace, Portgas D.]
[Bogdanov, Aleksandra]
[Zonk, Godot]
[Bogdanov, Alexei]
```

`Name` mengurutkan dengan sempurna, tetapi `SwitchName` menaruh Zonk *di antara* kedua Bogdanov. `switch`-nya hanya memiliki kasus untuk −1, 1, dan 0. Membandingkan "Bogdanov" dengan "Zonk" menghasilkan −24, yang tidak cocok dengan kasus mana pun, sehingga `result` tetap bernilai awal `Integer.MIN_VALUE`, yang berarti "datang sebelum" — benar karena kebetulan. Membandingkan "Zonk" dengan "Bogdanov" menghasilkan +24, *juga* tanpa kasus, *juga* `MIN_VALUE`. Kini metode itu mengklaim bahwa masing-masing nama datang sebelum yang lain, dan urutan hasil pengurutan bergantung pada urutan awalnya: dengan tiga nama *bug*-nya bersembunyi, dengan empat nama ia muncul — itulah sebabnya Subbab 6.8 bersikeras agar masukan uji dipilih dengan sengaja.

`Name` menghindari jebakan itu dengan tidak pernah melihat *nilai* `compareTo`, hanya apakah nilainya nol: jika nama belakang berbeda, perbandingan keduanya adalah jawabannya, berapa pun bilangannya; jika tidak, nama depan yang memutuskan.

Versi mentah juga memperlihatkan sebuah **cast**. `compareTo`-nya menerima `Object`, dan harus menulis `SwitchName o = (SwitchName) other;` sebelum dapat membaca `o.lastName`. **Gambar 6.6** memperlihatkan mengapa *cast* itu diperlukan.



Gambar 6.6 Bergerak naik dalam hierarki tipe (dari `Name` ke `Comparable` atau `Object`) terjadi otomatis; bergerak turun memerlukan *cast* eksplisit, yang diperiksa ketika program berjalan.

Mengubah dari subtipe ke supertipe — ***up-casting*** — terjadi otomatis. Mengubah ke bawah — ***down-casting*** — memerlukan *cast* eksplisit, `(Name) c`, karena mungkin salah: variabel `Comparable` bisa saja memuat sebuah `String`. Mesin virtual memeriksa *cast* itu saat program berjalan dan melempar `ClassCastException` jika gagal. `Comparable<Name>` yang generik menghapus *cast* itu sama sekali, karena `compareTo`-nya menerima sebuah `Name`; utamakan selalu bentuk generik (Bab 9 menjelaskan generik).

Python mengurutkan dengan bertanya `a < b`, yang memanggil `a.__lt__(b)`. **Kode 6.29** memberi `Name` sebuah `__lt__` yang membandingkan *tuple* `(last_name, first_name)`, karena Python membandingkan *tuple* butir demi butir; `@functools.total_ordering` melengkapi `<=`, `>`, dan `>=`. Berkas itu juga memindahkan versi `switch` melalui `functools.cmp_to_key`, dan memperlihatkan cara yang paling khas Python: fungsi `key=` yang menyatakan *apa* yang menjadi dasar pengurutan.

Kode 6.29 `name_sort.py`: pengurutan dengan kunci *tuple*, *bug* `switch` yang sama, dan pengurutan dengan fungsi `key`.

```
"""Ordering names: the Pythonic way, and a port of the switch-based compareTo."""
import sys
from functools import cmp_to_key, total_ordering


def java_compare(a, b):
    """Compare two strings the way Java's String.compareTo does."""
    for x, y in zip(a, b):
        if x != y:
            return ord(x) - ord(y)          # selisih karakter pertama yang tidak sama
    return len(a) - len(b)


@total_ordering                             # membangun <=, >, >= dari __eq__ dan __lt__
class Name:
    """A name ordered by last name, then by first name."""

    def __init__(self, last_name, first_name):
        self.last_name = last_name
        self.first_name = first_name

    def _key(self):
        return (self.last_name, self.first_name)

    def __eq__(self, other):
        if not isinstance(other, Name):
            return NotImplemented
        return self._key() == other._key()

    def __lt__(self, other):
        if not isinstance(other, Name):
            return NotImplemented
        return self._key() < other._key()   # tuple dibandingkan butir demi butir

    def __hash__(self):
        return hash(self._key())

    def __repr__(self):
        return f"[{self.last_name}, {self.first_name}]"
```

Kode 6.29 (Lanjutan) *name_sort.py: pengurutan dengan kunci tuple, bug switch yang sama, dan pengurutan dengan fungsi key.*

```
def switch_compare(a, b):
    """The switch version: correct only when the comparison gives -1, 0 or 1."""
    result = -sys.maxsize - 1
    match java_compare(a.last_name, b.last_name):
        case -1:
            result = -1
        case 1:
            result = 1
        case 0:
            result = java_compare(a.first_name, b.first_name)
    return result                           # bilangan negatif raksasa untuk setiap selisih
lainnya!


names = [Name("Zonk", "Godot"), Name("Bogdanov", "Alexei"),
         Name("Ace", "Portgas D."), Name("Bogdanov", "Aleksandra")]
print("Original order:")
for n in names:
    print(n)
print("\nAfter sorting:")
for n in sorted(names):                     # memakai Name.__lt__
    print(n)

risky = [Name("Bogdanov", "Alexei"), Name("Zonk", "Godot"),
         Name("Ace", "Portgas D."), Name("Bogdanov", "Aleksandra")]
print("\nSwitch version after sorting:")
for n in sorted(risky, key=cmp_to_key(switch_compare)):
    print(n)

print("\nBy first name, using a key function:")
for n in sorted(names, key=lambda n: n.first_name):
    print(n)
```

```
Original order:
[Zonk, Godot]
[Bogdanov, Alexei]
[Ace, Portgas D.]
[Bogdanov, Aleksandra]

After sorting:
[Ace, Portgas D.]
[Bogdanov, Aleksandra]
[Bogdanov, Alexei]
[Zonk, Godot]

Switch version after sorting:
[Ace, Portgas D.]
[Bogdanov, Aleksandra]
[Zonk, Godot]
[Bogdanov, Alexei]

By first name, using a key function:
[Bogdanov, Aleksandra]
[Bogdanov, Alexei]
[Zonk, Godot]
[Ace, Portgas D.]
```

Versi `switch` yang dipindahkan gagal dengan cara yang persis sama, yang menunjukkan bahwa *bug* itu tinggal di dalam gagasannya, bukan di dalam bahasanya.

6.14.2 Pendengar: gagasan di balik ActionListener

Tombol jendela harus memberi tahu objek lain ketika diklik, padahal orang yang menulis kelas `JButton` tidak mungkin tahu objek *Anda* akan seperti apa. Jawaban Java adalah sebuah antarmuka di paket `java.awt.event`:

```
interface ActionListener {
    void actionPerformed(ActionEvent e);
}
```

Sebuah tombol menyimpan daftar pendengar (*listener*), menawarkan `addActionListener` dan `removeActionListener`, dan ketika diklik memanggil `actionPerformed` pada setiap pendengar di daftarnya. Bola yang memantul, bandul yang berayun, dan spiral yang berputar semuanya dapat mendengarkan satu tombol *Stop*, yang tidak perlu tahu apa pun tentang mereka.

Bab 10 membangun jendela sungguhan. **Kode 6.30** memperlihatkan gagasan yang sama tanpa jendela, sehingga dapat Anda jalankan di mana saja.

Kode 6.30 `StopButtonDemo.java`: tombol yang memberi tahu setiap pendengar terdaftar melalui sebuah antarmuka.

```java
import java.util.ArrayList;
import java.util.List;

/** Anything that wants to hear when the Stop button is pressed. */
interface StopListener {
    void stopRequested(String reason);
}

/** A bouncing ball in an animation. */
class BouncingBall implements StopListener {
    @Override
    public void stopRequested(String reason) {
        System.out.println("  ball: stops bouncing (" + reason + ")");
    }
}

/** A swinging pendulum. */
class Pendulum implements StopListener {
    @Override
    public void stopRequested(String reason) {
        System.out.println("  pendulum: comes to rest (" + reason + ")");
    }
}

/** A rotating spiral. */
class RotatingSpiral implements StopListener {
    @Override
    public void stopRequested(String reason) {
        System.out.println("  spiral: stops rotating (" + reason + ")");
    }
}

/** A button that knows only the interface of the objects it notifies. */
class StopButton {
    private final List<StopListener> listeners = new ArrayList<>();

    void addStopListener(StopListener listener) {
        listeners.add(listener);
    }

    void removeStopListener(StopListener listener) {
        listeners.remove(listener);
    }
```

Kode 6.30 (Lanjutan) *StopButtonDemo.java: tombol yang memberi tahu setiap pendengar terdaftar melalui sebuah antarmuka.*

```
    void click(String reason) {
        System.out.println("Stop pressed, " + listeners.size() + " listener(s):");
        for (StopListener listener : listeners) {
            listener.stopRequested(reason);
        }
    }
}

/**
 * The listener idea behind ActionListener, without a window.
 */
public class StopButtonDemo {
    public static void main(String[] args) {
        StopButton stop = new StopButton();
        BouncingBall ball = new BouncingBall();
        stop.addStopListener(ball);
        stop.addStopListener(new Pendulum());
        stop.addStopListener(new RotatingSpiral());
        stop.click("first click");
        stop.removeStopListener(ball);
        stop.click("second click");
    }
}
```

```
Stop pressed, 3 listener(s):
  ball: stops bouncing (first click)
  pendulum: comes to rest (first click)
  spiral: stops rotating (first click)
Stop pressed, 2 listener(s):
  pendulum: comes to rest (second click)
  spiral: stops rotating (second click)
```

`StopButton` memuat kata `StopListener` dan tidak ada hal lain tentang pendengarnya; ketiga objek beranimasi memuat `implements StopListener` dan tidak ada hal lain tentang tombolnya. Bola yang sudah dilepas tidak lagi mendengar klik kedua. **Kode 6.31** adalah versi Python; di Python, fungsi biasa pun dapat menjadi pendengar, dan itulah yang dilakukan `command=` di Tkinter.

Kode 6.31 `stop_button_demo.py`: pola pendengar yang sama dengan sebuah *protocol* Python.

```python
"""The listener idea behind ActionListener, without a window."""
from typing import Protocol


class StopListener(Protocol):
    """Anything that wants to hear when the Stop button is pressed."""

    def stop_requested(self, reason: str) -> None:
        ...


class BouncingBall:
    def stop_requested(self, reason):
        print(f"  ball: stops bouncing ({reason})")


class Pendulum:
    def stop_requested(self, reason):
        print(f"  pendulum: comes to rest ({reason})")


class RotatingSpiral:
    def stop_requested(self, reason):
        print(f"  spiral: stops rotating ({reason})")


class StopButton:
    """A button that knows only the protocol of the objects it notifies."""

    def __init__(self):
        self._listeners: list[StopListener] = []

    def add_stop_listener(self, listener: StopListener):
        self._listeners.append(listener)

    def remove_stop_listener(self, listener: StopListener):
        self._listeners.remove(listener)

    def click(self, reason):
        print(f"Stop pressed, {len(self._listeners)} listener(s):")
        for listener in self._listeners:
            listener.stop_requested(reason)
```

Kode 6.31 (Lanjutan) *stop_button_demo.py: pola pendengar yang sama dengan sebuah protocol Python.*

```python
stop = StopButton()
ball = BouncingBall()
stop.add_stop_listener(ball)
stop.add_stop_listener(Pendulum())
stop.add_stop_listener(RotatingSpiral())
stop.click("first click")
stop.remove_stop_listener(ball)
stop.click("second click")
```

```
Stop pressed, 3 listener(s):
  ball: stops bouncing (first click)
  pendulum: comes to rest (first click)
  spiral: stops rotating (first click)
Stop pressed, 2 listener(s):
  pendulum: comes to rest (second click)
  spiral: stops rotating (second click)
```

6.15 Antarmuka di Python: Kelas Dasar Abstrak dan Protocol

Python tidak memiliki kata kunci `interface`: ia memanggil metode pada objek apa pun yang diberikan kepadanya, dan jika metode itu ada, metode itu berjalan. Inilah ***duck typing*** — "jika ia berjalan seperti bebek dan berbunyi seperti bebek, ia adalah bebek" — dan harganya adalah bahwa metode yang tidak ada baru ketahuan ketika pemanggilan terjadi. Karena itu Python menawarkan dua cara menuliskan antarmuka, dan keduanya dipakai di bab ini.

Kelas dasar abstrak (ABC, dari modul `abc` [13]) adalah cara *nominal*: sebuah kelas termasuk dalam antarmuka karena ia menyatakannya, dengan mewarisinya. `DBops(ABC)` di **Kode 6.27** dan `PriceRule(ABC)` di Subbab 6.16 adalah ABC. Python menegakkannya pada saat objek dibuat: subkelas yang melupakan sebuah `@abstractmethod` tidak dapat dibuat objeknya, dan `TypeError`-nya menyebutkan metode yang belum ada. ABC juga dapat membawa kode bersama, seperti diperlihatkan `add_several`.

Protocol (dari `typing` [14]) adalah cara *struktural*: sebuah kelas termasuk dalam antarmuka karena ia memiliki metode yang tepat, entah ia pernah mendengar tentang *protocol* itu atau tidak — *duck typing* yang dituliskan. `Measurable` dan `StopListener` adalah *protocol*. Python sendiri *tidak* memeriksa *protocol* ketika program berjalan (kecuali untuk `isinstance`, jika *protocol*-nya bertanda `@runtime_checkable`, dan itu pun hanya memeriksa bahwa nama-nama metodenya ada). Pemeriksaannya dilakukan lebih dulu oleh **pemeriksa tipe statis** seperti mypy, yang membaca anotasi. Untuk berkas yang mengirimkan sebuah `Dog` (tanpa `get_measure`) ke `average`, mypy melaporkan, tanpa menjalankan apa pun:

```
bad_measure.py:9: error: List item 0 has incompatible type "Dog"; expected "Measurable"
[list-item]
Found 1 error in 1 file (checked 1 source file)
```

sedangkan menjalankan berkas yang sama baru gagal ketika sampai di perulangan, dengan `AttributeError: 'Dog' object has no attribute 'get_measure'`. **Tabel 6.6** membandingkan keempat cara menyatakan kontrak.

Tabel 6.6 Empat cara menyatakan "setiap kelas jenis ini wajib menyediakan metode-metode ini".

	Antarmuka Java	Kelas abstrak Java (Bab 7)	ABC Python	*Protocol* Python
kelas bergabung dengan	`implements`	`extends`	mewarisinya	memiliki metode-metodenya
berapa per kelas	berapa saja	tepat satu	berapa saja	berapa saja
boleh memuat kode bersama	metode default dan statis	ya	ya	ya, tetapi jarang diwarisi kelas
boleh memuat *field* (keadaan)	hanya konstanta	ya	ya	tidak
metode yang hilang tertangkap	saat kompilasi	saat kompilasi	saat objek dibuat	oleh pemeriksa tipe, sebelum berjalan

Pakailah ABC ketika Anda memiliki semua implementasinya dan menginginkan kegagalan yang lantang begitu salah satunya tidak lengkap, atau ketika implementasi-implementasi itu harus berbagi kode. Pakailah *Protocol* ketika implementasinya datang dari tempat lain atau Anda tidak ingin memaksakan kelas dasar bersama kepada siapa pun. Keduanya jauh lebih baik daripada antarmuka yang hanya ada di kepala seorang pemrogram.

6.16 Memprogram terhadap Antarmuka: Penentuan Harga di Profitina Laundry

Semua yang ada di bab ini kini bertemu dalam satu perangkat lunak yang bekerja. Profitina Laundry menagih pekerjaannya dengan beberapa cara, dan cara-cara itu terus berubah:

- **cuci lipat** dan **cuci setrika** ditagih per kilogram, setiap kilogram yang sudah dimulai dihitung, dengan minimum 3 kg;
- **cuci kering** (*dry cleaning*) ditagih per potong;
- **cuci sepatu** memiliki satu harga tetap;
- layanan **ekspres** (selesai hari itu juga) 50% lebih mahal, apa pun layanannya;
- **anggota** mendapat potongan 10%, apa pun layanannya dan entah ekspres atau tidak.

Bulan depan akan ada promosi musim hujan; tahun depan, harga khusus untuk hotel. Jika kode nota memuat satu `if` untuk setiap aturan itu, setiap aturan baru berarti menyunting kode nota, dan setiap penyuntingan membahayakan aturan yang sudah ada. Prinsip rancangan yang mencegahnya sudah tua dan pendek [15]:

> ✓ **WAWASAN KUNCI.** *Memprogramlah terhadap antarmuka, bukan terhadap implementasi.* Kode yang membutuhkan harga harus meminta "sesuatu yang dapat menghargai sebuah pesanan" — sebuah antarmuka — dan tidak pernah meminta kelas penentu harga tertentu. Dengan demikian cara-cara baru menentukan harga adalah kelas-kelas baru, dan kode yang memakai harga tidak pernah berubah.

6.16.1 Pesanan

Pertama, benda yang dihargai. **Kode 6.32** adalah sebuah `Order`: nomor tiket, nama pelanggan, berat, jumlah potong, dan apakah pelanggan itu anggota. Konstruktornya menolak hal yang tidak masuk akal — nomor tiket kosong, jumlah negatif, atau pesanan tanpa berat maupun potongan — sehingga tidak ada aturan harga yang perlu mengkhawatirkannya. Pesanan dikenali dari nomor tiketnya, jadi `equals` dan `hashCode` hanya memakai nomor itu, seperti diajarkan Subbab 6.9; `toString` memberikan pandangan untuk pengembang.

Kode 6.32 `Order.java`: pesanan laundry, divalidasi ketika dibangun dan dikenali dari nomor tiketnya.

```java
/**
 * One laundry order as written on the ticket at the counter.
 *
 * <p>Orders are identified by their ticket id: two {@code Order} objects
 * with the same id are the same order.</p>
 */
public final class Order {
    private final String id;
    private final String customer;
    private final double weightKg;
    private final int pieces;
    private final boolean member;

    /**
     * Creates an order.
     *
     * @param id       the ticket id, for example "A001"
     * @param customer the customer's name
     * @param weightKg the weight on the scale, in kilograms (0 if priced per piece)
     * @param pieces   the number of pieces (0 if priced by weight)
     * @param member   whether the customer holds a member card
     * @throws IllegalArgumentException if the id is blank, a quantity is negative,
     *                                  or the order has neither weight nor pieces
     */
    public Order(String id, String customer, double weightKg, int pieces, boolean member) {
        if (id == null || id.isBlank()) {
            throw new IllegalArgumentException("an order needs a ticket id");
        }
        if (weightKg < 0 || pieces < 0) {
            throw new IllegalArgumentException("quantities must not be negative");
        }
        if (weightKg == 0 && pieces == 0) {
            throw new IllegalArgumentException("an order needs a weight or some pieces");
        }
        this.id = id;
        this.customer = customer;
        this.weightKg = weightKg;
        this.pieces = pieces;
        this.member = member;
    }

    /** @return the ticket id */
    public String getId() {
        return id;
    }
```

Kode 6.32 (Lanjutan) *Order.java: pesanan laundry, divalidasi ketika dibangun dan dikenali dari nomor tiketnya.*

```
    /** @return the customer's name */
    public String getCustomer() {
        return customer;
    }

    /** @return the weight in kilograms */
    public double getWeightKg() {
        return weightKg;
    }

    /** @return the number of pieces */
    public int getPieces() {
        return pieces;
    }

    /** @return whether the customer is a member */
    public boolean isMember() {
        return member;
    }

    @Override
    public boolean equals(Object other) {
        if (this == other) {
            return true;
        }
        if (!(other instanceof Order)) {
            return false;
        }
        return id.equals(((Order) other).id);
    }

    @Override
    public int hashCode() {
        return id.hashCode();
    }

    @Override
    public String toString() {
        return "Order[" + id + ", " + customer + ", " + weightKg + " kg, "
                + pieces + " pcs" + (member ? ", member" : "") + "]";
    }
}
```

Getter-nya hanya memuat satu baris `@return`. Untuk *getter* yang namanya sudah menjelaskan segalanya, bentuk singkat itu lazim diterima (meskipun `javadoc` mencatat bahwa keterangan utamanya tidak ada); metode yang mengerjakan pekerjaan sungguhan, seperti `priceOf` di bawah, layak didokumentasikan penuh. **Kode 6.33** adalah `Order` versi Python.

Kode 6.33 `order.py`: pesanan yang sama dalam Python, dengan *property* yang hanya dapat dibaca.

```
"""One laundry order as written on the ticket at the counter."""


class Order:
    """One laundry order. Orders with the same ticket id are the same order."""

    def __init__(self, id, customer, weight_kg, pieces, member):
        """Create an order.

        Args:
            id: the ticket id, for example "A001".
            customer: the customer's name.
            weight_kg: the weight on the scale in kilograms (0 if priced per piece).
            pieces: the number of pieces (0 if priced by weight).
            member: whether the customer holds a member card.

        Raises:
            ValueError: if the id is blank, a quantity is negative,
                or the order has neither weight nor pieces.
        """
        if not id or not id.strip():
            raise ValueError("an order needs a ticket id")
        if weight_kg < 0 or pieces < 0:
            raise ValueError("quantities must not be negative")
        if weight_kg == 0 and pieces == 0:
            raise ValueError("an order needs a weight or some pieces")
        self._id = id
        self._customer = customer
        self._weight_kg = weight_kg
        self._pieces = pieces
        self._member = member

    @property
    def id(self):
        return self._id

    @property
    def customer(self):
        return self._customer

    @property
    def weight_kg(self):
        return self._weight_kg
```

Kode 6.33 (Lanjutan) *order.py: pesanan yang sama dalam Python, dengan property yang hanya dapat dibaca.*

```
    @property
    def pieces(self):
        return self._pieces

    @property
    def member(self):
        return self._member

    def __eq__(self, other):
        if not isinstance(other, Order):
            return NotImplemented
        return self._id == other._id

    def __hash__(self):
        return hash(self._id)

    def __repr__(self):
        extra = ", member" if self._member else ""
        return (f"Order[{self._id}, {self._customer}, {self._weight_kg} kg, "
                f"{self._pieces} pcs{extra}]")
```

6.16.2 Antarmukanya

Kode 6.34 adalah kontrak yang wajib dipegang setiap aturan harga.

Kode 6.34 `PriceRule.java`: antarmuka yang diimplementasikan setiap aturan harga, dengan metode pembantu statis untuk memformat rupiah.

```java
import java.util.Locale;

/**
 * A rule that turns a laundry order into a price.
 *
 * <p>Every way Profitina Laundry charges for its work is a class that
 * implements this interface. Code that only needs a price asks for a
 * {@code PriceRule} and never learns which rule it received.</p>
 */
public interface PriceRule {
    /**
     * Computes the price of an order.
     *
     * @param order the order to price
     * @return the price in whole rupiah, never negative
     * @throws IllegalArgumentException if this rule cannot price the order
     */
    long priceOf(Order order);

    /**
     * Names the rule for the receipt.
     *
     * @return a short, human-readable name such as "wash & fold"
     */
    String name();

    /**
     * Formats an amount the Indonesian way, with dots between thousands.
     *
     * @param amount an amount in whole rupiah
     * @return for example {@code "Rp 28.000"}
     */
    static String rupiah(long amount) {
        return "Rp " + String.format(Locale.ROOT, "%,d", amount).replace(',', '.');
    }
}
```

Antarmuka itu menjanjikan dua metode, `priceOf` dan `name`, dan dokumentasinya adalah bagian dari kontrak: harga dinyatakan dalam **rupiah bulat** (sebuah `long`, agar harga dibandingkan secara tepat), **tidak pernah negatif**, dan sebuah aturan boleh **menolak** pesanan dengan melempar `IllegalArgumentException`. Setiap implementasi wajib menepati kalimat-kalimat itu, dan setiap pemanggil boleh mengandalkannya.

Antarmuka itu juga membawa metode `static`, `rupiah`, yang dipanggil sebagai `PriceRule.rupiah(...)` dan memformat jumlah uang dengan cara Indonesia: `Rp 28.000`. `String.format(Locale.ROOT, "%,d", amount)` menyisipkan koma apa pun pengaturan bahasa komputernya, dan `replace(',', '.')` mengubahnya menjadi titik.

Di Python (**Kode 6.35**), `PriceRule` adalah kelas dasar abstrak, karena Profitina Laundry memiliki semua aturannya dan ingin aturan yang tidak lengkap gagal dengan lantang begitu dibuat. Pembantu pemformatnya adalah fungsi biasa di tingkat modul.

Kode 6.35 `price_rule.py`: `PriceRule` sebagai kelas dasar abstrak, dan pembantu `rupiah`.

```
"""The PriceRule abstract base class, and a helper that formats rupiah."""
from abc import ABC, abstractmethod


class PriceRule(ABC):
    """A rule that turns a laundry order into a price.

    Every way Profitina Laundry charges for its work is a subclass of
    PriceRule. Code that only needs a price asks for a PriceRule and
    never learns which rule it received.
    """

    @abstractmethod
    def price_of(self, order):
        """Compute the price of an order.

        Args:
            order: the Order to price.

        Returns:
            The price in whole rupiah (an int), never negative.

        Raises:
            ValueError: if this rule cannot price the order.
        """

    @abstractmethod
    def name(self):
        """Name the rule for the receipt, e.g. "wash & fold"."""


def rupiah(amount):
    """Format an amount the Indonesian way: rupiah(28000) == 'Rp 28.000'."""
    return "Rp " + f"{amount:,}".replace(",", ".")
```

6.16.3 Tiga aturan dasar

Aturan per kilogram adalah satu-satunya yang aritmetikanya layak dituliskan. Jika timbangan menunjukkan w kilogram, tarifnya r rupiah per kilogram, dan minimumnya m kilogram, harganya adalah

$$P_{\text{kilo}} = r \cdot \max(m, \lceil w \rceil) \tag{6.6}$$

dengan $\lceil w \rceil$, *pembulatan ke atas* (*ceiling*) dari w, yaitu bilangan bulat terkecil yang tidak kurang dari w: $\lceil 3.2 \rceil = 4$, $\lceil 3.0 \rceil = 3$, $\lceil 2.0 \rceil = 2$. Periksalah Persamaan (6.6) terhadap **Tabel 6.3**: untuk $w = 2.0$, $\max(3,2) = 3$ dan $P = 7\,000 \times 3 = 21\,000$; untuk $w = 3.2$, $\max(3,4) = 4$ dan $P = 28\,000$. **Kode 6.36** adalah rumus itu dalam Java, baris demi baris; `Math.ceil` mengembalikan `double`, sehingga sebuah *cast* mengubahnya menjadi `long`.

Kode 6.36 `PerKiloRule.java`: harga per kilogram yang sudah dimulai, dengan minimum.

```java
/**
 * Charges by weight: every started kilogram counts, with a minimum.
 */
public class PerKiloRule implements PriceRule {
    private final String name;
    private final long ratePerKg;
    private final int minimumKg;

    /**
     * Creates a per-kilogram rule.
     *
     * @param name      the name printed on the receipt
     * @param ratePerKg the price of one kilogram, in rupiah
     * @param minimumKg the smallest number of kilograms ever charged
     */
    public PerKiloRule(String name, long ratePerKg, int minimumKg) {
        this.name = name;
        this.ratePerKg = ratePerKg;
        this.minimumKg = minimumKg;
    }

    @Override
    public long priceOf(Order order) {
        if (order.getWeightKg() <= 0) {
            throw new IllegalArgumentException(name + " needs a weight");
        }
        long startedKg = (long) Math.ceil(order.getWeightKg());   // 3,2 kg -> 4 kg
        long chargedKg = Math.max(minimumKg, startedKg);
        return ratePerKg * chargedKg;
    }

    @Override
    public String name() {
        return name;
    }
}
```

Dua aturan dasar lainnya nyaris remeh, dan justru itu kelebihannya. **Kode 6.37** mengalikan harga per potong dengan jumlah potong dan menolak pesanan tanpa potongan; **Kode 6.38** mengembalikan jumlah yang sama untuk pesanan apa pun.

Kode 6.37 `PerItemRule.java`: harga per potong.

```java
/**
 * Charges per piece, for garments that are handled one by one.
 */
public class PerItemRule implements PriceRule {
    private final String name;
    private final long pricePerPiece;

    /**
     * Creates a per-piece rule.
     *
     * @param name          the name printed on the receipt
     * @param pricePerPiece the price of one piece, in rupiah
     */
    public PerItemRule(String name, long pricePerPiece) {
        this.name = name;
        this.pricePerPiece = pricePerPiece;
    }

    @Override
    public long priceOf(Order order) {
        if (order.getPieces() <= 0) {
            throw new IllegalArgumentException(name + " needs at least one piece");
        }
        return pricePerPiece * order.getPieces();
    }

    @Override
    public String name() {
        return name;
    }
}
```

Kode 6.38 `FlatRule.java`: satu harga tetap.

```java
/**
 * Charges one fixed amount, whatever the order contains.
 */
public class FlatRule implements PriceRule {
    private final String name;
    private final long amount;

    /**
     * Creates a flat-price rule.
     *
     * @param name   the name printed on the receipt
     * @param amount the fixed price, in rupiah
     */
    public FlatRule(String name, long amount) {
        this.name = name;
        this.amount = amount;
    }

    @Override
    public long priceOf(Order order) {
        return amount;
    }

    @Override
    public String name() {
        return name;
    }
}
```

`FlatRule` memiliki karier kedua, yang akan kita pakai dalam uji: karena selalu mengembalikan bilangan yang diketahui, ia adalah **pengganti** (*stand-in*) yang sempurna untuk "suatu aturan, entah yang mana, yang menagih 21 001 rupiah". Para penulis uji menyebut pengganti seperti itu

stub. Kemampuan memasang *stub* di mana pun sebuah `PriceRule` diharapkan adalah hadiah langsung dari antarmuka.

6.16.4 Aturan yang membungkus aturan lain

Layanan ekspres dan potongan anggota berbeda jenis: keduanya *mengubah* harga yang dihitung aturan lain. Maka masing-masing *menyimpan sebuah `PriceRule` lain*, bernama `base`, menanyakan harganya, lalu menyesuaikan jawabannya; dan karena masing-masing juga sebuah `PriceRule`, keduanya dapat saling membungkus dalam kombinasi apa pun. Inilah pola **dekorator** (*decorator*) [15]: objek yang mengimplementasikan sebuah antarmuka dengan membungkus objek lain dari antarmuka yang sama lalu menambahkan sesuatu.

Aturan ekspres menambahkan separuh harga dasar. Harga dinyatakan dalam rupiah bulat, jadi separuhnya dihitung dengan pembagian bilangan bulat, yang membuang pecahan apa pun; dalam notasi matematika, dengan $\lfloor x \rfloor$ (*pembulatan ke bawah*, *floor*, dari x) berarti bilangan bulat terbesar yang tidak lebih dari x,

$$P_{\text{ekspres}} = P + \left\lfloor \frac{P}{2} \right\rfloor \tag{6.7}$$

Untuk $P = 28\,000$ hasilnya $28\,000 + 14\,000 = 42\,000$; untuk $P = 21\,001$ yang ganjil hasilnya $21\,001 + \lfloor 10\,500.5 \rfloor = 21\,001 + 10\,500 = 31\,501$. Aturan anggota mengurangkan p persen, lagi-lagi dengan membuang pecahan rupiah dari potongannya:

$$P_{\text{anggota}} = P - \left\lfloor \frac{p \cdot P}{100} \right\rfloor \tag{6.8}$$

Untuk $P = 42\,000$ dan $p = 10$ hasilnya $42\,000 - \lfloor 420\,000/100 \rfloor = 42\,000 - 4\,200 = 37\,800$. **Kode 6.39** dan **Kode 6.40** mengimplementasikan Persamaan (6.7) dan Persamaan (6.8). Perhatikan bahwa di Java, mengalikan sebelum membagi — `normal * percent / 100` — menjaga aritmetikanya tetap tepat; `normal * (percent / 100)` akan menghitung `10 / 100 = 0` dalam bilangan bulat dan tidak memberi potongan sama sekali.

Kode 6.39 `ExpressRule.java`: dekorator yang menambahkan 50%, dibulatkan ke bawah.

```
/**
 * Adds a 50% express surcharge to whatever another rule charges.
 */
public class ExpressRule implements PriceRule {
    private final PriceRule base;

    /**
     * Wraps another rule.
     *
     * @param base the rule that prices the ordinary service
     */
    public ExpressRule(PriceRule base) {
        this.base = base;
    }

    @Override
    public long priceOf(Order order) {
        long normal = base.priceOf(order);
        return normal + normal / 2;          // pembagian bilangan bulat: setengah rupiah
dibuang
    }

    @Override
    public String name() {
        return base.name() + " + express";
    }
}
```

Kode 6.40 `MemberDiscountRule.java`: dekorator yang memotong sekian persen untuk anggota, dan menolak persentase yang mustahil.

```java
/**
 * Gives members a percentage off whatever another rule charges.
 */
public class MemberDiscountRule implements PriceRule {
    private final PriceRule base;
    private final int percent;

    /**
     * Wraps another rule.
     *
     * @param base    the rule that prices the order before the discount
     * @param percent the discount for members, from 0 to 100
     * @throws IllegalArgumentException if {@code percent} is outside 0..100
     */
    public MemberDiscountRule(PriceRule base, int percent) {
        if (percent < 0 || percent > 100) {
            throw new IllegalArgumentException("percent must be 0..100: " + percent);
        }
        this.base = base;
        this.percent = percent;
    }

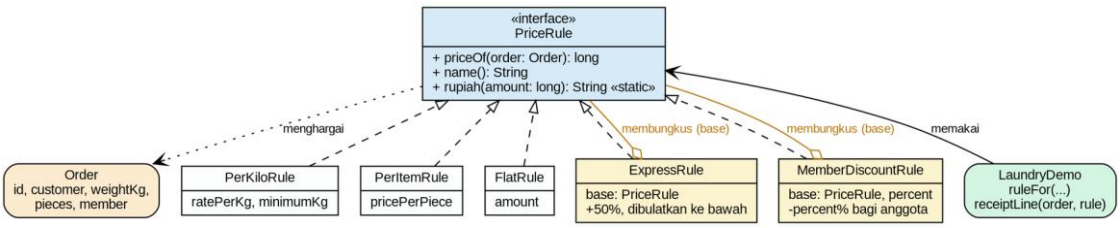
    @Override
    public long priceOf(Order order) {
        long normal = base.priceOf(order);
        if (!order.isMember()) {
            return normal;
        }
        return normal - normal * percent / 100;   // potongannya dibulatkan ke bawah
    }

    @Override
    public String name() {
        return base.name() + " + member " + percent + "%";
    }
}
```

Harga yang lengkap adalah sebuah *rantai* aturan. Untuk 3,2 kg cuci lipat milik Budi, ekspres, sebagai anggota, rantainya adalah `new MemberDiscountRule(new ExpressRule(new PerKiloRule("wash & fold", 7_000, 3)), 10)`, dan harganya adalah ketiga rumus yang diterapkan dari dalam ke luar:

$$7\,000 \cdot \max(3, \lceil 3.2 \rceil) = 28\,000 \overset{\text{ekspres}}{\rightarrow} 42\,000 \overset{\text{anggota}}{\rightarrow} 37\,800 \tag{6.9}$$

Persamaan (6.9) juga sebuah kasus uji, seperti akan kita lihat. Seluruh rancangannya digambar pada **Gambar 6.7**.



Gambar 6.7 Rancangan penentuan harga: lima kelas mengimplementasikan `PriceRule`, dua di antaranya dengan membungkus `PriceRule` lain; program demo hanya bergantung pada antarmuka.

Di Python, kelima aturan itu muat dengan leluasa dalam satu modul, **Kode 6.41**. Setiap subkelas `PriceRule` mengimplementasikan `price_of` dan `name`; `//` di Python adalah pembagian dengan pembulatan ke bawah, yang langsung memberikan Persamaan (6.7) dan Persamaan (6.8), dan `math.ceil` sudah mengembalikan `int`.

Kode 6.41 `rules.py`: kelima aturan harga dalam Python.

```
"""The price rules of Profitina Laundry: five subclasses of PriceRule."""
import math

from price_rule import PriceRule


class PerKiloRule(PriceRule):
    """Charges by weight: every started kilogram counts, with a minimum."""

    def __init__(self, name, rate_per_kg, minimum_kg):
        self._name = name
        self._rate_per_kg = rate_per_kg
        self._minimum_kg = minimum_kg

    def price_of(self, order):
        if order.weight_kg <= 0:
            raise ValueError(f"{self._name} needs a weight")
        started_kg = math.ceil(order.weight_kg)          # 3,2 kg -> 4 kg
        charged_kg = max(self._minimum_kg, started_kg)
        return self._rate_per_kg * charged_kg

    def name(self):
        return self._name


class PerItemRule(PriceRule):
    """Charges per piece, for garments that are handled one by one."""

    def __init__(self, name, price_per_piece):
        self._name = name
        self._price_per_piece = price_per_piece

    def price_of(self, order):
        if order.pieces <= 0:
            raise ValueError(f"{self._name} needs at least one piece")
        return self._price_per_piece * order.pieces

    def name(self):
        return self._name


class FlatRule(PriceRule):
    """Charges one fixed amount, whatever the order contains."""
```

Kode 6.41 (Lanjutan) *rules.py: kelima aturan harga dalam Python.*

```
    def __init__(self, name, amount):
        self._name = name
        self._amount = amount

    def price_of(self, order):
        return self._amount

    def name(self):
        return self._name


class ExpressRule(PriceRule):
    """Adds a 50% express surcharge to whatever another rule charges."""

    def __init__(self, base):
        self._base = base

    def price_of(self, order):
        normal = self._base.price_of(order)
        return normal + normal // 2          # pembagian lantai: setengah rupiah dibuang

    def name(self):
        return self._base.name() + " + express"


class MemberDiscountRule(PriceRule):
    """Gives members a percentage off whatever another rule charges."""

    def __init__(self, base, percent):
        if not 0 <= percent <= 100:
            raise ValueError(f"percent must be 0..100: {percent}")
        self._base = base
        self._percent = percent

    def price_of(self, order):
        normal = self._base.price_of(order)
        if not order.member:
            return normal
        return normal - normal * self._percent // 100   # potongannya dibulatkan ke bawah

    def name(self):
        return f"{self._base.name()} + member {self._percent}%"
```

Tabel 6.7 merangkum aturan-aturan itu beserta kontraknya.

Tabel 6.7 Kelima aturan harga Profitina Laundry.

Aturan	Harga	Menolak
`PerKiloRule(name, r, m)`	Persamaan (6.6)	pesanan tanpa berat
`PerItemRule(name, price)`	harga × jumlah potong	pesanan tanpa potongan
`FlatRule(name, amount)`	jumlah tetap	tidak ada
`ExpressRule(base)`	Persamaan (6.7) atas harga dasar	apa pun yang ditolak aturan dasarnya
`MemberDiscountRule(base, p)`	Persamaan (6.8) untuk anggota, harga dasar untuk yang lain	p di luar 0..100 (ketika dibangun)

6.16.5 Program yang memakai aturan-aturan itu

Kode 6.42 menghargai pesanan-pesanan satu pagi. Kedua metodenya memperlihatkan dua sisi dari memprogram terhadap antarmuka.

Kode 6.42 `LaundryDemo.java`: memilih aturan sekali, lalu memakainya hanya melalui antarmuka.

```
import java.util.List;

/**
 * Prices one morning's orders at Profitina Laundry.
 */
public class LaundryDemo {
    /**
     * Chooses the price rule for a service code.
     *
     * @param service one of WASH, WASH_IRON, DRYCLEAN, SHOES
     * @param express whether the customer wants the order the same day
     * @return the rule, already wrapped for express and member pricing
     * @throws IllegalArgumentException for an unknown service code
     */
    static PriceRule ruleFor(String service, boolean express) {
        PriceRule rule;
        switch (service) {
            case "WASH":
                rule = new PerKiloRule("wash & fold", 7_000, 3);
                break;
            case "WASH_IRON":
                rule = new PerKiloRule("wash & iron", 10_000, 3);
                break;
            case "DRYCLEAN":
                rule = new PerItemRule("dry clean", 25_000);
                break;
            case "SHOES":
                rule = new FlatRule("shoe cleaning", 35_000);
                break;
            default:
                throw new IllegalArgumentException("unknown service: " + service);
        }
        if (express) {
            rule = new ExpressRule(rule);
        }
        return new MemberDiscountRule(rule, 10);   // anggota selalu mendapat potongan 10%
    }
```

Kode 6.42 (Lanjutan) *LaundryDemo.java: memilih aturan sekali, lalu memakainya hanya melalui antarmuka.*

```
    /**
     * Formats one line of the receipt; it works for any PriceRule at all.
     *
     * @param order the order
     * @param rule  the rule that prices it
     * @return the receipt line
     */
    static String receiptLine(Order order, PriceRule rule) {
        return String.format("%-5s %-6s %-36s %10s",
                order.getId(), order.getCustomer(), rule.name(),
                PriceRule.rupiah(rule.priceOf(order)));
    }

    public static void main(String[] args) {
        List<Order> orders = List.of(
                new Order("A001", "Sari", 2.0, 0, false),
                new Order("A002", "Budi", 3.2, 0, true),
                new Order("A003", "Dewi", 0, 4, true),
                new Order("A004", "Eko", 0, 1, false),
                new Order("A005", "Sari", 5.5, 0, false),
                new Order("A006", "Fajar", 2.5, 0, false));
        String[] services = {"WASH", "WASH_IRON", "DRYCLEAN", "SHOES", "WASH", "DRYCLEAN"};
        boolean[] express = {false, true, false, false, true, false};

        long total = 0;
        for (int i = 0; i < orders.size(); i++) {
            Order order = orders.get(i);
            PriceRule rule = ruleFor(services[i], express[i]);
            try {
                System.out.println(receiptLine(order, rule));
                total += rule.priceOf(order);
            } catch (IllegalArgumentException e) {
                System.out.println(order.getId() + "  cannot be priced: " +
e.getMessage());
            }
        }
        System.out.println("Total for the morning: " + PriceRule.rupiah(total));
    }
}
```

Kompilasi semua kelas laundry bersama-sama (`javac Order.java PriceRule.java PerKiloRule.java PerItemRule.java FlatRule.java ExpressRule.java MemberDiscountRule.java LaundryDemo.java`) lalu jalankan `java LaundryDemo`:

```
A001  Sari   wash & fold + member 10%              Rp 21.000
A002  Budi   wash & iron + express + member 10%    Rp 54.000
A003  Dewi   dry clean + member 10%                Rp 90.000
A004  Eko    shoe cleaning + member 10%            Rp 35.000
A005  Sari   wash & fold + express + member 10%    Rp 63.000
A006  cannot be priced: dry clean needs at least one piece
Total for the morning: Rp 263.000
```

`ruleFor` adalah satu-satunya tempat di program yang mengenal kelas-kelas aturan konkret. Ia adalah *factory* (pabrik) kecil: mengubah kode layanan menjadi rantai aturan siap pakai dan mengembalikannya dengan tipe yang dideklarasikan `PriceRule`. `receiptLine` tidak mengenal apa pun selain antarmuka. Metode itu bekerja untuk setiap aturan yang ada hari ini dan setiap aturan yang ditulis siapa pun besok, dan tidak akan pernah perlu diubah karena harga baru. Perulangan di `main` menangani penolakan dengan `try`/`catch` biasa: pesanan A006 meminta cuci kering dengan berat tetapi tanpa potongan, `PerItemRule` menolak, dan penolakan itu melewati pembungkus potongan anggota tanpa berubah, persis seperti dijanjikan **Tabel 6.7**.

Periksalah nota itu terhadap rumus-rumusnya. A001: 2,0 kg ditagih sebagai 3 kg, $3 \times 7\,000 = 21\,000$, bukan anggota. A002: Persamaan (6.9) lagi, tetapi dengan 10 000 per kilogram: $40\,000 \rightarrow 60\,000 \rightarrow 54\,000$. A003: $4 \times 25\,000 = 100\,000$, dipotong 10% untuk anggota, 90 000. A004: tetap 35 000. A005: 5,5 kg menjadi 6 kg, 42 000, ekspres 63 000. Totalnya $21\,000 + 54\,000 + 90\,000 + 35\,000 + 63\,000 = 263\,000$. Program Python, **Kode 6.43**, mencetak nota yang persis sama.

Kode 6.43 `laundry_demo.py`: pagi yang sama dalam Python.

```
"""Prices one morning's orders at Profitina Laundry."""
from order import Order
from price_rule import PriceRule, rupiah
from rules import ExpressRule, FlatRule, MemberDiscountRule, PerItemRule, PerKiloRule


def rule_for(service, express):
    """Choose the price rule for a service code, wrapped for express and members."""
    if service == "WASH":
        rule = PerKiloRule("wash & fold", 7_000, 3)
    elif service == "WASH_IRON":
        rule = PerKiloRule("wash & iron", 10_000, 3)
    elif service == "DRYCLEAN":
        rule = PerItemRule("dry clean", 25_000)
    elif service == "SHOES":
        rule = FlatRule("shoe cleaning", 35_000)
    else:
        raise ValueError(f"unknown service: {service}")
    if express:
        rule = ExpressRule(rule)
    return MemberDiscountRule(rule, 10)     # anggota selalu mendapat potongan 10%


def receipt_line(order, rule: PriceRule):
    """Format one line of the receipt; it works for any PriceRule at all."""
    return (f"{order.id:<5} {order.customer:<6} {rule.name():<36} "
            f"{rupiah(rule.price_of(order)):>10}")


orders = [Order("A001", "Sari", 2.0, 0, False),
          Order("A002", "Budi", 3.2, 0, True),
          Order("A003", "Dewi", 0, 4, True),
          Order("A004", "Eko", 0, 1, False),
          Order("A005", "Sari", 5.5, 0, False),
          Order("A006", "Fajar", 2.5, 0, False)]
services = ["WASH", "WASH_IRON", "DRYCLEAN", "SHOES", "WASH", "DRYCLEAN"]
express = [False, True, False, False, True, False]
```

Kode 6.43 (Lanjutan) *laundry_demo.py: pagi yang sama dalam Python.*

```
total = 0
for order, service, fast in zip(orders, services, express):
    rule = rule_for(service, fast)
    try:
        print(receipt_line(order, rule))
        total += rule.price_of(order)
    except ValueError as e:
        print(f"{order.id}  cannot be priced: {e}")
print("Total for the morning:", rupiah(total))
```

```
A001  Sari   wash & fold + member 10%              Rp 21.000
A002  Budi   wash & iron + express + member 10%    Rp 54.000
A003  Dewi   dry clean + member 10%                Rp 90.000
A004  Eko    shoe cleaning + member 10%            Rp 35.000
A005  Sari   wash & fold + express + member 10%    Rp 63.000
A006  cannot be priced: dry clean needs at least one piece
Total for the morning: Rp 263.000
```

6.16.6 Menguji setiap aturan

Mesin harga yang meleset satu rupiah pada satu dari seribu pesanan adalah mesin yang salah, dan tak seorang pun menyadarinya selama berbulan-bulan. Karena itu setiap aturan diuji dengan teknik Subbab 6.8: setiap kelas ekuivalensi pada **Tabel 6.3**, kedua sisi setiap batas, setiap penolakan yang dijanjikan, pembulatan Persamaan (6.7) dengan harga *ganjil*, anggota dan bukan anggota, tumpukan Persamaan (6.9), dan pembantu pemformat. **Kode 6.44** adalah kelas uji JUnit 6-nya.

Kode 6.44 `PriceRuleTest.java`: uji JUnit 6 untuk setiap aturan harga.

```
import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertThrows;

import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Test;

/**
 * JUnit 5 tests for every PriceRule of Profitina Laundry.
 */
class PriceRuleTest {
    private final PriceRule wash = new PerKiloRule("wash & fold", 7_000, 3);

    private static Order byWeight(double kg, boolean member) {
        return new Order("T1", "Test", kg, 0, member);
    }

    private static Order byPieces(int pieces, boolean member) {
        return new Order("T2", "Test", 0, pieces, member);
    }

    @Test
    @DisplayName("a light load is charged the 3 kg minimum")
    void perKiloChargesMinimum() {
        assertEquals(21_000, wash.priceOf(byWeight(2.0, false)));
    }

    @Test
    @DisplayName("exactly 3 kg is charged as 3 kg")
    void perKiloExactBoundary() {
        assertEquals(21_000, wash.priceOf(byWeight(3.0, false)));
    }

    @Test
    @DisplayName("a started kilogram counts as a whole kilogram")
    void perKiloRoundsUp() {
        assertEquals(28_000, wash.priceOf(byWeight(3.01, false)));
        assertEquals(28_000, wash.priceOf(byWeight(3.2, false)));
    }

    @Test
    @DisplayName("per-kilo pricing refuses an order with no weight")
    void perKiloRejectsZeroWeight() {
        assertThrows(IllegalArgumentException.class, () -> wash.priceOf(byPieces(2,
false)));
    }
```

Kode 6.44 (Lanjutan) *PriceRuleTest.java: uji JUnit 6 untuk setiap aturan harga.*

```
    @Test
    @DisplayName("per-item pricing multiplies by the number of pieces")
    void perItemMultiplies() {
        PriceRule dryClean = new PerItemRule("dry clean", 25_000);
        assertEquals(100_000, dryClean.priceOf(byPieces(4, false)));
    }

    @Test
    @DisplayName("express adds half, dropping half a rupiah")
    void expressAddsHalfRoundingDown() {
        PriceRule odd = new FlatRule("odd", 21_001);       // pengganti dengan harga ganjil
        assertEquals(31_501, new ExpressRule(odd).priceOf(byPieces(1, false)));
    }

    @Test
    @DisplayName("the member discount applies to members only")
    void memberDiscountOnlyForMembers() {
        PriceRule rule = new MemberDiscountRule(new FlatRule("flat", 50_000), 10);
        assertEquals(45_000, rule.priceOf(byPieces(1, true)));
        assertEquals(50_000, rule.priceOf(byPieces(1, false)));
    }

    @Test
    @DisplayName("a discount outside 0..100 percent is refused")
    void memberDiscountRejectsBadPercent() {
        assertThrows(IllegalArgumentException.class,
                () -> new MemberDiscountRule(wash, 101));
    }

    @Test
    @DisplayName("rules stack: per-kilo, then express, then member")
    void rulesStack() {
        PriceRule rule = new MemberDiscountRule(new ExpressRule(wash), 10);
        assertEquals(37_800, rule.priceOf(byWeight(3.2, true)));
        assertEquals("wash & fold + express + member 10%", rule.name());
    }
```

Kode 6.44 (Lanjutan) *PriceRuleTest.java: uji JUnit 6 untuk setiap aturan harga.*

```
    @Test
    @DisplayName("rupiah amounts get dots between thousands")
    void rupiahFormatting() {
        assertEquals("Rp 0", PriceRule.rupiah(0));
        assertEquals("Rp 999", PriceRule.rupiah(999));
        assertEquals("Rp 1.000", PriceRule.rupiah(1_000));
        assertEquals("Rp 1.234.567", PriceRule.rupiah(1_234_567));
    }
}
```

Dua pembantu, `byWeight` dan `byPieces`, menjaga langkah *arrange* tetap satu pemanggilan pendek. Perhatikan `expressAddsHalfRoundingDown`: uji itu memakai *stub* `new FlatRule("odd", 21_001)` alih-alih aturan layanan sungguhan, sehingga aturan ekspres diuji *sendirian*. Melalui antarmuka, aturan itu tidak dapat membedakannya — itulah manfaat antarmuka bagi pengujian yang dijanjikan di Subbab 6.5.

Jalankan di tampilan *Testing* VS Code, atau di terminal dengan *jar* di `lib` seperti di Subbab 6.7.2:

```
javac -cp "lib/*" -d out *.java
java -jar lib/junit-platform-console-standalone-6.1.3.jar execute -cp out --select-class
PriceRuleTest
```

Bagi pembaca yang tidak memiliki *jar* JUnit, dan untuk memperlihatkan putusan nyata di sini, **Kode 6.45** menjalankan pemeriksaan yang sama sebagai program biasa bergaya `DogCheck`,

dengan satu pembantu tambahan, `checkThrows`, yang mengharapkan eksepsi. Kode yang diuji diserahkan kepadanya sebagai lambda, sebagaimana pada `assertThrows` di JUnit.

Kode 6.45 `PriceRuleCheck.java`: uji-uji aturan harga sebagai program biasa.

```
/**
 * The PriceRuleTest checks as a plain program: no library needed.
 */
public class PriceRuleCheck {
    private static int passed = 0;
    private static int failed = 0;

    static void check(String what, Object expected, Object actual) {
        if (expected.equals(actual)) {
            passed++;
            System.out.println("PASS  " + what);
        } else {
            failed++;
            System.out.println("FAIL  " + what + ": expected " + expected + " but was " +
actual);
        }
    }

    static void checkThrows(String what, Runnable code) {
        try {
            code.run();
            failed++;
            System.out.println("FAIL  " + what + ": no exception was thrown");
        } catch (IllegalArgumentException e) {
            passed++;
            System.out.println("PASS  " + what + " (" + e.getMessage() + ")");
        }
    }

    static Order byWeight(double kg, boolean member) {
        return new Order("T1", "Test", kg, 0, member);
    }

    static Order byPieces(int pieces, boolean member) {
        return new Order("T2", "Test", 0, pieces, member);
    }

    public static void main(String[] args) {
        PriceRule wash = new PerKiloRule("wash & fold", 7_000, 3);
        check("light load pays the 3 kg minimum", 21_000L, wash.priceOf(byWeight(2.0,
false)));
        check("exactly 3 kg is 3 kg", 21_000L, wash.priceOf(byWeight(3.0, false)));
        check("3.01 kg is charged as 4 kg", 28_000L, wash.priceOf(byWeight(3.01, false)));
        check("3.2 kg is charged as 4 kg", 28_000L, wash.priceOf(byWeight(3.2, false)));
        checkThrows("per-kilo refuses zero weight", () -> wash.priceOf(byPieces(2,
false)));
```

Kode 6.45 (Lanjutan) *PriceRuleCheck.java: uji-uji aturan harga sebagai program biasa.*

```
        PriceRule dryClean = new PerItemRule("dry clean", 25_000);
        check("4 pieces of dry cleaning", 100_000L, dryClean.priceOf(byPieces(4, false)));

        PriceRule odd = new FlatRule("odd", 21_001);
        check("express drops half a rupiah", 31_501L, new
ExpressRule(odd).priceOf(byPieces(1, false)));

        PriceRule member = new MemberDiscountRule(new FlatRule("flat", 50_000), 10);
        check("member gets 10% off", 45_000L, member.priceOf(byPieces(1, true)));
        check("non-member pays full price", 50_000L, member.priceOf(byPieces(1, false)));
        checkThrows("101% discount is refused", () -> new MemberDiscountRule(wash, 101));

        PriceRule stacked = new MemberDiscountRule(new ExpressRule(wash), 10);
        check("stacked rules", 37_800L, stacked.priceOf(byWeight(3.2, true)));
        check("stacked name", "wash & fold + express + member 10%", stacked.name());

        check("Rp 0", "Rp 0", PriceRule.rupiah(0));
        check("Rp 1.234.567", "Rp 1.234.567", PriceRule.rupiah(1_234_567));

        System.out.println(passed + " passed, " + failed + " failed");
        if (failed > 0) {
            System.exit(1);
        }
    }
}
```

```
PASS  light load pays the 3 kg minimum
PASS  exactly 3 kg is 3 kg
PASS  3.01 kg is charged as 4 kg
PASS  3.2 kg is charged as 4 kg
PASS  per-kilo refuses zero weight (wash & fold needs a weight)
PASS  4 pieces of dry cleaning
PASS  express drops half a rupiah
PASS  member gets 10% off
PASS  non-member pays full price
PASS  101% discount is refused (percent must be 0..100: 101)
PASS  stacked rules
PASS  stacked name
PASS  Rp 0
PASS  Rp 1.234.567
14 passed, 0 failed
```

> ⚠ **JEBAKAN.** Nilai-nilai yang diharapkan di **Kode 6.45** ditulis `21_000L`, dengan `L`, dan itu bukan kebetulan. `check` menerima dua `Object`, sehingga Java melakukan *boxing* pada kedua bilangan: harga sebenarnya, sebuah `long`, menjadi `Long`; `21_000` biasa akan menjadi `Integer`. Dan `Integer.valueOf(21000).equals(Long.valueOf(21000))` bernilai **false** — `equals` mengembalikan salah untuk objek dari kelas berbeda, sebagaimana diwajibkan kontrak di Subbab 6.9. Tanpa `L`, setiap pemeriksaan akan gagal dengan pesan membingungkan "expected 21000 but was 21000". `assertEquals(long, long)` di JUnit menghindari jebakan ini dengan membandingkan nilai primitif.

Uji Python memakai pytest, dan memakai satu fitur yang di JUnit disimpan dalam modul terpisah: **uji berparameter** (*parametrized test*). Dekorator `@pytest.mark.parametrize` menjalankan satu fungsi uji sekali untuk setiap baris sebuah tabel, sehingga keempat baris **Tabel 6.3** menjadi empat uji dengan satu tubuh. **Kode 6.46** adalah berkasnya.

Kode 6.46 `test_price_rules.py`: uji pytest untuk setiap aturan harga, dengan batas-batas berparameter.

```python
"""pytest tests for every PriceRule of Profitina Laundry."""
import pytest

from order import Order
from price_rule import rupiah
from rules import ExpressRule, FlatRule, MemberDiscountRule, PerItemRule, PerKiloRule

WASH = PerKiloRule("wash & fold", 7_000, 3)


def by_weight(kg, member=False):
    return Order("T1", "Test", kg, 0, member)


def by_pieces(pieces, member=False):
    return Order("T2", "Test", 0, pieces, member)


@pytest.mark.parametrize("kg, expected", [
    (2.0, 21_000),      # di bawah minimum
    (3.0, 21_000),      # tepat di batas
    (3.01, 28_000),     # sedikit melewatinya: kilogram yang sudah dimulai dihitung
    (3.2, 28_000),
])
def test_per_kilo_minimum_and_rounding(kg, expected):
    assert WASH.price_of(by_weight(kg)) == expected


def test_per_kilo_rejects_zero_weight():
    with pytest.raises(ValueError, match="needs a weight"):
        WASH.price_of(by_pieces(2))


def test_per_item_multiplies():
    assert PerItemRule("dry clean", 25_000).price_of(by_pieces(4)) == 100_000


def test_express_adds_half_rounding_down():
    odd = FlatRule("odd", 21_001)            # pengganti dengan harga ganjil
    assert ExpressRule(odd).price_of(by_pieces(1)) == 31_501
```

Kode 6.46 (Lanjutan) *test_price_rules.py: uji pytest untuk setiap aturan harga, dengan batas-batas berparameter.*

```
def test_member_discount_only_for_members():
    rule = MemberDiscountRule(FlatRule("flat", 50_000), 10)
    assert rule.price_of(by_pieces(1, member=True)) == 45_000
    assert rule.price_of(by_pieces(1, member=False)) == 50_000


def test_member_discount_rejects_bad_percent():
    with pytest.raises(ValueError):
        MemberDiscountRule(WASH, 101)


def test_rules_stack():
    rule = MemberDiscountRule(ExpressRule(WASH), 10)
    assert rule.price_of(by_weight(3.2, member=True)) == 37_800
    assert rule.name() == "wash & fold + express + member 10%"


@pytest.mark.parametrize("amount, text", [
    (0, "Rp 0"), (999, "Rp 999"), (1_000, "Rp 1.000"), (1_234_567, "Rp 1.234.567"),
])
def test_rupiah_formatting(amount, text):
    assert rupiah(amount) == text
```

`python3 -m pytest -v test_price_rules.py` memberikan satu baris per uji, setiap baris parameter didaftar terpisah (kepalanya, yang menyebutkan versi Python dan folder Anda, dihilangkan di sini):

```
test_price_rules.py::test_per_kilo_minimum_and_rounding[2.0-21000] PASSED [  7%]
test_price_rules.py::test_per_kilo_minimum_and_rounding[3.0-21000] PASSED [ 14%]
test_price_rules.py::test_per_kilo_minimum_and_rounding[3.01-28000] PASSED [ 21%]
test_price_rules.py::test_per_kilo_minimum_and_rounding[3.2-28000] PASSED [ 28%]
test_price_rules.py::test_per_kilo_rejects_zero_weight PASSED            [ 35%]
test_price_rules.py::test_per_item_multiplies PASSED                     [ 42%]
test_price_rules.py::test_express_adds_half_rounding_down PASSED         [ 50%]
test_price_rules.py::test_member_discount_only_for_members PASSED        [ 57%]
test_price_rules.py::test_member_discount_rejects_bad_percent PASSED     [ 64%]
test_price_rules.py::test_rules_stack PASSED                             [ 71%]
test_price_rules.py::test_rupiah_formatting[0-Rp 0] PASSED               [ 78%]
test_price_rules.py::test_rupiah_formatting[999-Rp 999] PASSED           [ 85%]
test_price_rules.py::test_rupiah_formatting[1000-Rp 1.000] PASSED        [ 92%]
test_price_rules.py::test_rupiah_formatting[1234567-Rp 1.234.567] PASSED [100%]
============================== 14 passed in 0.02s ==============================
```

Untuk melihat apa yang dilakukan pytest ketika sebuah uji gagal, kami mengubah baris terakhir `ExpressRule.price_of` menjadi `return round(normal * 1.5)` — "perbaikan" yang masuk akal, yang membulatkan ke rupiah terdekat alih-alih ke bawah — lalu menjalankan pytest lagi (`-q` untuk ringkasan yang senyap). Bagian laporan yang relevan:

```
......F.......                                                          [100%]
=================================== FAILURES ===================================
_____________________ test_express_adds_half_rounding_down _____________________

    def test_express_adds_half_rounding_down():
        odd = FlatRule("odd", 21_001)           # a stand-in with an odd price
>       assert ExpressRule(odd).price_of(by_pieces(1)) == 31_501
E       assert 31502 == 31501
E        +  where 31502 = price_of(Order[T2, Test, 0 kg, 1 pcs])
```

pytest menandai baris yang gagal dengan `>` lalu membongkar ekspresinya: sisi kiri bernilai `31502`, berasal dari `price_of` pada `Order[T2, Test, 0 kg, 1 pcs]` — `__repr__` kita yang membuat laporan mudah dibaca. 21 001 × 1.5 = 31 501.5, dan `round` di Python membulatkan setengah ke

bilangan *genap* terdekat, 31 502. Apakah laundry menginginkannya adalah keputusan bisnis; yang penting, perubahan itu tidak dapat menyelinap tanpa ketahuan, dan harga ganjil dipilih justru untuk itu.

Akhirnya, **Kode 6.47** adalah pemeriksa Python biasa, sepadan baris demi baris dengan **Kode 6.45**.

Kode 6.47 `price_rule_check.py`: uji-uji aturan harga sebagai program Python biasa.

```
"""The test_price_rules checks as a plain program: no library needed."""
import sys

from order import Order
from price_rule import rupiah
from rules import ExpressRule, FlatRule, MemberDiscountRule, PerItemRule, PerKiloRule

passed = 0
failed = 0


def check(what, expected, actual):
    global passed, failed
    if expected == actual:
        passed += 1
        print("PASS ", what)
    else:
        failed += 1
        print("FAIL ", what + f": expected {expected!r} but was {actual!r}")


def check_raises(what, code):
    global passed, failed
    try:
        code()
        failed += 1
        print("FAIL ", what + ": no exception was raised")
    except ValueError as e:
        passed += 1
        print("PASS ", what, f"({e})")


def by_weight(kg, member=False):
    return Order("T1", "Test", kg, 0, member)


def by_pieces(pieces, member=False):
    return Order("T2", "Test", 0, pieces, member)
```

Kode 6.47 (Lanjutan) *price_rule_check.py: uji-uji aturan harga sebagai program Python biasa.*

```
wash = PerKiloRule("wash & fold", 7_000, 3)
check("light load pays the 3 kg minimum", 21_000, wash.price_of(by_weight(2.0)))
check("exactly 3 kg is 3 kg", 21_000, wash.price_of(by_weight(3.0)))
check("3.01 kg is charged as 4 kg", 28_000, wash.price_of(by_weight(3.01)))
check("3.2 kg is charged as 4 kg", 28_000, wash.price_of(by_weight(3.2)))
check_raises("per-kilo refuses zero weight", lambda: wash.price_of(by_pieces(2)))

dry_clean = PerItemRule("dry clean", 25_000)
check("4 pieces of dry cleaning", 100_000, dry_clean.price_of(by_pieces(4)))

odd = FlatRule("odd", 21_001)
check("express drops half a rupiah", 31_501, ExpressRule(odd).price_of(by_pieces(1)))

member = MemberDiscountRule(FlatRule("flat", 50_000), 10)
check("member gets 10% off", 45_000, member.price_of(by_pieces(1, member=True)))
check("non-member pays full price", 50_000, member.price_of(by_pieces(1)))
check_raises("101% discount is refused", lambda: MemberDiscountRule(wash, 101))

stacked = MemberDiscountRule(ExpressRule(wash), 10)
check("stacked rules", 37_800, stacked.price_of(by_weight(3.2, member=True)))
check("stacked name", "wash & fold + express + member 10%", stacked.name())

check("Rp 0", "Rp 0", rupiah(0))
check("Rp 1.234.567", "Rp 1.234.567", rupiah(1_234_567))

print(f"{passed} passed, {failed} failed")
if failed > 0:
    sys.exit(1)
```

```
PASS  light load pays the 3 kg minimum
PASS  exactly 3 kg is 3 kg
PASS  3.01 kg is charged as 4 kg
PASS  3.2 kg is charged as 4 kg
PASS  per-kilo refuses zero weight (wash & fold needs a weight)
PASS  4 pieces of dry cleaning
PASS  express drops half a rupiah
PASS  member gets 10% off
PASS  non-member pays full price
PASS  101% discount is refused (percent must be 0..100: 101)
PASS  stacked rules
PASS  stacked name
PASS  Rp 0
PASS  Rp 1.234.567
14 passed, 0 failed
```

6.16.7 Apa yang kita peroleh dari antarmuka

Hitunglah apa yang dibutuhkan untuk menambahkan promosi musim hujan — misalnya potongan 5% untuk setiap pesanan di bulan Januari. Satu kelas baru, `SeasonalRule`, yang membungkus aturan dasar persis seperti `MemberDiscountRule`; satu baris di `ruleFor`; dan segelintir uji. `Order`, `PriceRule`, kelima aturan yang ada, `receiptLine`, dan setiap uji yang ada tidak tersentuh — dan uji-uji lama terus membuktikan bahwa harga-harga lama tidak berubah. Soal Latihan 6.11 meminta Anda menambahkan aturan semacam ini.

Itulah inti seluruh bab dalam satu paragraf: *dokumentasi* `PriceRule` menyatakan apa yang dijanjikan setiap aturan, *uji* memeriksa bahwa setiap aturan menepati janji itu, dan *antarmuka* memungkinkan bagian lain program mengandalkan janji itu tanpa tahu aturan mana yang sedang dipegangnya.

6.17 Profitina dalam Praktik

Profitina, platform visibilitas AI yang berjalan di app.profitina.com, mengukur seberapa siap sebuah situs web untuk dikutip oleh mesin penjawab berbasis AI, dan ia melakukannya dari banyak sinyal yang saling bebas: apakah isi halaman dapat diekstrak dengan bersih, apakah halaman membawa data terstruktur, seberapa kokoh dan tepercaya situs itu secara teknis, dan lain-lain. Bentuk bagian produk itu adalah bentuk Subbab 6.16. Setiap sinyal adalah kelas tersendiri di balik **satu antarmuka** — kira-kira "diberikan sebuah situs yang telah dipindai, hasilkan skor dan penjelasannya" — dan kode yang menyusun laporan hanya meminta antarmuka itu. Sinyal baru adalah kelas baru; kode laporan, dasbor, dan grafik tren mingguan tidak berubah. (Bagaimana setiap sinyal dinilai adalah urusan produk itu sendiri; yang menjadi pelajaran adalah *strukturnya*.)

Gagasan yang sama melindungi produk di titik-titik ia bersentuhan dengan hal yang tidak dikendalikannya. Profitina menjalankan model bahasanya secara lokal (Qwen2.5 melalui Ollama), dan kode yang membutuhkan jawaban model paling baik ditulis terhadap antarmuka yang sempit: "diberikan sebuah *prompt*, kembalikan jawaban, atau katakan bahwa jawaban tidak tersedia". Dalam uji, pengganti yang mengembalikan jawaban tetap mengambil tempat model itu, persis seperti `FlatRule` menggantikan aturan harga sungguhan, sehingga logika di sekitar model diuji dalam hitungan milidetik. Dalam produksi, ketika tidak ada model yang tersedia, Profitina mengatakannya terus terang alih-alih menyajikan templat seolah-olah jawaban AI: pengganti adalah untuk uji, tidak pernah untuk mengelabui pengguna.

Di sana pengujian adalah gerbang rilis. Seluruh rangkaian uji otomatis — jauh di atas seribu uji, dalam Python untuk server dan dalam JavaScript untuk antarmuka pengguna — harus hijau sebelum sebuah rilis diluncurkan, dan rilis baru dianggap selesai ketika versi yang baru dipasang menjawab pemeriksaan kesehatannya. Itulah status keluar bukan nol milik `DogCheck`, diterapkan pada seluruh produk: mesin, bukan suasana hati, yang memutuskan apakah kode siap.

Dokumentasi dan kesamaan juga muncul. Servernya ditulis dalam Python dengan FastAPI, yang membangun halaman rujukan API interaktif langsung dari anotasi tipe dan *docstring* — gagasan `pydoc` yang telah dewasa. Ketika sebuah situs dirayapi, alamat seperti `https://Example.com/` dan `https://example.com` harus dihitung sebagai halaman yang sama; itulah nomor telepon ternormalkan milik `Customer` dalam kostum lain, dan jika `equals` dan `hashCode` salah di sana, halaman yang sama dipindai dan dinilai dua kali. Dan karena Profitina mematuhi undang-undang pelindungan data pribadi Indonesia, UU PDP 27/2022, apa yang diperlihatkan sebuah kelas tentang dirinya dalam bentuk cetaknya adalah keputusan rancangan yang disengaja, seperti diperingatkan Subbab 6.10.

6.18 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya

Mengakhiri komentar Javadoc dengan `/` alih-alih `*/`. Komentar itu menelan segala sesuatu sampai `*/` berikutnya, mungkin seluruh metode, dan kompilator diam saja (Subbab 6.3.1). *Hindari dengan:* memakai penyunting berpewarnaan sintaksis, dan membangkitkan dokumentasi secara teratur.

Nama `@param` yang tidak cocok dengan parameternya. Setelah sebuah parameter diganti namanya, komentar masih menggambarkan nama lama. *Hindari dengan:* menjalankan `javadoc` dan memperlakukan *error* dan *warning*-nya sebagai *bug*; di Python, ganti nama di *docstring* pada saat yang sama.

Dokumentasi yang menggambarkan implementasi. "Memutari larik dan memakai `Math.pow`" tidak menolong pemanggil mana pun dan menjadi salah pada hari perulangannya ditulis ulang. *Hindari dengan:* menggambarkan tujuan, parameter, hasil, kegagalan, dan satuan.

Uji yang menghitung ulang jawaban dengan rumus yang sama. Uji itu menyetujui setiap *bug* dalam rumusnya. *Hindari dengan:* menulis nilai yang diharapkan sebagai literal yang dihitung dengan tangan.

Uji yang saling bergantung atau bergantung pada urutan. *Hindari dengan:* membangun objek segar di `@BeforeEach`/`setUp`, atau di dalam setiap uji.

Hanya menguji kasus yang menyenangkan. Kebanyakan *bug* bersarang di batas-batas dan di penanganan galat. *Hindari dengan:* mendaftar kelas-kelas ekuivalensi, menguji kedua sisi setiap batas, dan menguji setiap eksepsi yang dijanjikan dokumentasi.

Membandingkan `double` dengan `==`, atau menukar nilai yang diharapkan dan nilai sebenarnya. `0.1 + 0.2 == 0.3` bernilai salah, dan `assertEquals(rover.getAge(), 6)` berbohong ketika gagal. *Hindari dengan:* membandingkan dengan toleransi (Persamaan (6.2)) atau menyimpan uang dalam satuan bulat; letakkan nilai yang diharapkan lebih dulu.

`equals` tanpa `hashCode`, atau `equals(Customer other)`. Yang pertama membuat himpunan dan peta *hash* menganggap objek yang sama sebagai berbeda; yang kedua membebani (*overload*) alih-alih menimpa (*override*). *Hindari dengan:* menimpa kedua metode dari *field* yang sama, memakai parameter `Object`, menulis `@Override`, dan mengompilasi dengan `-Xlint:all`.

Memakai `==` untuk membandingkan string atau objek lain di Java. Operator itu menguji identitas, bukan isi. *Hindari dengan:* `a.equals(b)` untuk objek; `==` hanya untuk tipe primitif.

Membiarkan `toString` bawaan, atau memasukkan rahasia ke dalamnya. `PlainCustomer@1b6d3586` tidak memberi tahu apa-apa, dan kata sandi di dalam log adalah kebocoran. *Hindari dengan:* menimpa `toString`/`__repr__` secara sengaja di setiap kelas yang mungkin Anda cetak.

Melakukan *overloading* satu nama untuk argumen yang berbeda makna, atau mencoba *overloading* di Python. `describe(3L)` menjadi kilogram dan `describe('A')` menjadi 65 potong; `def` kedua di Python diam-diam menggantikan yang pertama. *Hindari dengan:* memberi makna yang berbeda nama yang berbeda; di Python pakailah argumen bawaan, `*args`, atau `singledispatch`.

Mengira `compareTo` mengembalikan -1, 0, atau 1. Hanya tandanya yang dijanjikan (Persamaan (6.5)). *Hindari dengan:* menguji `< 0`, `== 0`, `> 0`, dan mengimplementasikan `Comparable<Name>` yang generik.

Mendeklarasikan variabel dengan tipe kelas padahal antarmuka sudah cukup. `ArrayList<Order> orders` atau `TreeDB db` mengikat kode pada satu implementasi dan menghidupkan kembali pengujian penanda Subbab 6.13.1. *Hindari dengan:* mendeklarasikan `List<Order>`, `DBops`, `PriceRule`; sebutkan kelas konkret hanya di tempat objek dibuat.

Membulatkan uang di tempat yang salah. `normal * (percent / 100)` menghasilkan nol dalam bilangan bulat, dan `round` alih-alih pembulatan ke bawah mengubah harga satu rupiah. *Hindari dengan:* menuliskan aturan pembulatan di dokumentasi dan mengujinya dengan jumlah ganjil.

6.19 Ringkasan Bab

- Sebuah kelas mendaftar *field*, konstruktor, dan metodenya. Java mengizinkan beberapa konstruktor (*overload*); Python memiliki satu `__init__` dan memakai argumen bawaan atau *factory* `@classmethod`.
- **Komentar dokumentasi** menggambarkan apa yang dijanjikan sebuah kelas atau metode. Komentar Javadoc diawali `/**` dan memakai tag seperti `@param`, `@return`, dan `@throws` (**Tabel 6.1**); *docstring* Python mengikuti PEP 257, di buku ini dengan gaya Google. `javadoc` membangun dan memeriksa halaman HTML, `pydoc` menampilkan atau menulis dokumentasi sebuah modul, dan `doctest` menjalankan contoh-contoh di dalam *docstring* (**Gambar 6.1**). **Benih** yang tetap membuat bilangan acak dapat diulang, dan itulah yang dibutuhkan uji.
- **Uji unit** menjalankan satu unit pada masukan terpilih dan membandingkan hasil sebenarnya dengan hasil yang diharapkan yang dihitung sebelumnya, dalam tiga langkah: ***arrange, act, assert*** (**Gambar 6.2**). Program biasa dengan metode `check` dan status keluar bukan nol sudah merupakan rangkaian uji.
- **JUnit 6** (`@Test`, `@BeforeEach`, `@DisplayName`, `assertEquals`, `assertThrows`), **unittest**, dan **pytest** mengerjakan tugas yang sama (**Tabel 6.2**); `double` dibandingkan dengan toleransi (Persamaan (6.2)).
- Uji yang baik dipilih berdasarkan **kelas ekuivalensi dan batas** (**Tabel 6.3**) dan saling bebas. Menulis uji lebih dulu — **merah, hijau, *refactor*** (**Gambar 6.3**) — membuat Anda merancang dari sisi pemanggil.
- **Identitas** bukanlah **kesamaan** (**Gambar 6.4**). Timpalah `equals` dan `hashCode` bersama-sama, dari *field* yang sama, agar objek yang sama memiliki kode *hash* yang sama (Persamaan (6.3)). `toString` memberi objek bentuk cetaknya; Python memisahkan `__repr__` (untuk pengembang) dari `__str__` (untuk pengguna).
- ***Overloading*** memberi satu nama beberapa daftar parameter, dipilih saat kompilasi berdasarkan kecocokan persis, lalu pelebaran, lalu *boxing* (**Kode 6.18**). Python tidak dapat melakukan *overloading*; ia memakai argumen bawaan, `*args`, dan `singledispatch` (**Tabel 6.4**).
- **Antarmuka** mendaftar metode tanpa tubuh; sebuah kelas mengimplementasikannya (`implements`), antarmuka berfungsi sebagai tipe, dan pemanggilan dikirim ke kelas objek yang sebenarnya saat program berjalan. Menulis klien terhadap antarmuka menghapus pengujian penanda (**Gambar 6.5**); **metode default** berbagi kode; *up-casting* otomatis dan *down-casting* memerlukan *cast* yang diperiksa (**Gambar 6.6**).
- Pustaka Java dibangun di atas antarmuka seperti `Comparable`, `Iterable`, `List`, dan `ActionListener` (**Tabel 6.5**); `compareTo` hanya menjanjikan tanda (Persamaan (6.5)). Python menyatakan antarmuka sebagai **kelas dasar abstrak** atau ***protocol*** (**Tabel 6.6**).
- **Memprogramlah terhadap antarmuka, bukan terhadap implementasi.** Harga Profitina Laundry adalah lima kelas di balik `PriceRule` (**Gambar 6.7**, **Tabel 6.7**), dua di antaranya dekorator; nota hanya mengenal antarmuka, setiap aturan diuji sendirian, dan aturan baru adalah kelas baru dan tidak lebih.

6.20 Soal Latihan

Soal-soal latihan mengikuti urutan bab. Soal yang berupa program memiliki penyelesaian lengkap dalam Java dan Python di Subbab 6.21; soal lainnya dijawab dengan kata-kata. Cobalah setiap soal sebelum Anda melihat penyelesaiannya.

Soal Latihan 6.1 (mendokumentasikan metode). Profitina Laundry membutuhkan pembantu kecil, `kilosCharged(weightKg, minimumKg)`, yang mengembalikan banyaknya kilogram yang ditagihkan untuk sebuah pesanan: setiap kilogram yang sudah dimulai dihitung, dan tidak pernah kurang dari minimumnya; berat nol atau kurang ditolak. Tulislah metode itu dalam Java dengan komentar Javadoc lengkap (ringkasan, dua contoh, setiap `@param`, `@return`, `@throws`) dan dalam Python dengan *docstring* gaya Google yang contoh-contohnya berupa *doctest*, termasuk penolakannya. Jalankan `javadoc` dan `python3 -m doctest -v` dan pastikan keduanya berjalan bersih.

Soal Latihan 6.2 (benar atau salah?). Tentukan, beserta alasannya, apakah setiap pernyataan berikut benar. (a) Komentar yang diawali `/*` adalah komentar Javadoc. (b) Kompilator Java menyimpan komentar Javadoc di dalam berkas `.class` agar `javadoc` dapat membacanya kemudian. (c) *Docstring* Python dapat dibaca ketika program berjalan. (d) Dokumentasi sebuah metode harus menjelaskan perulangan mana yang dipakainya, agar pemanggil tahu seberapa cepat metode itu. (e) Jika `javadoc` tidak mencetak peringatan apa pun, dokumentasinya benar.

Soal Latihan 6.3 (satu konstruktor yang bekerja). Tulis ulang `BankAccount` dari **Kode 6.1** sebagai `ChainedAccount`, sehingga kedua konstruktor publiknya mendelegasikan dengan `this(...)` kepada satu konstruktor privat yang mengisi *field*-nya, dan beri kelas itu sebuah `toString`. Di Python, pertahankan satu `__init__(balance, overdraft)` dan tawarkan dua cara membuka rekening sebagai metode kelas `empty()` dan `with_deposit(amount)`. Tunjukkan bahwa rekening yang dibuka dengan setoran 0 mendapat cerukan 50.

Soal Latihan 6.4 (menguji `double`). Tulislah uji unit untuk `interestOn` / `interest_on` dari Subbab 6.2: nol tahun, satu tahun (yang harus sama dengan bunga sederhana), contoh dua tahun pada Persamaan (6.1), suku bunga nol, dan penolakan tahun negatif. Pakailah JUnit 6 di Java dan `unittest` di Python, dan bandingkan dengan toleransi. Untuk setiap uji, sebutkan kelas ekuivalensi atau batas mana yang diwakilinya.

Soal Latihan 6.5 (apa yang salah dengan uji-uji ini?). Setiap uji di bawah lulus hari ini. Jelaskan mengapa masing-masing tetap merupakan uji yang buruk, lalu tulis ulang.

```
(a) assertEquals(7000 * Math.max(3, (long) Math.ceil(3.2)), wash.priceOf(order));
(b) static Dog shared = new Dog("Rover", 5);
    @Test void birthday() { shared.incrementAge(); assertEquals(6, shared.getAge()); }
    @Test void age()      { assertEquals(5, shared.getAge()); }
(c) assertEquals(rover.getAge(), 6);
(d) def test_prices(): assert rules_work()
```

Soal Latihan 6.6 (uji lebih dulu: kartu loyalitas). Anggota Profitina Laundry memperoleh satu poin untuk setiap 10 000 rupiah penuh yang mereka bayar; 9 999 rupiah tidak menghasilkan apa-apa, dan jumlah negatif adalah galat. Dengan mengikuti siklus merah–hijau–*refactor* pada **Gambar 6.3**, tulislah uji-ujinya *lebih dulu* — pemeriksa biasa `MemberCardCheck` di Java dan berkas pytest `test_member_card.py` — dengan memilih batas-batasnya secara sengaja; jalankan dan saksikan keduanya gagal; lalu tulislah `MemberCard.pointsFor` dan `member_card.points_for`.

Soal Latihan 6.7 (kesamaan dengan dua *field*). Mesin-mesin cuci laundry dikenali dari nomor *dan* mereknya (mesin 3 dari dua pemasok yang berbeda adalah dua mesin). Tulislah kelas `Machine` dengan `equals`/`hashCode` (memakai `Objects.hash`) dan `__eq__`/`__hash__`, lalu tunjukkan bahwa ketika dua staf melaporkan mesin rusak yang sama, himpunan mesin yang perlu diperbaiki hanya memuatnya sekali.

Soal Latihan 6.8 (*record* dan kelas data). Tulislah `CustomerRecord` sebagai `record` Java dan sebagai `@dataclass(frozen=True)` Python dengan *field* `name` dan `phone`. Masukkan “Sari” dan

"Sari W." dengan telepon yang sama ke sebuah himpunan. Berapa elemen yang dimuat himpunan itu, dan mengapa `Customer` dari **Kode 6.15** memberi jawaban yang berbeda?

Soal Latihan 6.9 (teka-teki *overloading*). Tebaklah, *sebelum* menjalankannya, apa yang dicetak setiap baris. Java: tiga *overload* `describe(int)`, `describe(double)`, `describe(Object)`, dipanggil dengan sebuah `short`, `3f`, `Integer.valueOf(3)`, `"3"`, dan `'3'`. Python: fungsi `singledispatch` `describe` dengan versi untuk `int` dan `float`, dipanggil dengan `3`, `3.0`, `False`, `'3'`, `Fraction(3, 1)`, dan `2 ** 100`. Lalu jalankan program-programnya dan jelaskan setiap kejutan.

Soal Latihan 6.10 (menyesuaikan kelas dengan antarmuka). Pakailah metode `average` dari **Kode 6.22** dan **Kode 6.23** untuk menghitung rata-rata berat tiga pesanan, *tanpa mengubah* `Order` dan tanpa mengubah `average`.

Soal Latihan 6.11 (aturan harga baru). Profitina Laundry memperkenalkan potongan borongan: pesanan yang *lebih berat dari* suatu ambang (tidak termasuk ambangnya) mendapat potongan sekian persen dari harga yang dihitung aturan lain. Tulislah `BulkDiscountRule` sebagai dekorator bergaya `MemberDiscountRule`, tolak persentase di luar 0..100, dan ujilah dengan batas 9 kg, tepat 10 kg, 10,01 kg, dan 12 kg untuk ambang 10 kg, potongan 20%, pada cuci lipat. Hitung dulu harga-harga yang diharapkan dengan tangan.

Soal Latihan 6.12 (apakah urutan dekorator penting?). Seorang anggota yang memesan layanan ekspres dapat dihargai sebagai `MemberDiscountRule(ExpressRule(base))` atau sebagai `ExpressRule(MemberDiscountRule(base))`. Dalam aritmetika eksak keduanya memberikan $1.5 \times 0.9 = 1.35$ kali harga dasar. Tulislah program yang mencoba setiap harga dasar dari 21 000 sampai 21 020 rupiah dengan sebuah `FlatRule` dan mencetak harga dasar yang membuat kedua urutan itu tidak sepakat. Jelaskan hasilnya, dan katakan apa yang sebaiknya dilakukan laundry.

Soal Latihan 6.13 (sebuah Comparator). Urutkan list pesanan dari yang terberat, dan pesanan yang sama beratnya menurut nomor tiket, dengan implementasi antarmuka `java.util.Comparator<Order>` di Java dan fungsi `key=` di Python. Mengapa `Comparator` lebih tepat di sini daripada membuat `Order` mengimplementasikan `Comparable`?

6.21 Penyelesaian Soal Latihan

Penyelesaian 6.1. Pembantu ini adalah Persamaan (6.6) tanpa tarif. Komentarnya menyatakan kontrak — kilogram yang sudah dimulai, minimum, penolakan — dan tidak menyebut `Math.ceil`, yang merupakan rincian implementasi. **Kode 6.48** adalah versi Java-nya.

Kode 6.48 `KilosCharged.java`: pembantu yang terdokumentasi penuh, dengan contoh-contoh terdokumentasinya di `main`.

```java
/**
 * Works out how many kilograms a per-kilo laundry order is charged for.
 */
public class KilosCharged {
    private KilosCharged() {
        // hanya metode statis: objek kelas ini tidak diperlukan
    }

    /**
     * Returns the number of kilograms to charge: every started kilogram
     * counts, and never fewer than the minimum.
     *
     * <p>For example, {@code kilosCharged(3.2, 3)} is 4 and
     * {@code kilosCharged(1.5, 3)} is 3.</p>
     *
     * @param weightKg  the weight on the scale, in kilograms; must be positive
     * @param minimumKg the smallest number of kilograms ever charged
     * @return the number of kilograms to charge
     * @throws IllegalArgumentException if {@code weightKg} is zero or negative
     */
    public static long kilosCharged(double weightKg, int minimumKg) {
        if (weightKg <= 0) {
            throw new IllegalArgumentException("weight must be positive: " + weightKg);
        }
        return Math.max(minimumKg, (long) Math.ceil(weightKg));
    }

    /**
     * Prints the two documented examples.
     *
     * @param args not used
     */
    public static void main(String[] args) {
        System.out.println(kilosCharged(3.2, 3));
        System.out.println(kilosCharged(1.5, 3));
    }
}
```

```
4
3
```

`javadoc -d docs KilosCharged.java` berjalan tanpa satu peringatan pun. Konstruktor privat itu penting di sini: kelas ini hanya memiliki metode statis, dan tanpanya kompilator akan menambahkan konstruktor bawaan publik yang dilaporkan `javadoc` sebagai tidak terdokumentasi. Versi Python, **Kode 6.49**, menjadikan contoh-contohnya *doctest*; contoh ketiga memperlihatkan cara *doctest* mengharapkan eksepsi: baris `Traceback`, satu baris titik-titik untuk bagian yang berubah-ubah, dan baris terakhir pesan galatnya.

Kode 6.49 `kilos_charged.py`: *docstring* gaya Google yang contoh-contohnya berupa *doctest*, salah satunya eksepsi yang diharapkan.

```
"""How many kilograms a per-kilo laundry order is charged for."""
import math


def kilos_charged(weight_kg, minimum_kg):
    """Return the number of kilograms to charge.

    Every started kilogram counts, and never fewer than the minimum.

    Args:
        weight_kg: the weight on the scale in kilograms; must be positive.
        minimum_kg: the smallest number of kilograms ever charged.

    Returns:
        The number of kilograms to charge, as an int.

    Raises:
        ValueError: if weight_kg is zero or negative.

    >>> kilos_charged(3.2, 3)
    4
    >>> kilos_charged(1.5, 3)
    3
    >>> kilos_charged(0, 3)
    Traceback (most recent call last):
        ...
    ValueError: weight must be positive: 0
    """
    if weight_kg <= 0:
        raise ValueError(f"weight must be positive: {weight_kg}")
    return max(minimum_kg, math.ceil(weight_kg))


if __name__ == "__main__":
    print(kilos_charged(3.2, 3))
    print(kilos_charged(1.5, 3))
```

```
3 tests in 2 items.
3 passed and 0 failed.
Test passed.
```

Penyelesaian 6.2. (a) Salah: komentar Javadoc diawali `/**`; `/*` memulai komentar blok biasa. (b) Salah: kompilator membuang semua komentar, dan `javadoc` membaca berkas-berkas `.java`. (c) Benar: *docstring* disimpan di atribut `__doc__`, dan `help()` mencetaknya. (d) Salah: dokumentasi menyatakan apa yang dijanjikan metode — parameter, hasil, kegagalan, satuan, dan jaminan kecepatan apa pun yang boleh diandalkan pemanggil — bukan perulangan mana yang dipakainya. (e) Salah: `javadoc` memeriksa *bentuk*, bukan *kebenaran*; "mengembalikan saldo" pada metode yang mengembalikan cerukan lolos tanpa suara. Hanya pembaca, atau sebuah uji, yang menangkapnya.

Penyelesaian 6.3. **Kode 6.50** menaruh seluruh pengisian *field* di satu konstruktor privat. `this(...)` harus menjadi pernyataan pertama sebuah konstruktor, dan konstruktor privat boleh menerima parameter yang tidak boleh dipadukan sembarangan oleh pemanggil dari luar.

Kode 6.50 `ChainedAccount.java`: dua konstruktor publik yang mendelegasikan kepada satu konstruktor privat.

```java
/**
 * BankAccount with constructor chaining: both public constructors
 * delegate to one private constructor that does all the work.
 */
public class ChainedAccount {
    private final double balance;
    private final int overdraft;

    /**
     * Opens an account with an opening deposit and an overdraft of 50.
     *
     * @param openingAmount the money paid in on the first day
     */
    public ChainedAccount(double openingAmount) {
        this(openingAmount, 50);          // wajib menjadi pernyataan pertama
    }

    /**
     * Opens an empty account with an overdraft of 10.
     */
    public ChainedAccount() {
        this(0, 10);
    }

    private ChainedAccount(double balance, int overdraft) {
        this.balance = balance;           // satu-satunya tempat field diisi
        this.overdraft = overdraft;
    }

    @Override
    public String toString() {
        return "ChainedAccount[balance=" + balance + ", overdraft=" + overdraft + "]";
    }

    public static void main(String[] args) {
        System.out.println(new ChainedAccount());
        System.out.println(new ChainedAccount(25));
        System.out.println(new ChainedAccount(0));
    }
}
```

```
ChainedAccount[balance=0.0, overdraft=10]
ChainedAccount[balance=25.0, overdraft=50]
ChainedAccount[balance=0.0, overdraft=50]
```

Di Python (**Kode 6.51**), sebuah *metode kelas* menerima kelas itu sendiri sebagai `cls`, dan `cls(0, 10)` memanggil `__init__`. Kedua metode kelas itu adalah konstruktor *bernama*, yang lebih banyak berbicara kepada pembaca daripada `BankAccount()`.

Kode 6.51 `chained_account.py`: konstruktor alternatif yang ditulis sebagai metode kelas.

```python
"""Alternative constructors in Python: class methods that call the one __init__."""


class ChainedAccount:
    def __init__(self, balance, overdraft):
        self._balance = float(balance)      # satu-satunya tempat atribut diisi
        self._overdraft = overdraft

    @classmethod
    def empty(cls):
        """Open an empty account with an overdraft of 10."""
        return cls(0, 10)

    @classmethod
    def with_deposit(cls, opening_amount):
        """Open an account with a deposit and an overdraft of 50."""
        return cls(opening_amount, 50)

    def __repr__(self):
        return f"ChainedAccount[balance={self._balance}, overdraft={self._overdraft}]"


print(ChainedAccount.empty())
print(ChainedAccount.with_deposit(25))
print(ChainedAccount.with_deposit(0))
```

```
ChainedAccount[balance=0.0, overdraft=10]
ChainedAccount[balance=25.0, overdraft=50]
ChainedAccount[balance=0.0, overdraft=50]
```

Baris ketiga kedua keluaran menjawab pertanyaan terakhir: setoran nol tetaplah setoran, jadi cerukannya 50.

Penyelesaian 6.4. Kelas-kelas ekuivalensinya adalah: tanpa waktu sama sekali (nol tahun), kasus nyata paling sederhana (satu tahun, ketika bunga majemuk dan bunga sederhana berimpit: $1\,000 \times 0.06 = 60$), kasus umum (dua tahun, Persamaan (6.1)), suku bunga yang merosot (nol), dan kelas yang tidak sah (tahun negatif). **Kode 6.52** adalah kelas JUnit 6-nya; setiap `assertEquals` memiliki argumen ketiga, toleransi pada Persamaan (6.2).

Kode 6.52 `BankAccountTest.java`: uji JUnit 6 untuk `interestOn`, membandingkan `double` dengan sebuah delta.

```
import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertThrows;

import org.junit.jupiter.api.Test;

/**
 * JUnit 5 tests for BankAccount.interestOn, comparing doubles with a delta.
 */
class BankAccountTest {
    private final BankAccount account = new BankAccount();

    @Test
    void zeroYearsEarnNothing() {
        assertEquals(0.0, account.interestOn(1_000, 0, 0.06), 1e-9);
    }

    @Test
    void oneYearIsSimpleInterest() {
        assertEquals(60.0, account.interestOn(1_000, 1, 0.06), 1e-9);
    }

    @Test
    void twoYearsCompound() {
        assertEquals(123_600.0, account.interestOn(1_000_000, 2, 0.06), 1e-6);
    }

    @Test
    void zeroRateEarnsNothing() {
        assertEquals(0.0, account.interestOn(1_000, 5, 0.0), 1e-9);
    }

    @Test
    void negativeYearsAreRejected() {
        assertThrows(IllegalArgumentException.class, () -> account.interestOn(1_000, -1,
0.06));
    }
}
```

Toleransi untuk kasus dua tahun adalah 10^{-6}, bukan 10^{-9}, karena bilangannya sejuta kali lebih besar, demikian pula derau pembulatannya. **Kode 6.53** adalah versi `unittest`-nya.

Kode 6.53 `test_bank_account.py`: uji `unittest` untuk `interest_on` dengan `assertAlmostEqual`.

```python
"""unittest tests for BankAccount.interest_on, comparing floats with a tolerance."""
import unittest

from bank_account import BankAccount


class InterestTest(unittest.TestCase):
    def setUp(self):
        self.account = BankAccount()

    def test_zero_years_earn_nothing(self):
        self.assertAlmostEqual(0.0, self.account.interest_on(1_000, 0, 0.06), delta=1e-9)

    def test_one_year_is_simple_interest(self):
        self.assertAlmostEqual(60.0, self.account.interest_on(1_000, 1, 0.06), delta=1e-9)

    def test_two_years_compound(self):
        self.assertAlmostEqual(123_600.0, self.account.interest_on(1_000_000, 2, 0.06),
                               delta=1e-6)

    def test_zero_rate_earns_nothing(self):
        self.assertAlmostEqual(0.0, self.account.interest_on(1_000, 5, 0.0), delta=1e-9)

    def test_negative_years_are_rejected(self):
        with self.assertRaises(ValueError):
            self.account.interest_on(1_000, -1, 0.06)


if __name__ == "__main__":
    unittest.main()
```

```
test_negative_years_are_rejected
(test_bank_account.InterestTest.test_negative_years_are_rejected) ... ok
test_one_year_is_simple_interest
(test_bank_account.InterestTest.test_one_year_is_simple_interest) ... ok
test_two_years_compound (test_bank_account.InterestTest.test_two_years_compound) ... ok
test_zero_rate_earns_nothing (test_bank_account.InterestTest.test_zero_rate_earns_nothing)
... ok
test_zero_years_earn_nothing (test_bank_account.InterestTest.test_zero_years_earn_nothing)
... ok

----------------------------------------------------------------------
Ran 5 tests in 0.000s

OK
```

Penyelesaian 6.5. (a) Nilai yang diharapkan memakai rumus milik kode itu sendiri, sehingga *bug* dalam rumus (`Math.floor` alih-alih `Math.ceil`) muncul di kedua sisi. Tulis literalnya, `28_000`. (b) Kedua uji berbagi satu anjing, sehingga `age` gagal setiap kali `birthday` kebetulan berjalan lebih dulu. Bangun anjingnya di `@BeforeEach`. (c) Nilai yang diharapkan dan nilai sebenarnya tertukar, sehingga kegagalan akan melaporkan “expected: <7> but was: <6>”. Tulis `assertEquals(6, rover.getAge())`. (d) Satu uji memeriksa segalanya melalui pembantu yang mengembalikan *boolean*, sehingga kegagalan hanya dapat berkata `assert False`. Tulislah satu uji per perilaku, masing-masing memastikan nilai yang konkret.

Penyelesaian 6.6. Uji-ujinya datang lebih dulu, dan memilihnya adalah pekerjaan yang sesungguhnya: nol, nilai tepat di bawah poin pertama, tepat poin pertama, tepat di atasnya, satu pesanan yang lazim, dan penolakan. **Kode 6.54** adalah pemeriksa Java-nya; sebelum `MemberCard` ada, pemeriksa itu bahkan tidak dapat dikompilasi, dan itulah “merah” yang pertama.

Kode 6.54 `MemberCardCheck.java`: uji-uji kartu loyalitas, ditulis sebelum kode yang diujinya.

```
/**
 * The tests for MemberCard, written BEFORE MemberCard itself.
 */
public class MemberCardCheck {
    private static int passed = 0;
    private static int failed = 0;

    static void check(String what, long expected, long actual) {
        if (expected == actual) {
            passed++;
            System.out.println("PASS  " + what);
        } else {
            failed++;
            System.out.println("FAIL  " + what + ": expected " + expected + " but was " +
actual);
        }
    }

    public static void main(String[] args) {
        check("nothing paid, no points", 0, MemberCard.pointsFor(0));
        check("just below one point", 0, MemberCard.pointsFor(9_999));
        check("exactly one point", 1, MemberCard.pointsFor(10_000));
        check("just above one point", 1, MemberCard.pointsFor(10_001));
        check("a typical order", 6, MemberCard.pointsFor(63_000));
        try {
            MemberCard.pointsFor(-1);
            failed++;
            System.out.println("FAIL  a negative amount is refused: no exception");
        } catch (IllegalArgumentException e) {
            passed++;
            System.out.println("PASS  a negative amount is refused");
        }
        System.out.println(passed + " passed, " + failed + " failed");
        if (failed > 0) {
            System.exit(1);
        }
    }
}
```

Kode paling sederhana yang meluluskannya adalah satu pembagian bilangan bulat — “hijau” — dan **Kode 6.55** adalah kode itu, lengkap dengan dokumentasinya. Tidak ada yang perlu dirapikan.

Kode 6.55 `MemberCard.java`: aturan kartu loyalitas, ditulis untuk meluluskan uji-ujinya.

```
/**
 * Loyalty points for Profitina Laundry members.
 */
public class MemberCard {
    /** Rupiah needed for one point. */
    public static final long RUPIAH_PER_POINT = 10_000;

    /**
     * Returns the points earned for a payment: one point per full 10.000 rupiah.
     *
     * @param amount the amount paid, in rupiah
     * @return the number of points earned
     * @throws IllegalArgumentException if {@code amount} is negative
     */
    public static long pointsFor(long amount) {
        if (amount < 0) {
            throw new IllegalArgumentException("amount must not be negative: " + amount);
        }
        return amount / RUPIAH_PER_POINT;   // pembagian bilangan bulat membuang bagian
yang belum genap
    }
}
```

```
PASS  nothing paid, no points
PASS  just below one point
PASS  exactly one point
PASS  just above one point
PASS  a typical order
PASS  a negative amount is refused
6 passed, 0 failed
```

Di Python, berkas ujinya (**Kode 6.56**) memakai `parametrize` untuk kelima baris batas; dijalankan sebelum `member_card.py` ada, pytest berhenti dengan `ModuleNotFoundError: No module named 'member_card'` — merah. **Kode 6.57** adalah aturan satu baris yang membuatnya hijau.

Kode 6.56 `test_member_card.py`: versi pytest dari uji-uji kartu loyalitas.

```python
"""The tests for member_card, written BEFORE member_card itself."""
import pytest

from member_card import points_for


@pytest.mark.parametrize("amount, points", [
    (0, 0),             # tidak ada pembayaran
    (9_999, 0),         # sedikit di bawah satu poin
    (10_000, 1),        # tepat satu poin
    (10_001, 1),        # sedikit di atas
    (63_000, 6),        # pesanan yang lazim
])
def test_points(amount, points):
    assert points_for(amount) == points


def test_negative_amount_is_refused():
    with pytest.raises(ValueError):
        points_for(-1)
```

Kode 6.57 `member_card.py`: aturannya, dengan pembagian yang membulatkan ke bawah.

```python
"""Loyalty points for Profitina Laundry members."""

RUPIAH_PER_POINT = 10_000


def points_for(amount):
    """Return the points earned for a payment: one point per full 10.000 rupiah.

    Raises:
        ValueError: if amount is negative.
    """
    if amount < 0:
        raise ValueError(f"amount must not be negative: {amount}")
    return amount // RUPIAH_PER_POINT       # pembagian lantai membuang bagian yang belum genap
```

```
test_member_card.py::test_points[0-0] PASSED                              [ 16%]
test_member_card.py::test_points[9999-0] PASSED                           [ 33%]
test_member_card.py::test_points[10000-1] PASSED                          [ 50%]
test_member_card.py::test_points[10001-1] PASSED                          [ 66%]
test_member_card.py::test_points[63000-6] PASSED                          [ 83%]
test_member_card.py::test_negative_amount_is_refused PASSED               [100%]
============================== 6 passed in 0.01s ===============================
```

Penyelesaian 6.7. Kesamaan kini melibatkan dua *field*, sehingga `equals` dan `hashCode` sama-sama memakai keduanya (Persamaan (6.3)). `Objects.hash(number, brand)` menggabungkan

kedua kode *hash*; di Python, meng-*hash* *tuple* `(number, brand)` melakukan hal yang sama. **Kode 6.58** dan **Kode 6.59** adalah kedua kelas itu beserta peragaan kecil.

Kode 6.58 `Machine.java`: kesamaan dan kode *hash* yang dibangun dari dua *field*.

```
import java.util.HashSet;
import java.util.Objects;
import java.util.Set;

/**
 * A washing machine, identified by its number and brand.
 */
public final class Machine {
    private final int number;
    private final String brand;

    public Machine(int number, String brand) {
        this.number = number;
        this.brand = brand;
    }

    @Override
    public boolean equals(Object other) {
        if (this == other) {
            return true;
        }
        if (!(other instanceof Machine)) {
            return false;
        }
        Machine that = (Machine) other;
        return number == that.number && brand.equals(that.brand);
    }

    @Override
    public int hashCode() {
        return Objects.hash(number, brand);   // memakai tepat field yang dipakai equals
    }

    @Override
    public String toString() {
        return "Machine #" + number + " (" + brand + ")";
    }
```

Kode 6.58 (Lanjutan) *Machine.java: kesamaan dan kode hash yang dibangun dari dua field.*

```
    public static void main(String[] args) {
        Set<Machine> broken = new HashSet<>();
        broken.add(new Machine(3, "Tosaki"));
        broken.add(new Machine(3, "Tosaki"));   // dilaporkan dua kali oleh dua karyawan
        System.out.println("machines to repair: " + broken.size());
        System.out.println("is #3 broken? " + broken.contains(new Machine(3, "Tosaki")));
        System.out.println(broken);
    }
}
```

```
machines to repair: 1
is #3 broken? true
[Machine #3 (Tosaki)]
```

Kode 6.59 `machine.py`: `__eq__` dan `__hash__` yang dibangun dari *tuple* dua *field*.

```
"""A washing machine, identified by its number and brand."""


class Machine:
    def __init__(self, number, brand):
        self._number = number
        self._brand = brand

    def __eq__(self, other):
        if not isinstance(other, Machine):
            return NotImplemented
        return (self._number, self._brand) == (other._number, other._brand)

    def __hash__(self):
        return hash((self._number, self._brand))   # tepat field yang dipakai __eq__

    def __repr__(self):
        return f"Machine #{self._number} ({self._brand})"


broken = {Machine(3, "Tosaki"), Machine(3, "Tosaki")}   # dilaporkan dua kali
print("machines to repair:", len(broken))
print("is #3 broken?", Machine(3, "Tosaki") in broken)
print(broken)
```

```
machines to repair: 1
is #3 broken? True
{Machine #3 (Tosaki)}
```

Penyelesaian 6.8. **Kode 6.60** dan **Kode 6.61** memperlihatkan bahwa metode-metode yang dibangkitkan membandingkan *setiap* komponen. “Sari” dan “Sari W.” berbeda namanya, sehingga himpunan itu memuat dua *record*.

Kode 6.60 `CustomerRecordDemo.java`: sebuah *record* membandingkan semua komponennya.

```
import java.util.HashSet;
import java.util.Set;

/**
 * A record gets equals, hashCode and toString written by the compiler.
 */
public class CustomerRecordDemo {
    record CustomerRecord(String name, String phone) {
    }

    public static void main(String[] args) {
        CustomerRecord a = new CustomerRecord("Sari", "08125550101");
        CustomerRecord b = new CustomerRecord("Sari", "08125550101");
        CustomerRecord c = new CustomerRecord("Sari W.", "08125550101");
        System.out.println(a);
        System.out.println("a.equals(b): " + a.equals(b));
        System.out.println("a.equals(c): " + a.equals(c));   // setiap komponen
diperhitungkan
        Set<CustomerRecord> set = new HashSet<>();
        set.add(a);
        set.add(b);
        set.add(c);
        System.out.println("set size   : " + set.size());
    }
}
```

```
CustomerRecord[name=Sari, phone=08125550101]
a.equals(b): true
a.equals(c): false
set size   : 2
```

Kode 6.61 `customer_record.py`: *dataclass* beku membandingkan semua *field*-nya.

```
"""A frozen dataclass gets __eq__, __hash__ and __repr__ written for it."""
from dataclasses import dataclass


@dataclass(frozen=True)
class CustomerRecord:
    name: str
    phone: str


a = CustomerRecord("Sari", "08125550101")
b = CustomerRecord("Sari", "08125550101")
c = CustomerRecord("Sari W.", "08125550101")
print(a)
print("a == b  :", a == b)
print("a == c  :", a == c)           # setiap field diperhitungkan
print("set size:", len({a, b, c}))
```

```
CustomerRecord(name='Sari', phone='08125550101')
a == b  : True
a == c  : False
set size: 2
```

`Customer` pada **Kode 6.15** memutuskan bahwa hanya telepon yang sudah dinormalkan yang mengenali seorang pelanggan, sehingga kedua entri yang sama itu akan dihitung sebagai satu. Tidak ada jawaban yang "benar" secara umum; kelaslah yang memutuskan arti kesamaan, dan *record* atau kelas data hanya tepat jika keputusannya adalah "semua *field*".

Penyelesaian 6.9. **Kode 6.62** dan **Kode 6.63** adalah teka-tekinya.

Kode 6.62 `OverloadPuzzle.java`: pemilihan *overload* dengan pelebaran dan *boxing*.

```
/**
 * Which describe runs? Predict each line before you run the program.
 */
public class OverloadPuzzle {
    static String describe(int pieces) {
        return "int";
    }

    static String describe(double kilos) {
        return "double";
    }

    static String describe(Object anything) {
        return "Object";
    }

    public static void main(String[] args) {
        short s = 3;
        System.out.println("describe(s)                  : " + describe(s));
        System.out.println("describe(3f)                 : " + describe(3f));
        System.out.println("describe(Integer.valueOf(3)) : " +
describe(Integer.valueOf(3)));
        System.out.println("describe(\"3\")                : " + describe("3"));
        System.out.println("describe('3')                : " + describe('3'));
    }
}
```

```
describe(s)                  : int
describe(3f)                 : double
describe(Integer.valueOf(3)) : Object
describe("3")                : Object
describe('3')                : int
```

`short` melebar menjadi `int`, dan `float` menjadi `double`. `Integer.valueOf(3)` sudah berupa objek, dan fase pertama, yang mengizinkan pelebaran tetapi tidak *unboxing*, langsung menemukan `describe(Object)`. `String` hanya dapat menuju `Object`, dan `'3'` adalah kode karakter 51, yang melebar menjadi `int`.

Kode 6.63 `overload_puzzle.py`: `singledispatch` dengan `bool`, `Fraction`, dan bilangan bulat raksasa.

```
"""Which describe runs? Predict each line before you run the program."""
from fractions import Fraction
from functools import singledispatch


@singledispatch
def describe(value):
    return "object"


@describe.register
def _(value: int):
    return "int"


@describe.register
def _(value: float):
    return "float"


print("describe(3)            :", describe(3))
print("describe(3.0)          :", describe(3.0))
print("describe(False)        :", describe(False))
print("describe('3')          :", describe("3"))
print("describe(Fraction(3,1)):", describe(Fraction(3, 1)))
print("describe(2 ** 100)     :", describe(2 ** 100))
```

```
describe(3)            : int
describe(3.0)          : float
describe(False)        : int
describe('3')          : object
describe(Fraction(3,1)): object
describe(2 ** 100)     : int
```

`False` adalah `bool`, subkelas `int`, sehingga ia mengambil versi `int`. `Fraction(3, 1)` sama dengan 3 tetapi bukan `int` dan bukan `float`. Dan `2 ** 100` tetap `int`, karena bilangan bulat Python tidak memiliki batas ukuran. `singledispatch` hanya melihat kelas argumen pertama, tidak pernah nilainya.

Penyelesaian 6.10. Kuncinya adalah sebuah **adapter** (penyesuai): kelas kecil yang membungkus sebuah `Order` dan menyajikannya melalui antarmuka yang dipahami `average`. Di Java (**Kode 6.64**), `LoadWeight` mengimplementasikan `Measurable` dan mengembalikan berat pesanan; `Order` dan `average` tetap persis seperti semula.

Kode 6.64 `AverageLoad.java`: sebuah adapter membuat `Order` dapat ikut serta dalam `average`.

```java
/**
 * Lets an Order take part in MeasurableDemo.average without changing Order.
 */
public class AverageLoad {
    /** An adapter: it wraps an Order and presents it as a Measurable. */
    static class LoadWeight implements Measurable {
        private final Order order;

        LoadWeight(Order order) {
            this.order = order;
        }

        @Override
        public double getMeasure() {
            return order.getWeightKg();
        }
    }

    public static void main(String[] args) {
        Measurable[] loads = {
            new LoadWeight(new Order("A001", "Sari", 2.0, 0, false)),
            new LoadWeight(new Order("A002", "Budi", 3.2, 0, true)),
            new LoadWeight(new Order("A005", "Sari", 5.5, 0, false))
        };
        System.out.printf("average load: %.2f kg%n", MeasurableDemo.average(loads));
    }
}
```

```
average load: 3.57 kg
```

Menurut Persamaan (6.4), $(2.0 + 3.2 + 5.5)/3 = 10.7/3 \approx 3.57$. Di Python (**Kode 6.65**), adapter itu hanya membutuhkan metode `get_measure`, karena `Measurable` adalah *protocol* dan cocok berdasarkan bentuk.

Kode 6.65 `average_load.py`: adapter yang sama dalam Python.

```python
"""Averaging order weights with the Measurable protocol: a small adapter is enough."""
from measurable_demo import average
from order import Order


class LoadWeight:
    """Wraps an Order and offers get_measure(), so it satisfies Measurable."""

    def __init__(self, order):
        self._order = order

    def get_measure(self):
        return self._order.weight_kg


loads = [LoadWeight(Order("A001", "Sari", 2.0, 0, False)),
         LoadWeight(Order("A002", "Budi", 3.2, 0, True)),
         LoadWeight(Order("A005", "Sari", 5.5, 0, False))]
print(f"average load: {average(loads):.2f} kg")
```

```
average load: 3.57 kg
```

Adapter adalah pola lain yang dinamai dalam [15], berguna setiap kali kelas yang sebaiknya tidak diubah harus cocok dengan antarmuka yang tidak dirancang untuknya.

Penyelesaian 6.11. Harga-harga yang diharapkan lebih dulu, menurut Persamaan (6.6) dan potongan yang ditulis seperti Persamaan (6.8): 9 kg tidak melewati ambang, $9 \times 7\,000 = 63\,000$. Tepat 10 kg tidak *lebih berat dari* ambang, jadi 70 000. 10,01 kg ditagih sebagai 11 kg, 77 000, dan

potongan 20% menghasilkan $77\,000 - 15\,400 = 61\,600$. 12 kg menjadi $84\,000 - 16\,800 = 67\,200$. **Kode 6.66** adalah aturannya. Perhatikan bahwa aturan itu menguji *berat* pesanan, bukan harganya, sehingga ia menanyai pesanan melalui *getter*-nya sementara harganya datang dari aturan yang dibungkus. **Kode 6.67** memeriksa keempat batas dan penolakannya.

Kode 6.66 `BulkDiscountRule.java`: dekorator yang memberi potongan untuk pesanan berat.

```
/**
 * Gives a percentage off orders heavier than a threshold.
 */
public class BulkDiscountRule implements PriceRule {
    private final PriceRule base;
    private final double thresholdKg;
    private final int percent;

    /**
     * Wraps another rule.
     *
     * @param base        the rule that prices the order before the discount
     * @param thresholdKg orders strictly heavier than this get the discount
     * @param percent     the discount, from 0 to 100
     * @throws IllegalArgumentException if {@code percent} is outside 0..100
     */
    public BulkDiscountRule(PriceRule base, double thresholdKg, int percent) {
        if (percent < 0 || percent > 100) {
            throw new IllegalArgumentException("percent must be 0..100: " + percent);
        }
        this.base = base;
        this.thresholdKg = thresholdKg;
        this.percent = percent;
    }

    @Override
    public long priceOf(Order order) {
        long normal = base.priceOf(order);
        if (order.getWeightKg() <= thresholdKg) {
            return normal;
        }
        return normal - normal * percent / 100;
    }

    @Override
    public String name() {
        return base.name() + " + bulk " + percent + "%";
    }
}
```

Kode 6.67 `BulkDiscountCheck.java`: uji-uji batas untuk potongan borongan.

```java
/**
 * Boundary tests for BulkDiscountRule.
 */
public class BulkDiscountCheck {
    private static int passed = 0;
    private static int failed = 0;

    static void check(String what, long expected, long actual) {
        if (expected == actual) {
            passed++;
            System.out.println("PASS  " + what);
        } else {
            failed++;
            System.out.println("FAIL  " + what + ": expected " + expected + " but was " +
actual);
        }
    }

    static Order kg(double weight) {
        return new Order("B1", "Test", weight, 0, false);
    }

    public static void main(String[] args) {
        PriceRule bulk = new BulkDiscountRule(new PerKiloRule("wash & fold", 7_000, 3), 10,
20);
        check("9 kg: no discount", 63_000, bulk.priceOf(kg(9)));
        check("exactly 10 kg: no discount", 70_000, bulk.priceOf(kg(10)));
        check("10.01 kg: 11 kg charged, 20% off", 61_600, bulk.priceOf(kg(10.01)));
        check("12 kg: 20% off", 67_200, bulk.priceOf(kg(12)));
        try {
            new BulkDiscountRule(new FlatRule("flat", 1), 10, -5);
            failed++;
            System.out.println("FAIL  a negative percent is refused: no exception");
        } catch (IllegalArgumentException e) {
            passed++;
            System.out.println("PASS  a negative percent is refused");
        }
        System.out.println(passed + " passed, " + failed + " failed");
        if (failed > 0) {
            System.exit(1);
        }
    }
}
```

```
PASS  9 kg: no discount
PASS  exactly 10 kg: no discount
PASS  10.01 kg: 11 kg charged, 20% off
PASS  12 kg: 20% off
PASS  a negative percent is refused
5 passed, 0 failed
```

Aturan Python (**Kode 6.68**) adalah subkelas ABC `PriceRule`, dan uji-ujinya (**Kode 6.69**) berupa satu fungsi berparameter.

Kode 6.68 `bulk_discount.py`: potongan borongan dalam Python.

```
"""A price rule that gives a percentage off orders heavier than a threshold."""
from price_rule import PriceRule


class BulkDiscountRule(PriceRule):
    """Orders strictly heavier than threshold_kg get percent off."""

    def __init__(self, base, threshold_kg, percent):
        if not 0 <= percent <= 100:
            raise ValueError(f"percent must be 0..100: {percent}")
        self._base = base
        self._threshold_kg = threshold_kg
        self._percent = percent

    def price_of(self, order):
        normal = self._base.price_of(order)
        if order.weight_kg <= self._threshold_kg:
            return normal
        return normal - normal * self._percent // 100

    def name(self):
        return f"{self._base.name()} + bulk {self._percent}%"
```

Kode 6.69 `test_bulk_discount.py`: uji-uji batas pytest untuk potongan borongan.

```
"""Boundary tests for BulkDiscountRule."""
import pytest

from bulk_discount import BulkDiscountRule
from order import Order
from rules import FlatRule, PerKiloRule

BULK = BulkDiscountRule(PerKiloRule("wash & fold", 7_000, 3), 10, 20)


@pytest.mark.parametrize("kg, expected", [
    (9, 63_000),         # di bawah ambang
    (10, 70_000),        # tepat di ambang: tanpa diskon
    (10.01, 61_600),     # sedikit melewatinya: ditagih 11 kg, potongan 20%
    (12, 67_200),
])
def test_bulk_boundaries(kg, expected):
    assert BULK.price_of(Order("B1", "Test", kg, 0, False)) == expected


def test_negative_percent_is_refused():
    with pytest.raises(ValueError):
        BulkDiscountRule(FlatRule("flat", 1), 10, -5)
```

```
test_bulk_discount.py::test_bulk_boundaries[9-63000] PASSED              [ 20%]
test_bulk_discount.py::test_bulk_boundaries[10-70000] PASSED             [ 40%]
test_bulk_discount.py::test_bulk_boundaries[10.01-61600] PASSED          [ 60%]
test_bulk_discount.py::test_bulk_boundaries[12-67200] PASSED             [ 80%]
test_bulk_discount.py::test_negative_percent_is_refused PASSED           [100%]
============================== 5 passed in 0.01s ===============================
```

Tidak ada hal lain yang berubah — bukan `Order`, bukan `PriceRule`, bukan aturan-aturan lain maupun uji-ujinya — dan itulah Subbab 6.16.7 dalam praktik.

Penyelesaian 6.12. Kode 6.70 mencoba ke-21 harga dasar itu.

Kode 6.70 `DecoratorOrder.java`: menumpuk dua dekorator yang sama dalam kedua urutan.

```java
/**
 * Does the order in which ExpressRule and MemberDiscountRule are stacked matter?
 */
public class DecoratorOrder {
    public static void main(String[] args) {
        Order member = new Order("D1", "Test", 0, 1, true);
        int differences = 0;
        for (long base = 21_000; base <= 21_020; base++) {
            PriceRule flat = new FlatRule("flat", base);
            long expressFirst = new MemberDiscountRule(new ExpressRule(flat),
10).priceOf(member);
            long discountFirst = new ExpressRule(new MemberDiscountRule(flat,
10)).priceOf(member);
            if (expressFirst != discountFirst) {
                differences++;
                System.out.println("base " + base + ": express first " + expressFirst
                        + ", discount first " + discountFirst);
            }
        }
        System.out.println(differences + " of 21 bases differ");
    }
}
```

```
base 21007: express first 28359, discount first 28360
base 21008: express first 28361, discount first 28362
base 21009: express first 28362, discount first 28363
base 21010: express first 28364, discount first 28363
base 21012: express first 28367, discount first 28366
base 21015: express first 28370, discount first 28371
base 21017: express first 28373, discount first 28374
base 21019: express first 28376, discount first 28377
8 of 21 bases differ
```

Program Python, **Kode 6.71**, mencetak delapan baris yang sama.

Kode 6.71 `decorator_order.py`: percobaan yang sama dalam Python.

```python
"""Does the order in which ExpressRule and MemberDiscountRule are stacked matter?"""
from order import Order
from rules import ExpressRule, FlatRule, MemberDiscountRule

member = Order("D1", "Test", 0, 1, True)
differences = 0
for base in range(21_000, 21_021):
    flat = FlatRule("flat", base)
    express_first = MemberDiscountRule(ExpressRule(flat), 10).price_of(member)
    discount_first = ExpressRule(MemberDiscountRule(flat, 10)).price_of(member)
    if express_first != discount_first:
        differences += 1
        print(f"base {base}: express first {express_first}, discount first
{discount_first}")
print(differences, "of 21 bases differ")
```

Kedua urutan berbeda pada 8 dari 21 kasus, selalu satu rupiah, karena setiap dekorator membulatkan ke bawah *pada langkahnya sendiri*. Ambil harga dasar 21 007. Ekspres lebih dulu: $21\,007 + 10\,503 = 31\,510$, lalu dikurangi $\lfloor 3\,151.0 \rfloor = 3\,151$, menghasilkan 28 359. Potongan lebih dulu: $21\,007 - \lfloor 2\,100.7 \rfloor = 18\,907$, lalu ditambah $\lfloor 9\,453.5 \rfloor = 9\,453$, menghasilkan 28 360. Secara matematis kedua operasi itu komutatif; dengan pembulatan di setiap langkah, tidak lagi. Laundry sebaiknya *memutuskan* satu urutan, menuliskannya di dokumentasi *factory* `ruleFor`, dan menambahkan uji dengan harga dasar yang canggung seperti 21 007 agar tak seorang pun menukar urutannya tanpa ketahuan. (`ruleFor` pada **Kode 6.42** menerapkan ekspres lebih dulu, lalu potongan anggota.)

Penyelesaian 6.13. **Kode 6.72** mengimplementasikan `Comparator<Order>`. `Double.compare(b, a)` dengan argumen yang dibalik memberikan urutan menurun berdasarkan berat; jika beratnya sama, nomor tiket yang memutuskan, dengan `String.compareTo` — yang hanya kita teruskan tandanya, bukan nilainya, sebagaimana diwajibkan Persamaan (6.5).

Kode 6.72 `OrderSortDemo.java`: `Comparator` yang mengurutkan dari yang terberat, lalu menurut nomor tiket.

```
import java.util.ArrayList;
import java.util.Comparator;
import java.util.List;

/**
 * Sorting orders with a Comparator: another interface from java.util.
 */
public class OrderSortDemo {
    /** Heaviest order first; equal weights in ticket-id order. */
    static class HeaviestFirst implements Comparator<Order> {
        @Override
        public int compare(Order a, Order b) {
            int byWeight = Double.compare(b.getWeightKg(), a.getWeightKg());  // b sebelum
a: menurun
            if (byWeight != 0) {
                return byWeight;
            }
            return a.getId().compareTo(b.getId());
        }
    }

    public static void main(String[] args) {
        List<Order> orders = new ArrayList<>(List.of(
                new Order("A003", "Dewi", 3.2, 0, true),
                new Order("A001", "Sari", 2.0, 0, false),
                new Order("A005", "Sari", 5.5, 0, false),
                new Order("A002", "Budi", 3.2, 0, true)));
        orders.sort(new HeaviestFirst());
        for (Order o : orders) {
            System.out.println(o);
        }
    }
}
```

```
Order[A005, Sari, 5.5 kg, 0 pcs]
Order[A002, Budi, 3.2 kg, 0 pcs, member]
Order[A003, Dewi, 3.2 kg, 0 pcs, member]
Order[A001, Sari, 2.0 kg, 0 pcs]
```

Di Python (**Kode 6.73**), kunci `(-o.weight_kg, o.id)` mengatakan hal yang sama sebagai *tuple*: tanda minus membalik urutan berat, dan nomor tiket memutus seri.

Kode 6.73 `order_sort_demo.py`: pengurutan dengan kunci *tuple*.

```
"""Sorting orders with a key function: heaviest first, then by ticket id."""
from order import Order

orders = [Order("A003", "Dewi", 3.2, 0, True),
          Order("A001", "Sari", 2.0, 0, False),
          Order("A005", "Sari", 5.5, 0, False),
          Order("A002", "Budi", 3.2, 0, True)]
for o in sorted(orders, key=lambda o: (-o.weight_kg, o.id)):   # tanda minus: menurun
    print(o)
```

```
Order[A005, Sari, 5.5 kg, 0 pcs]
Order[A002, Budi, 3.2 kg, 0 pcs, member]
Order[A003, Dewi, 3.2 kg, 0 pcs, member]
Order[A001, Sari, 2.0 kg, 0 pcs]
```

`Comparable` menggambarkan satu-satunya urutan *alami* sebuah objek. Pesanan tidak memilikinya: kasir menginginkannya menurut nomor tiket, ruang cuci dari yang terberat, kurir menurut alamat. `Comparator` adalah urutan yang dipilih dari luar, dan sebuah kelas boleh memiliki sebanyak yang dibutuhkan para penggunanya.

Daftar Pustaka

[1] D. Goodger and G. van Rossum, "PEP 257 – Docstring Conventions," Python Software Foundation, 2001. [Online]. Available: https://peps.python.org/pep-0257/

[2] Python Software Foundation, "doctest — Test interactive Python examples," *The Python Standard Library*, 2025. [Online]. Available: https://docs.python.org/3/library/doctest.html

[3] Oracle, "The javadoc Command," *Java Development Kit Version 21 Tool Specifications*, 2023. [Online]. Available: https://docs.oracle.com/en/java/javase/21/docs/specs/man/javadoc.html

[4] The JUnit Team, *JUnit User Guide*, version 6.1.3, Aug. 2026. Accessed: Sep. 24, 2026. [Online]. Available: https://docs.junit.org/current/

[5] Python Software Foundation, "unittest — Unit testing framework," *The Python Standard Library*, 2025. [Online]. Available: https://docs.python.org/3/library/unittest.html

[6] H. Krekel and the pytest-dev team, *pytest Documentation*, 2025. [Online]. Available: https://docs.pytest.org/

[7] G. J. Myers, C. Sandler, and T. Badgett, *The Art of Software Testing*, 3rd ed. Hoboken, NJ, USA: Wiley, 2011.

[8] K. Beck, *Test-Driven Development: By Example*. Boston, MA, USA: Addison-Wesley, 2002.

[9] Oracle, "Class Object," *Java Platform, Standard Edition 21 API Specification*, 2023. [Online]. Available: https://docs.oracle.com/en/java/javase/21/docs/api/java.base/java/lang/Object.html

[10] J. Bloch, *Effective Java*, 3rd ed. Boston, MA, USA: Addison-Wesley, 2018.

[11] J. Gosling, B. Joy, G. Steele, G. Bracha, A. Buckley, D. Smith, and G. Bierman, *The Java Language Specification, Java SE 21 Edition*, §15.12.2 "Compile-Time Step 2: Determine Method Signature," Oracle, 2023. [Online]. Available: https://docs.oracle.com/javase/specs/jls/se21/html/

[12] Python Software Foundation, "functools — Higher-order functions and operations on callable objects," *The Python Standard Library*, 2025. [Online]. Available: https://docs.python.org/3/library/functools.html

[13] Python Software Foundation, "abc — Abstract Base Classes," *The Python Standard Library*, 2025. [Online]. Available: https://docs.python.org/3/library/abc.html

[14] I. Levkivskyi, J. Lehtosalo, and Ł. Langa, "PEP 544 – Protocols: Structural subtyping (static duck typing)," Python Software Foundation, 2017. [Online]. Available: https://peps.python.org/pep-0544/

[15] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Reading, MA, USA: Addison-Wesley, 1994.

Bab 7

Pewarisan, Polimorfisme, dan Kontrol Akses

Pada akhir bab ini Anda akan sanggup mengenali jenis kode kembar yang menjadi alasan diciptakannya pewarisan, lalu menghapusnya; menulis sebuah superkelas beserta subkelas-subkelasnya, di Java dengan `extends` dan di Python dengan `class Sub(Base)`; merangkai konstruktor dengan benar melalui `super(...)` dan `super().__init__(...)`; mendefinisikan ulang (*override*) metode yang diwarisi, serta membedakan *overriding* dari *overloading* dalam sekali pandang; menjelaskan, dengan gambaran memori, bagaimana persisnya satu baris seperti `s.price()` dapat menjalankan metode yang berbeda untuk setiap objek yang dijumpainya; memutuskan kapan sebuah kelas sebaiknya abstrak; memilih antara pewarisan dan komposisi serta mempertahankan pilihan itu; membaca urutan resolusi metode (*method resolution order*) Python dan membandingkannya dengan pewarisan tunggal ditambah antarmuka di Java; serta memakai `public`, `protected`, akses paket, dan `private` di Java, juga konvensi `_name`, `__name`, dan `@property` di Python, untuk membangun kelas yang melindungi datanya sendiri. Semuanya bertemu dalam sebuah mesin harga yang berfungsi untuk Profitina Laundry, tempat satu perulangan menghitung harga cuci, setrika, *dry cleaning*, dan pesanan kilat tanpa perlu tahu mana yang mana.

7.1 Di Mana Kita Berhenti

Bab 3 memberi Profitina Laundry kelas sungguhan yang pertama, `Service`, dengan kode, nama, harga per kilogram, dan berat minimum. Kelas itu bekerja. Namun ada sedikit bau yang waktu itu tidak kita bahas: untuk menjawab “berapa lama pengerjaannya?”, kelas itu memeriksa kodenya sendiri dengan `switch` — `"WF"` berarti dua hari, `"IR"` satu hari, dan seterusnya — sedangkan apakah sebuah pesanan *kilat* ditentukan oleh sebuah `boolean` yang dikirim dari luar dan diuji dengan `if` di dalam metode penghitung harga. Bab 5 menampung banyak layanan dan pesanan di dalam list dan mengajari kita untuk gagal dengan lantang melalui eksepsi. Bab 6 mengajari kita mendokumentasikan dan menguji sebuah kelas, memperlihatkan bagaimana sebuah *interface* (antarmuka) menyatakan apa yang dijanjikan sebuah kelas tanpa menyebutkan caranya, dan memperkenalkan *overloading*: beberapa metode bernama sama dengan daftar parameter yang berbeda.

Sekarang bayangkan binatu itu berkembang. Pemiliknya menambahkan *dry cleaning* (cuci kering), yang harganya dihitung *per potong*, bukan per kilogram. Lalu cuci sepatu, per pasang. Lalu promosi potongan 20% untuk satu layanan. Setiap jenis layanan baru berarti membuka `Service` lagi, menambah satu `case` lagi pada `switch`, satu `if` lagi pada metode harga, sambil berharap tidak ada yang tadinya jalan kini rusak. Setahun kemudian kelas itu menjadi seribu baris kasus khusus yang tidak berani disentuh siapa pun.

Bab ini membahas tiga gagasan yang ditawarkan pemrograman berorientasi objek untuk melawan kemerosotan semacam itu. **Pewarisan** (*inheritance*) memungkinkan kita menulis bagian bersama dari beberapa kelas satu kali saja, di dalam sebuah *superkelas*, lalu membiarkan setiap *subkelas* menambah atau mengubah hanya apa yang berbeda darinya. **Polimorfisme**

(*polymorphism*) memungkinkan satu baris kode, misalnya `s.price()`, berbuat hal yang tepat untuk setiap jenis objek yang diberikan kepadanya, sehingga kode yang *memakai* layanan tidak pernah memerlukan `switch` sama sekali. **Kontrol akses** (*access control*) memungkinkan setiap kelas menentukan bagian mana dari dirinya yang boleh disentuh bagian lain program, sehingga janji-janji sebuah kelas tidak dapat dilanggar dari luar.

Kita menjumpai gagasan-gagasan itu dalam urutan yang biasa dialami seorang pemrogram: sebuah masalah (dua kelas yang saling mengulang), obatnya (sebuah superkelas), kejutan-kejutan yang dibawa obat itu (metode mana yang berjalan? apa yang terjadi pada konstruktor?), dan disiplin yang menjaga sebuah hierarki tetap sehat (kelas abstrak, komposisi, antarmuka, *access modifier*). Setiap gagasan diperlihatkan dalam Java, lalu dalam Python, lalu dijalankan, lalu dijelaskan.

> **WAWASAN KUNCI:** Pewarisan berarti *tidak menulis kode yang sama dua kali*. Polimorfisme berarti *tidak menanyakan jenis objek apa yang sedang Anda pegang*. Kontrol akses berarti *tidak membiarkan siapa pun melanggar janji sebuah objek*. Ingatlah ketiga kalimat ini; segala isi bab ini adalah rincian dari salah satunya.

7.2 Masalah Duplikasi: Dua Basis Data yang Saling Mengulang

Kita tidak mulai dengan binatu, tetapi dengan sebuah contoh kecil dan bersih yang memperlihatkan masalahnya dalam bentuk paling murni. Misalkan kita menulis basis data mungil yang menyimpan kunci bilangan bulat. Kita memerlukan dua implementasi, karena masing-masing unggul dalam hal yang berbeda:

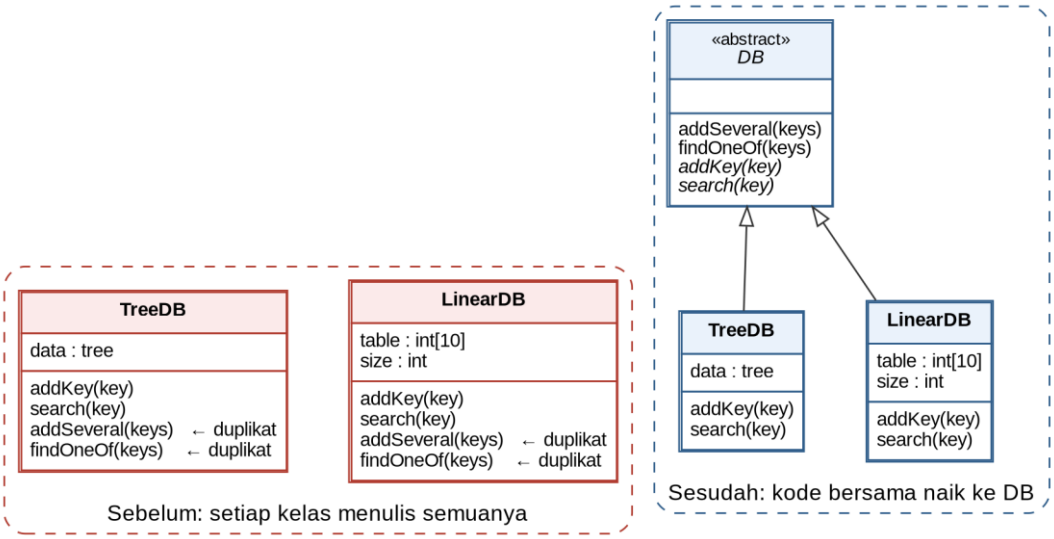
- `LinearDB` menyimpan kunci di dalam larik, sesuai urutan kedatangannya. Menambah kunci terjadi seketika (taruh di ujung); mencari kunci berarti menelusuri larik dari awal.
- `TreeDB` menyimpan kunci di dalam pohon pencarian biner (*binary search tree*). Menambah maupun mencari sama-sama menelusuri satu lintasan dari akar ke bawah, yang pendek bila pohonnya seimbang.

Kedua kelas memerlukan empat operasi yang sama:

```
void    addKey(int key)          menambahkan satu kunci
boolean search(int key)          apakah kunci ini tersimpan?
void    addSeveral(int keys[])   menambahkan setiap kunci di dalam larik
boolean findOneOf(int keys[])    apakah paling sedikit satu kunci ini tersimpan?
```

Perhatikan baik-baik dua operasi terakhir. Bagaimana cara menambahkan beberapa kunci? Panggil `addKey` sekali untuk setiap kunci. Bagaimana cara mengetahui apakah salah satu dari beberapa kunci ada? Panggil `search` untuk setiap kunci dan berhenti pada keberhasilan pertama. Tidak satu pun jawaban itu menyebut larik atau pohon. `addSeveral` dan `findOneOf` tidak peduli *bagaimana* sebuah kunci disimpan; keduanya hanya membutuhkan *suatu* `addKey` dan *suatu* `search` untuk dipanggil.

Jadi, bila `LinearDB` dan `TreeDB` kita tulis sebagai dua kelas yang tidak berhubungan, `addSeveral` dan `findOneOf` tertulis dua kali, huruf demi huruf. Bagian kiri **Gambar 7.1** memperlihatkan hasilnya: dua kelas, delapan metode, empat di antaranya salinan persis. Kode kembar bukan sekadar pemborosan. Ia jebakan. Pada hari seseorang menemukan *bug* di `findOneOf` dan memperbaikinya di `LinearDB`, salinan di `TreeDB` tetap membawa *bug* itu, dan dua kelas yang semestinya berperilaku sama kini diam-diam berbeda pendapat.



Gambar 7.1 Masalah duplikasi dan obatnya. Kiri: dua kelas yang berdiri sendiri, masing-masing membawa salinan `addSeveral` dan `findOneOf`. Kanan: metode bersama naik ke superkelas abstrak `DB`; setiap subkelas hanya menyimpan apa yang benar-benar miliknya.

Bagian kanan **Gambar 7.1** memperlihatkan obatnya. Kita menciptakan kelas ketiga, `DB`, yang menampung segala sesuatu yang dimiliki bersama oleh kedua basis data: dua metode yang kodenya sama, ditambah *deklarasi* — sebuah nama dan daftar parameter, tanpa badan — dari dua metode yang kodenya berbeda. `LinearDB` dan `TreeDB` lalu menyatakan "aku adalah sejenis `DB`", dan keduanya otomatis menerima `addSeveral` dan `findOneOf`. Masing-masing hanya menulis `addKey` dan `search` miliknya sendiri.

Gagasan ini tiga puluh tahun lebih tua daripada Java dan Python. Ia pertama kali muncul dalam Simula 67, bahasa dari Norwegia yang dirancang untuk menyimulasikan sistem berisi objek-objek yang saling berinteraksi; para perancangnya memperhatikan bahwa berbagai jenis objek simulasi berbagi sebagian besar perilakunya dan ingin menulis perilaku itu sekali saja [1]. Setiap bahasa berorientasi objek arus utama sesudahnya mewarisi — permainan kata ini sulit ditolak — mekanisme yang sama.

7.2.1 Kosakata

> **DEFINISI.** Bila kelas `C` dideklarasikan sebagai perluasan dari kelas `B`, kita katakan bahwa `C` **mewarisi** `B`. `B` adalah **superkelas** (juga disebut *kelas dasar* atau *kelas induk*) dan `C` adalah **subkelas** (juga *kelas turunan* atau *kelas anak*). Subkelas secara otomatis memiliki setiap field dan setiap metode superkelasnya, di samping yang ia deklarasikan sendiri.

Dalam contoh kita, `DB` adalah superkelas, sedangkan `LinearDB` dan `TreeDB` adalah subkelasnya. Kita katakan `LinearDB` dan `TreeDB` *mewarisi* `DB`, karena keduanya memperoleh definisi `addSeveral` dan `findOneOf` darinya tanpa menulisnya.

Hubungan ini punya nama yang patut Anda ucapkan keras-keras setiap kali merancang hierarki: **is-a** (*adalah sejenis*). Sebuah `LinearDB` *adalah sejenis* `DB`. Sebuah `TreeDB` *adalah sejenis* `DB`. Bila kalimatnya terdengar janggal — "sebuah `Customer` adalah sejenis `Service`" — pewarisannya keliru, sebanyak apa pun kode yang dapat dihematnya. Kita kembali ke uji ini di Subbab 7.11.

7.3 Superkelas dan Subkelas di Java dan Python

Kode 7.1 adalah proyek basis data yang ditulis dalam Java. Semuanya berada dalam satu berkas, `DBTest.java`, di dalam paket bernama `db` (sebuah *package* atau paket hanyalah folder kelas yang diberi nama; Subbab 7.13 menjelaskan mengapa paket penting bagi kontrol akses). Berkas itu memuat empat kelas: kelas publik `DBTest`, yang berisi `main`, dan tiga kelas pembantu yang hanya terlihat di dalam paketnya.

Kode 7.1 `db/DBTest.java`: proyek basis data dengan superkelas abstrak `DB` dan dua subkelas, `LinearDB` dan `TreeDB`, yang metode-metodenya hanya melaporkan bahwa mereka dipanggil.

```
package db;

public class DBTest {
    public static void main(String[] args) {
        DB ldb = new LinearDB();
        DB tdb = new TreeDB();
        ldb.addKey(1);
        if (ldb.search(1)) {
            System.out.println("It's found!");
        } else {
            System.out.println("It's not found!");
        }
        tdb.addKey(2);
        if (tdb.search(1)) {
            System.out.println("It's found!");
        } else {
            System.out.println("It's not found!");
        }
        // Dua metode di bawah ini ditulis sekali, di DB, lalu diwariskan.
        System.out.println("-- inherited addSeveral and findOneOf --");
        ldb.addSeveral(new int[] {3, 4});
        System.out.println("findOneOf: " + tdb.findOneOf(new int[] {5, 6}));
    }
}

abstract class DB {
    void addSeveral(int keys[]) {
        for (int i = 0; i < keys.length; i++) {
            addKey(keys[i]);
        }
    }

    boolean findOneOf(int keys[]) {
        for (int i = 0; i < keys.length; i++) {
            if (search(keys[i])) {
                return true;
            }
        }
        return false;
    }

    abstract void addKey(int key);     // Tanpa kode
    abstract boolean search(int key);  // Tanpa kode
}
```

Kode 7.1 (Lanjutan) *db/DBTest.java: proyek basis data dengan superkelas abstrak DB dan dua subkelas, LinearDB dan TreeDB, yang metode-metodenya hanya melaporkan bahwa mereka dipanggil.*

```
class LinearDB extends DB {
    int table[] = new int[10];
    int size = 0;

    void addKey(int key) {
        System.out.println("LinearDB: addKey()");
    }

    boolean search(int key) {
        boolean result = true;
        System.out.println("LinearDB: searchKey()");
        return result;
    }
}

class TreeDB extends DB {
    Object data;  // pohon pencarian biner akan disimpan di sini

    void addKey(int key) {
        System.out.println("TreeDB: addKey()");
    }

    boolean search(int key) {
        boolean result = false;
        System.out.println("TreeDB: searchKey()");
        return result;
    }
}
```

Untuk mengompilasi dan menjalankannya, bukalah terminal di folder `java` bab ini dan ketik `javac -d out db/DBTest.java`, lalu `java -cp out db.DBTest` (README folder kode mengulang langkah ini untuk Windows, macOS, dan Linux). Program mencetak:

```
LinearDB: addKey()
LinearDB: searchKey()
It's found!
TreeDB: addKey()
TreeDB: searchKey()
It's not found!
-- inherited addSeveral and findOneOf --
LinearDB: addKey()
LinearDB: addKey()
TreeDB: searchKey()
TreeDB: searchKey()
findOneOf: false
```

Mari kita baca listing ini baris demi baris, dimulai dari superkelasnya.

- `abstract class DB {` memulai superkelas. Kata `abstract` berarti "kelas ini belum lengkap; Anda tidak boleh membuat objeknya secara langsung". Ia belum lengkap karena dua metodenya tidak berbadan, seperti yang segera kita lihat. Subbab 7.9 dikhususkan untuk kelas abstrak; untuk sementara terimalah bahwa `DB` ada hanya untuk diwarisi.
- `void addSeveral(int keys[])` adalah metode biasa dengan badan biasa: sebuah perulangan `for` yang memanggil `addKey(keys[i])` untuk setiap kunci. Perhatikan apa yang dipanggilnya. `DB` tidak punya kode untuk `addKey` — namun panggilan itu lolos kompilasi, karena `DB` *mendeklarasikan* bahwa setiap `DB` memiliki `addKey`.
- `boolean findOneOf(int keys[])` memakai gagasan yang sama: ulangi, panggil `search`, kembalikan `true` pada keberhasilan pertama, `false` bila tidak ada yang berhasil.

- `abstract void addKey(int key);` dan `abstract boolean search(int key);` adalah *metode abstrak*: tipe kembalian, nama, dan daftar parameter, lalu titik koma sebagai ganti badan. Keduanya adalah janji: "setiap jenis `DB` yang konkret akan menyediakan metode ini".
- `class LinearDB extends DB {` adalah jantung listing ini. Kata kunci `extends` menjadikan `LinearDB` subkelas dari `DB`. `LinearDB` lalu mendeklarasikan dua field miliknya sendiri, `table` dan `size`, dan menyediakan badan untuk `addKey` dan `search`. Pada versi pertama ini badan-badan itu hanya mencetak satu baris dan mengembalikan jawaban yang tetap; `search` selalu berkata `true`. Itu disengaja: program ini kerangka yang memperlihatkan *metode mana* yang berjalan, bukan basis data yang berfungsi. Versi yang berfungsi menyusul di Subbab 7.4.
- `class TreeDB extends DB {` melakukan hal yang sama untuk versi pohon. `search` miliknya selalu berkata `false`.
- Di dalam `main`, baris `DB ldb = new LinearDB();` layak dibaca pelan-pelan. *Variabel* `ldb` dideklarasikan bertipe `DB`. *Objek* yang dibuat adalah `LinearDB`. Hal ini diizinkan justru karena sebuah `LinearDB` adalah sejenis `DB`: di mana pun program mengharapkan `DB`, objek subkelas apa pun dapat dipakai. Namun kita tidak dapat menulis `new DB()`, karena `DB` abstrak; tidak ada objek `DB`, yang ada hanya objek `LinearDB` dan `TreeDB`.

Sekarang cocokkan keluaran dengan kodenya. `ldb.addKey(1)` mencetak `LinearDB: addKey()`: panggilan itu menuju metode milik `LinearDB` walaupun tipe variabelnya `DB`. `ldb.search(1)` mengembalikan `true`, sehingga program mencetak `It's found!`; pohon mengembalikan `false`, maka tercetak `It's not found!`. Dua baris tambahan di akhir `main` menguji metode-metode warisan. `ldb.addSeveral(new int[] {3, 4})` menjalankan perulangan yang ditulis di `DB`, dan keluaran memperlihatkan `LinearDB: addKey()` dua kali: perulangan warisan itu memanggil versi milik `LinearDB`. Demikian pula `tdb.findOneOf(new int[] {5, 6})` menjalankan perulangan milik `DB`, yang memanggil `search` milik `TreeDB` dua kali sehingga mengembalikan `false`.

Pengamatan terakhir itu adalah seluruh polimorfisme dalam bentuk mini, dan kita akan menghabiskan Subbab 7.7 untuknya: **kode yang ditulis di superkelas memanggil metode yang dipilih oleh kelas objek yang sebenarnya**.

7.3.1 "Seolah-olah LinearDB didefinisikan sebagai…"

Cara yang baik untuk memahami apa yang dilakukan `extends` adalah membayangkan kompilator menyalin anggota-anggota superkelas ke dalam subkelas. Setelah pewarisan, `LinearDB` berperilaku persis seolah-olah kita menuliskannya seperti ini:

```
class LinearDB {
    int table[] = new int[10];      // dideklarasikan di LinearDB
    int size = 0;                   // dideklarasikan di LinearDB
    void addKey(int key)            // dideklarasikan di LinearDB
    boolean search(int key)         // dideklarasikan di LinearDB
    void addSeveral(int keys[])     // diwarisi dari DB
    boolean findOneOf(int keys[])   // diwarisi dari DB
}
```

dan demikian pula untuk `TreeDB`. Gambaran ini model mental, bukan apa yang harfiah dilakukan mesin — tidak ada kode yang disalin; Subbab 7.7 memperlihatkan apa yang sebenarnya terjadi — tetapi ia meramalkan dengan tepat anggota apa saja yang dimiliki sebuah subkelas.

Aturan ini berlaku menurun sepanjang rantai. Diberikan

```
abstract class B { … }
abstract class C extends B { … }
class D extends C { … }
```

`C` memiliki semua field dan metode `B` di samping miliknya sendiri, dan `D` memiliki milik `C` — yang sudah mencakup milik `B` — di samping miliknya sendiri. Pewarisan bersifat *transitif*: `D` mewarisi `C` sekaligus `B`. Kita akan membangun rantai tiga tingkat untuk binatu di Subbab 7.15.

7.3.2 Program yang sama dalam Python

Kode 7.2 adalah proyek yang sama dalam Python. Python tidak punya kata kunci `abstract`; sebagai gantinya modul pustaka standar `abc` (*Abstract Base Classes*) menyediakan kelas dasar `ABC` dan dekorator `@abstractmethod`, dan keduanya bersama-sama memberi Python gagasan "kelas yang belum lengkap" yang sama [2].

Kode 7.2 `db_test.py`: proyek basis data dalam Python. `DB` mewarisi `ABC` dan menandai `add_key` dan `search` sebagai abstrak.

```
"""db_test.py: proyek basis data, dengan DB sebagai superkelas abstrak."""
from abc import ABC, abstractmethod


class DB(ABC):
    def add_several(self, keys):
        for key in keys:
            self.add_key(key)

    def find_one_of(self, keys):
        for key in keys:
            if self.search(key):
                return True
        return False

    @abstractmethod
    def add_key(self, key):  # Tanpa kode
        ...

    @abstractmethod
    def search(self, key):  # Tanpa kode
        ...


class LinearDB(DB):
    def __init__(self):
        self.table = [0] * 10
        self.size = 0

    def add_key(self, key):
        print("LinearDB: addKey()")

    def search(self, key):
        result = True
        print("LinearDB: searchKey()")
        return result


class TreeDB(DB):
    def __init__(self):
        self.data = None  # pohon pencarian biner akan disimpan di sini

    def add_key(self, key):
        print("TreeDB: addKey()")
```

Kode 7.2 (Lanjutan) *db_test.py: proyek basis data dalam Python. DB mewarisi ABC dan menandai add_key dan search sebagai abstrak.*

```
    def search(self, key):
        result = False
        print("TreeDB: searchKey()")
        return result


def main():
    ldb = LinearDB()
    tdb = TreeDB()
    ldb.add_key(1)
    if ldb.search(1):
        print("It's found!")
    else:
        print("It's not found!")
    tdb.add_key(2)
    if tdb.search(1):
        print("It's found!")
    else:
        print("It's not found!")
    # Dua metode di bawah ini ditulis sekali, di DB, lalu diwariskan.
    print("-- inherited add_several and find_one_of --")
    ldb.add_several([3, 4])
    print("find_one_of:", tdb.find_one_of([5, 6]))
    try:
        DB()
    except TypeError as e:
        print("DB():", e)


if __name__ == "__main__":
    main()
```

Jalankan dengan `python3 db_test.py` (di Windows, `python db_test.py` atau `py db_test.py`):

```
LinearDB: addKey()
LinearDB: searchKey()
It's found!
TreeDB: addKey()
TreeDB: searchKey()
It's not found!
-- inherited add_several and find_one_of --
LinearDB: addKey()
LinearDB: addKey()
TreeDB: searchKey()
TreeDB: searchKey()
find_one_of: False
DB(): Can't instantiate abstract class DB with abstract methods add_key, search
```

Perbedaannya dengan Java sedikit dan patut didaftar:

- **Sintaksis pewarisan.** Python menuliskan superkelas di dalam tanda kurung setelah nama kelas: `class LinearDB(DB):`. Tidak ada `extends`.
- **Metode abstrak.** `DB` sendiri mewarisi `ABC`, dan setiap metode yang belum selesai diberi `@abstractmethod`. Badan `...` (*Ellipsis*, cara Python mengatakan "belum ada apa-apa di sini") adalah pengisi tempat; ia tidak pernah dijalankan.
- **Field.** Python tidak punya deklarasi field. Data sebuah objek tercipta dengan memberi nilai kepada `self.sesuatu`, biasanya di dalam `__init__`, metode yang dipanggil Python untuk menginisialisasi objek baru. Jadi `LinearDB.__init__` menciptakan `self.table` dan `self.size`.

- **Nama.** Gaya Python menulis metode dalam `snake_case` (`add_key`), Java dalam `camelCase` (`addKey`). String yang dicetak sengaja dibuat sama dengan milik Java agar kedua keluaran dapat dibandingkan baris demi baris.
- **Kelas abstrak menolak dibuat.** Baris-baris terakhir `main` mencoba `DB()` dan menangkap `TypeError` yang muncul. Java menolak saat *kompilasi* (programnya tidak pernah berjalan); Python menolak saat *program berjalan*, tepat ketika konstruktor dipanggil. Kalimat pesannya berubah di Python 3.12, yang berbunyi `Can't instantiate abstract class DB without an implementation for abstract methods 'add_key', 'search'`; maknanya sama.

Sebagai perbandingan, inilah yang dikatakan Java bila Anda menambahkan baris `DB nothing = new DB();` tepat setelah `DB tdb = new TreeDB();` di **Kode 7.1**:

```
db/DBTest.java:7: error: DB is abstract; cannot be instantiated
        DB nothing = new DB();
                     ^
1 error
```

7.4 Mendefinisikan Ulang Metode: Overriding

Subkelas yang hanya dapat *menambah* sesuatu pada superkelasnya tidak akan banyak gunanya. Aspek pewarisan yang benar-benar penting adalah bahwa subkelas juga dapat **mendefinisikan ulang** metode yang diwarisinya — menyediakan badannya sendiri bagi metode yang sudah dimiliki superkelas. Aturannya yang tepat adalah:

> **DEFINISI.** Subkelas `C` dari `B` mewarisi variabel instans dan metode instans milik `B`, **kecuali yang didefinisikan sendiri oleh `C`**. Bila `C` mendefinisikan metode dengan nama yang sama dan tipe parameter yang sama dengan sebuah metode milik `B`, kita katakan bahwa `C` **meng-*override*** (menimpa) metode itu. Panggilan pada objek `C` lalu menjalankan versi milik `C`.

Mengapa `LinearDB` ingin meng-*override* `addSeveral`? Karena versi warisan memanggil `addKey` sekali per kunci, dan setiap panggilan memeriksa apakah larik masih punya ruang. Basis data berbasis larik dapat berbuat lebih baik: sediakan ruang sekali, lalu salin semua kunci dalam satu blok. Versi langsung untuk kerangka **Kode 7.1** dapat berbunyi:

```
void addSeveral(int keys[]) {
    for (int i = 0; i < keys.length; i++) {
        table[size + i] = keys[i];
    }
    size += keys.length;
}
```

Versi itu lebih cepat, dan sah karena `LinearDB` mengenal `table` dan `size`, sedangkan `DB` tidak. Namun ia juga agak berbahaya: ia lupa memeriksa apakah sepuluh slot cukup. Mari kita tulis basis datanya dengan benar, dengan larik yang dapat membesar dan pohon pencarian biner sungguhan, agar kita dapat melihat *overriding* dalam program yang benar-benar menyimpan kunci.

Kode 7.3 mengganti nama kelas-kelasnya menjadi `IntDB`, `LinearIntDB`, dan `TreeIntDB` (agar dapat tinggal di folder yang sama dengan program-program lain bab ini) dan menjadikannya nyata.

Kode 7.3 `IntDBDemo.java`: basis data kunci yang berfungsi. `LinearIntDB` meng-*override* `addSeveral` warisan dengan penyalinan borongan; `TreeIntDB` mewarisinya tanpa perubahan.

```
import java.util.Arrays;

public class IntDBDemo {
    public static void main(String[] args) {
        IntDB[] dbs = {new LinearIntDB(), new TreeIntDB()};
        for (IntDB db : dbs) {
            System.out.println(db.name());
            db.addSeveral(new int[] {42, 7, 19, 3, 25});
            System.out.println("  size                = " + db.size());
            System.out.println("  search(19)          = " + db.search(19));
            System.out.println("  search(8)           = " + db.search(8));
            System.out.println("  findOneOf({8,9,25}) = " + db.findOneOf(new int[] {8, 9,
25}));
            System.out.println("  findOneOf({1,2})    = " + db.findOneOf(new int[] {1,
2}));
            System.out.println("  comparisons so far  = " + db.comparisons);
        }
    }
}

abstract class IntDB {
    protected int comparisons = 0;  // dihitung oleh search(), agar keduanya dapat
dibandingkan

    // Ditulis sekali, diwarisi oleh setiap subkelas.
    void addSeveral(int[] keys) {
        for (int key : keys) {
            addKey(key);
        }
    }

    // Ditulis sekali, diwarisi oleh setiap subkelas.
    boolean findOneOf(int[] keys) {
        for (int key : keys) {
            if (search(key)) {
                return true;
            }
        }
        return false;
    }

    abstract void addKey(int key);
    abstract boolean search(int key);
    abstract int size();
    abstract String name();
}
```

Kode 7.3 (Lanjutan) *IntDBDemo.java: basis data kunci yang berfungsi. LinearIntDB meng-override addSeveral warisan dengan penyalinan borongan; TreeIntDB mewarisinya tanpa perubahan.*

```
class LinearIntDB extends IntDB {
    private int[] table = new int[4];
    private int size = 0;

    @Override
    void addKey(int key) {
        if (size == table.length) {
            table = Arrays.copyOf(table, 2 * table.length);  // perbesar ketika penuh
        }
        table[size++] = key;
    }

    // Pendefinisian ulang: satu salinan borongan, bukan satu panggilan addKey() per kunci.
    @Override
    void addSeveral(int[] keys) {
        while (size + keys.length > table.length) {
            table = Arrays.copyOf(table, 2 * table.length);
        }
        System.arraycopy(keys, 0, table, size, keys.length);
        size += keys.length;
        System.out.println("  (bulk copy of " + keys.length + " keys)");
    }

    @Override
    boolean search(int key) {
        for (int i = 0; i < size; i++) {
            comparisons++;
            if (table[i] == key) {
                return true;
            }
        }
        return false;
    }

    @Override
    int size() {
        return size;
    }

    @Override
    String name() {
        return "LinearIntDB (unsorted array)";
    }
}
```

Kode 7.3 (Lanjutan) *IntDBDemo.java: basis data kunci yang berfungsi. LinearIntDB meng-override addSeveral warisan dengan penyalinan borongan; TreeIntDB mewarisinya tanpa perubahan.*

```
class TreeIntDB extends IntDB {
    private static class Node {
        int key;
        Node left, right;

        Node(int key) {
            this.key = key;
        }
    }

    private Node root;
    private int size = 0;

    @Override
    void addKey(int key) {
        if (root == null) {
            root = new Node(key);
            size++;
            return;
        }
        Node cur = root;
        while (true) {
            if (key == cur.key) {
                return;  // sudah ada: himpunan hanya menyimpan satu salinan
            } else if (key < cur.key) {
                if (cur.left == null) {
                    cur.left = new Node(key);
                    size++;
                    return;
                }
                cur = cur.left;
            } else {
                if (cur.right == null) {
                    cur.right = new Node(key);
                    size++;
                    return;
                }
                cur = cur.right;
            }
        }
    }
```

Kode 7.3 (Lanjutan) *IntDBDemo.java: basis data kunci yang berfungsi. LinearIntDB meng-override addSeveral warisan dengan penyalinan borongan; TreeIntDB mewarisinya tanpa perubahan.*

```
    @Override
    boolean search(int key) {
        Node cur = root;
        while (cur != null) {
            comparisons++;
            if (key == cur.key) {
                return true;
            }
            cur = (key < cur.key) ? cur.left : cur.right;
        }
        return false;
    }

    @Override
    int size() {
        return size;
    }

    @Override
    String name() {
        return "TreeIntDB (binary search tree)";
    }
}
```

```
LinearIntDB (unsorted array)
  (bulk copy of 5 keys)
  size                = 5
  search(19)          = true
  search(8)           = false
  findOneOf({8,9,25}) = true
  findOneOf({1,2})    = false
  comparisons so far  = 33
TreeIntDB (binary search tree)
  size                = 5
  search(19)          = true
  search(8)           = false
  findOneOf({8,9,25}) = true
  findOneOf({1,2})    = false
  comparisons so far  = 22
```

Telusurilah bagian-bagiannya.

- **`IntDB`** adalah superkelas abstrak. Ia memiliki satu field, `comparisons`, bertanda `protected` — terlihat oleh subkelas (Subbab 7.13 memberikan aturan tepatnya). Kedua subkelas menaikkan nilainya di dalam `search`, agar kita dapat membandingkan seberapa keras kedua struktur data itu bekerja. `addSeveral` dan `findOneOf` ditulis sekali. `addKey`, `search`, `size`, dan `name` abstrak.
- **`LinearIntDB`** menyimpan kunci di `table`, dimulai dengan empat slot. `addKey` miliknya memeriksa apakah larik sudah penuh dan, bila ya, menggantinya dengan salinan berukuran dua kali lipat (`Arrays.copyOf(table, 2 * table.length)`); lalu kunci disimpan dan `size` dinaikkan. `search` miliknya menelusuri larik dan menghitung satu perbandingan untuk setiap elemen yang diperiksa.
- **Override-nya.** `LinearIntDB` juga mendefinisikan `addSeveral(int[] keys)`, dengan daftar parameter yang persis sama dengan metode warisan. Badannya menggandakan larik sampai semuanya muat, menyalin semua kunci dengan satu panggilan `System.arraycopy`, lalu mencetak catatan agar kita dapat melihat bahwa ia berjalan. Anotasi `@Override` di atasnya berkata kepada kompilator "saya bermaksud meng-*override* metode superkelas; tolong periksa bahwa memang demikian". Anotasi itu tidak mengubah apa pun bila metodenya benar, dan

mengubah kesalahan yang diam — nama yang salah eja, tipe parameter yang keliru — menjadi galat kompilasi. Selalu tuliskan.

- **`TreeIntDB`** menyimpan kunci dalam pohon pencarian biner yang tersusun dari objek `Node` (kelas kecil yang *bersarang* di dalam `TreeIntDB`, karena tidak ada kelas lain yang memerlukannya). `addKey` menelusur dari akar, ke kiri untuk kunci yang lebih kecil dan ke kanan untuk yang lebih besar, lalu menggantungkan simpul baru di tempat penelusuran itu jatuh keluar dari pohon; kunci yang sudah ada diabaikan. `search` menelusuri lintasan yang sama dan menghitung satu perbandingan untuk setiap simpul yang dikunjungi. `TreeIntDB` **tidak** meng-*override* `addSeveral`, sehingga ia memakai perulangan milik `IntDB`.

Keluaran menegaskan setiap klaim itu. Untuk basis data linear, muncul `(bulk copy of 5 keys)`: *override*-nya berjalan. Untuk pohon catatan itu tidak muncul: perulangan warisanlah yang berjalan, memanggil `TreeIntDB.addKey` lima kali. Kedua basis data memberikan jawaban yang sama — sebagaimana mestinya, karena keduanya mengimplementasikan gagasan abstrak yang sama — tetapi jumlah perbandingannya berbeda, 33 lawan 22. Mari kita periksa angka 33 dengan tangan, sebab angka yang tidak dapat Anda hasilkan ulang adalah angka yang tidak patut Anda percayai. Larik berisi ⟨42,7,19,3,25⟩. `search(19)` memeriksa 42, 7, 19: tiga perbandingan. `search(8)` memeriksa kelimanya dan gagal: lima. `findOneOf({8,9,25})` mencari 8 (lima), 9 (lima), dan 25 (lima, ditemukan di posisi terakhir): lima belas. `findOneOf({1,2})` mencari 1 dan 2, masing-masing lima: sepuluh. Totalnya $3 + 5 + 15 + 10 = 33$. Untuk pohon, penyisipan 42, 7, 19, 3, 25 menghasilkan akar 42 dengan anak kiri 7; 7 punya anak kiri 3 dan anak kanan 19; 19 punya anak kanan 25. Setiap pencarian mengikuti satu lintasan akar-ke-daun sepanjang paling banyak empat simpul, itulah sebabnya pohon hanya memerlukan 22 perbandingan secara keseluruhan. Soal Latihan 7.2 meminta Anda menghasilkan ulang angka 22 itu.

Kode 7.4 adalah versi Python-nya.

Kode 7.4 `int_db_demo.py`: basis data yang sama dalam Python; `LinearIntDB` meng-*override* `add_several` dengan satu panggilan `list.extend`.

```
"""int_db_demo.py: basis data kunci yang berfungsi, dua implementasi, satu superkelas."""
from abc import ABC, abstractmethod


class IntDB(ABC):
    def __init__(self):
        self.comparisons = 0  # dihitung oleh search(), agar keduanya dapat dibandingkan

    # Ditulis sekali, diwarisi oleh setiap subkelas.
    def add_several(self, keys):
        for key in keys:
            self.add_key(key)

    # Ditulis sekali, diwarisi oleh setiap subkelas.
    def find_one_of(self, keys):
        for key in keys:
            if self.search(key):
                return True
        return False

    @abstractmethod
    def add_key(self, key): ...

    @abstractmethod
    def search(self, key): ...

    @abstractmethod
    def size(self): ...

    @abstractmethod
    def name(self): ...


class LinearIntDB(IntDB):
    def __init__(self):
        super().__init__()
        self._table = []

    def add_key(self, key):
        self._table.append(key)

    # Pendefinisian ulang: satu extend borongan, bukan satu panggilan add_key() per kunci.
    def add_several(self, keys):
        self._table.extend(keys)
        print(f"  (bulk copy of {len(keys)} keys)")
```

Kode 7.4 (Lanjutan) *int_db_demo.py: basis data yang sama dalam Python; LinearIntDB meng-override add_several dengan satu panggilan list.extend.*

```
    def search(self, key):
        for value in self._table:
            self.comparisons += 1
            if value == key:
                return True
        return False

    def size(self):
        return len(self._table)

    def name(self):
        return "LinearIntDB (unsorted array)"


class _Node:
    def __init__(self, key):
        self.key = key
        self.left = None
        self.right = None


class TreeIntDB(IntDB):
    def __init__(self):
        super().__init__()
        self._root = None
        self._size = 0

    def add_key(self, key):
        if self._root is None:
            self._root = _Node(key)
            self._size += 1
            return
        cur = self._root
        while True:
            if key == cur.key:
                return  # sudah ada: himpunan hanya menyimpan satu salinan
            elif key < cur.key:
                if cur.left is None:
                    cur.left = _Node(key)
                    self._size += 1
                    return
                cur = cur.left
            else:
                if cur.right is None:
                    cur.right = _Node(key)
                    self._size += 1
```

Kode 7.4 (Lanjutan) *int_db_demo.py: basis data yang sama dalam Python; LinearIntDB meng-override add_several dengan satu panggilan list.extend.*

```
                    return
                cur = cur.right

    def search(self, key):
        cur = self._root
        while cur is not None:
            self.comparisons += 1
            if key == cur.key:
                return True
            cur = cur.left if key < cur.key else cur.right
        return False

    def size(self):
        return self._size

    def name(self):
        return "TreeIntDB (binary search tree)"


def main():
    for db in (LinearIntDB(), TreeIntDB()):
        print(db.name())
        db.add_several([42, 7, 19, 3, 25])
        print("  size                =", db.size())
        print("  search(19)          =", db.search(19))
        print("  search(8)           =", db.search(8))
        print("  findOneOf({8,9,25}) =", db.find_one_of([8, 9, 25]))
        print("  findOneOf({1,2})    =", db.find_one_of([1, 2]))
        print("  comparisons so far  =", db.comparisons)


if __name__ == "__main__":
    main()
```

```
LinearIntDB (unsorted array)
  (bulk copy of 5 keys)
  size                = 5
  search(19)          = True
  search(8)           = False
  findOneOf({8,9,25}) = True
  findOneOf({1,2})    = False
  comparisons so far  = 33
TreeIntDB (binary search tree)
  size                = 5
  search(19)          = True
  search(8)           = False
  findOneOf({8,9,25}) = True
  findOneOf({1,2})    = False
  comparisons so far  = 22
```

Kode Python mengikuti Java baris demi baris, dengan tiga perbedaan kecil. List Python membesar dengan sendirinya, sehingga `add_key` cukup satu `append` dan *override* `add_several` cukup satu `extend`. Python tidak punya anotasi `@Override` (Python 3.12 menambahkan `typing.override`, yang hanya dibaca alat pemeriksa tipe); sebuah metode meng-*override* semata-mata karena bernama sama. Dan kelas simpulnya adalah kelas tingkat modul `_Node`, yang garis bawah di depannya berarti "rincian internal" (Subbab 7.13).

> **PERINGATAN:** Di Python, sebuah metode *meng-override* bila **namanya** sama; parameternya tidak berpengaruh. Di Java, metode meng-*override* hanya bila nama **dan tipe parameternya** sama; dengan tipe parameter yang berbeda ia menjadi *overload*, metode yang terpisah. Kedua

> bahasa berbeda pendapat di titik ini, dan hal itu menimbulkan *bug* sungguhan ketika pemrogram berpindah dari satu bahasa ke bahasa lain. Subbab 7.6 kembali membahasnya.

7.4.1 Aturan yang diperiksa Java ketika Anda meng-override

Java ketat soal *overriding*, dan setiap aturannya melindungi janji yang telah diberikan superkelas kepada para pemanggilnya. Bila sebuah metode di kelas `C` memiliki nama dan tipe parameter yang sama dengan metode warisan dari `B`, maka:

1. **Tipe kembaliannya harus sama, atau subtipenya.** Bila `B.make()` mengembalikan `Garment`, maka `C.make()` boleh mengembalikan `Garment` atau `Shirt` (subkelas `Garment`), tetapi tidak `String`. Pemanggil yang dijanjikan `Garment` tetap mendapatkannya. Tipe kembalian yang dipersempit seperti ini disebut *kovarian*.

2. **Aksesnya tidak boleh menjadi lebih ketat.** Metode `public` tidak boleh di-*override* oleh metode *package-private*. Inilah kata `javac` kepada subkelas yang mencoba meng-*override* `public void start()` dengan `void start()`:

```
• Weaker.java:12: error: start() in Derived cannot override start() in Base
    void start() {
         ^
  attempting to assign weaker access privileges; was public
1 error
```

- Alasannya: kode yang memegang variabel `Base` telah dijanjikan bahwa ia boleh memanggil `start()`; bila objeknya ternyata `Derived`, janji itu akan dilanggar.

3. **Metode `final` sama sekali tidak dapat di-*override*.** Menulis `final` di depan sebuah metode berarti "perilaku ini tetap untuk setiap subkelas":

```
• FinalDemo.java:13: error: footer() in FancyReceipt cannot override footer() in
Receipt
    String footer() {
           ^
  overridden method is final
1 error
```

- Sebuah kelas utuh pun dapat dibuat `final` (`String` salah satunya), dan kelas seperti itu tidak boleh diperluas oleh kelas mana pun.

4. **Metode `static` dan metode `private` tidak di-*override*.** Metode statis milik kelas, bukan milik objek, sehingga metode subkelas dengan signature yang sama hanya *menyembunyikannya*; metode private tidak terlihat oleh subkelas, sehingga metode subkelas yang bernama sama hanyalah metode baru yang tidak berhubungan. Subbab 7.7 memperlihatkan mengapa hal ini penting.

5. **Eksepsi terperiksa tidak boleh bertambah.** Metode yang meng-*override* boleh melempar eksepsi terperiksa yang lebih sedikit atau lebih sempit daripada metode yang di-*override*-nya, tidak pernah yang baru atau lebih luas, dengan alasan yang sama seperti aturan 2.

Kelima aturan itu dinyatakan secara formal dalam Spesifikasi Bahasa Java [3]. Python tidak menegakkan satu pun; ia memercayai pemrogram, dan harga kepercayaan itu adalah bahwa kesalahan semacam ini baru tampak ketika program berjalan.

7.5 Konstruktor dan `super`

Sebuah objek subkelas memuat objek superkelas di dalamnya. `LinearIntDB` memiliki `table` dan `size` miliknya, tetapi ia juga memiliki field `comparisons` yang dideklarasikan di `IntDB`, dan harus ada yang menginisialisasi field itu. Aturan yang dianut setiap bahasa berorientasi objek sederhana untuk dinyatakan: **bagian superkelas dari sebuah objek dibangun lebih dahulu, oleh konstruktor superkelas itu sendiri**, dan baru sesudahnya konstruktor subkelas berjalan. Subkelas tidak boleh melewati langkah ini, karena superkelas mengetahui hal-hal tentang field-nya sendiri yang tidak diketahui subkelas.

Untuk melihat urutannya, kita memakai rantai tiga tingkat dari binatu: setiap `LaundryItem` mendapat tag bernomor; `Garment` adalah barang cucian dengan jenis tertentu (handuk, kemeja); `Shirt` adalah pakaian yang boleh dikanji atau tidak. **Kode 7.5** mencetak satu baris dari setiap konstruktor.

Kode 7.5 `ConstructorChain.java`: rantai tiga tingkat `LaundryItem`, `Garment`, `Shirt`; setiap konstruktor mengumumkan dirinya, dan setiap `describe` memperpanjang teks induknya dengan `super.describe()`.

```
public class ConstructorChain {
    public static void main(String[] args) {
        System.out.println("-- new Shirt(\"blue\", true) --");
        Shirt shirt = new Shirt("blue", true);
        System.out.println(shirt.describe());

        System.out.println("-- new Garment(\"towel\") --");
        Garment towel = new Garment("towel");
        System.out.println(towel.describe());
    }
}

class LaundryItem {
    private static int nextTag = 1;
    protected final int tag;

    LaundryItem() {
        tag = nextTag++;
        System.out.println("  LaundryItem() gives tag #" + tag);
    }

    String describe() {
        return "item #" + tag;
    }
}

class Garment extends LaundryItem {
    protected final String kind;

    Garment(String kind) {
        // Kompilator menyisipkan super(); di sini, sehingga LaundryItem() berjalan lebih
dulu.
        this.kind = kind;
        System.out.println("  Garment(\"" + kind + "\")");
    }

    @Override
    String describe() {
        return super.describe() + ", a " + kind;
    }
}

class Shirt extends Garment {
    private final boolean starch;
```

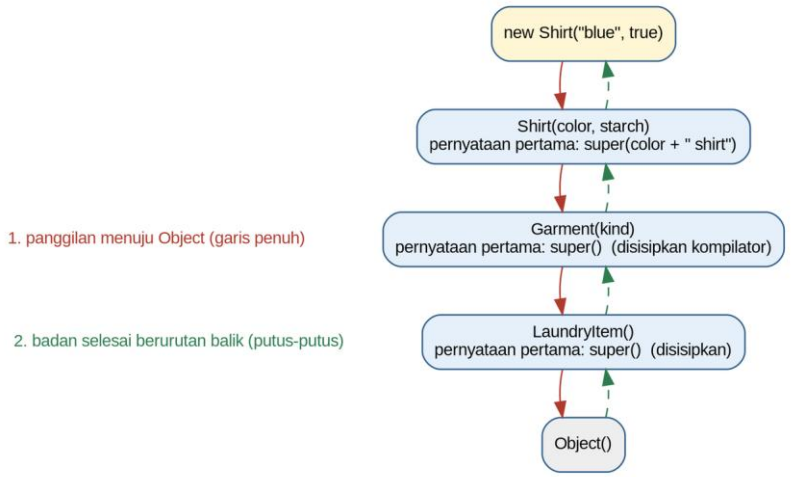
Kode 7.5 (Lanjutan) *ConstructorChain.java: rantai tiga tingkat LaundryItem, Garment, Shirt; setiap konstruktor mengumumkan dirinya, dan setiap describe memperpanjang teks induknya dengan super.describe().*

```
    Shirt(String color, boolean starch) {
        super(color + " shirt");  // harus menjadi pernyataan pertama
        this.starch = starch;
        System.out.println("  Shirt(..., starch=" + starch + ")");
    }

    @Override
    String describe() {
        return super.describe() + (starch ? ", starched" : ", no starch");
    }
}
```

```
-- new Shirt("blue", true) --
  LaundryItem() gives tag #1
  Garment("blue shirt")
  Shirt(..., starch=true)
item #1, a blue shirt, starched
-- new Garment("towel") --
  LaundryItem() gives tag #2
  Garment("towel")
item #2, a towel
```

Bacalah blok keluaran pertama dari atas. Kita meminta sebuah `Shirt`, namun baris pertama yang tercetak berasal dari `LaundryItem`, lalu `Garment`, dan baru kemudian `Shirt`. **Gambar 7.2** memperlihatkan sebabnya. Memanggil `new Shirt("blue", true)` memulai konstruktor `Shirt`; pernyataan pertamanya adalah `super(color + " shirt")`, yang memanggil konstruktor `Garment` dengan argumen `"blue shirt"`. Konstruktor `Garment` diawali panggilan yang tidak kita lihat: bila sebuah konstruktor tidak dimulai dengan `super(...)` atau `this(...)`, kompilator Java menyisipkan `super();` — panggilan ke konstruktor tanpa argumen milik superkelas — atas namanya. Maka `LaundryItem()` berjalan, dan ia pun diawali `super()` yang tak terlihat, yang mencapai konstruktor `Object`, akar setiap kelas Java (Subbab 7.8). Kemudian badan-badan konstruktor itu selesai dalam urutan terbalik: `LaundryItem` memberi tag 1 dan mencetak; `Garment` menyimpan jenisnya dan mencetak; `Shirt` menyimpan penanda kanji dan mencetak.



Gambar 7.2 Perangkaian konstruktor untuk `new Shirt("blue", true)`. Panggilan-panggilan naik menyusuri hierarki sampai `Object` lebih dahulu (panah utuh); badan-badan konstruktor lalu selesai dalam urutan terbalik (panah putus-putus), sehingga bagian objek yang paling umum selalu siap sebelum bagian yang paling khusus.

Tiga aturan lahir dari sini, dan Java menegakkan ketiganya:

1. `super(...)` **harus menjadi pernyataan pertama sebuah konstruktor**, bila memang ditulis. Tidak boleh terjadi apa pun pada sebuah `Shirt` sebelum bagian `Garment`-nya ada. (Sejak Java 25 aturan ini sedikit dilonggarkan oleh *flexible constructor bodies* [4]: pernyataan yang tidak memakai objek yang sedang dibangun boleh ditulis sebelum panggilan itu; gagasannya tidak berubah.)

2. **Bila Anda tidak menulis `super(...)`, kompilator menuliskan `super()` untuk Anda** — dan panggilan itu harus sah. `Garment` *tidak* punya konstruktor tanpa argumen; satu-satunya konstruktornya memerlukan `String`. Jadi, bila konstruktor `Shirt` lupa baris `super(color + " shirt")`, `super()` yang tak terlihat itu tidak punya apa-apa untuk dipanggil. Hapus baris itu dari **Kode 7.5** dan `javac` berkata:

```
• ConstructorChain.java:45: error: constructor Garment in class Garment cannot be
  applied to given types;
      Shirt(String color, boolean starch) {
                                          ^
    required: String
    found:    no arguments
    reason: actual and formal argument lists differ in length
  1 error
```

- Pesan itu menunjuk konstruktor `Shirt` tetapi mengeluhkan konstruktor `Garment`: persis panggilan yang tak terlihat itu.

3. **Konstruktor tidak diwariskan.** `Shirt` tidak mendapat konstruktor `Shirt(String kind)` hanya karena `Garment` punya. Setiap kelas menyatakan sendiri bagaimana objek-objeknya dibangun.

Kata kunci `super` punya kegunaan kedua, yang tampak di ketiga metode `describe`. Di dalam metode yang meng-*override*, `super.describe()` memanggil versi *superkelas* dari metode yang sama — versi yang sedang di-*override*. `Shirt.describe()` meminta teks dari `Garment.describe()`, yang pada gilirannya meminta dari `LaundryItem.describe()`, dan setiap tingkat menambahkan bagiannya sendiri: `item #1`, lalu `, a blue shirt`, lalu `, starched`. Pola ini, "lakukan apa yang dilakukan induk saya, lalu sedikit lagi", adalah alasan paling umum memanggil `super` dari sebuah metode, dan kita akan menjumpainya lagi pada program Animal dan nota binatu.

Kode 7.6 adalah versi Python-nya.

Kode 7.6 `constructor_chain.py`: rantai yang sama dalam Python, ditambah sebuah kelas yang lupa memanggil `super().__init__()`.

```
"""constructor_chain.py: siapa yang berjalan lebih dulu ketika Shirt dibuat?"""


class LaundryItem:
    _next_tag = 1

    def __init__(self):
        self.tag = LaundryItem._next_tag
        LaundryItem._next_tag += 1
        print(f"  LaundryItem() gives tag #{self.tag}")

    def describe(self):
        return f"item #{self.tag}"


class Garment(LaundryItem):
    def __init__(self, kind):
        super().__init__()  # Python tidak pernah menyisipkan ini untuk Anda
        self.kind = kind
        print(f'  Garment("{kind}")')

    def describe(self):
        return super().describe() + f", a {self.kind}"


class Shirt(Garment):
    def __init__(self, color, starch):
        super().__init__(color + " shirt")
        self.starch = starch
        print(f"  Shirt(..., starch={str(starch).lower()})")

    def describe(self):
        return super().describe() + (", starched" if self.starch else ", no starch")


class ForgetfulGarment(LaundryItem):
    def __init__(self, kind):
        self.kind = kind  # lupa super().__init__(): tag tidak pernah diisi


def main():
    print('-- new Shirt("blue", true) --')
    shirt = Shirt("blue", True)
    print(shirt.describe())
```

Kode 7.6 (Lanjutan) *constructor_chain.py: rantai yang sama dalam Python, ditambah sebuah kelas yang lupa memanggil super().__init__().*

```
    print('-- new Garment("towel") --')
    towel = Garment("towel")
    print(towel.describe())

    print("-- ForgetfulGarment(\"sock\") --")
    sock = ForgetfulGarment("sock")
    try:
        print(sock.describe())
    except AttributeError as e:
        print("  AttributeError:", e)


if __name__ == "__main__":
    main()
```

```
-- new Shirt("blue", true) --
  LaundryItem() gives tag #1
  Garment("blue shirt")
  Shirt(..., starch=true)
item #1, a blue shirt, starched
-- new Garment("towel") --
  LaundryItem() gives tag #2
  Garment("towel")
item #2, a towel
-- ForgetfulGarment("sock") --
  AttributeError: 'ForgetfulGarment' object has no attribute 'tag'
```

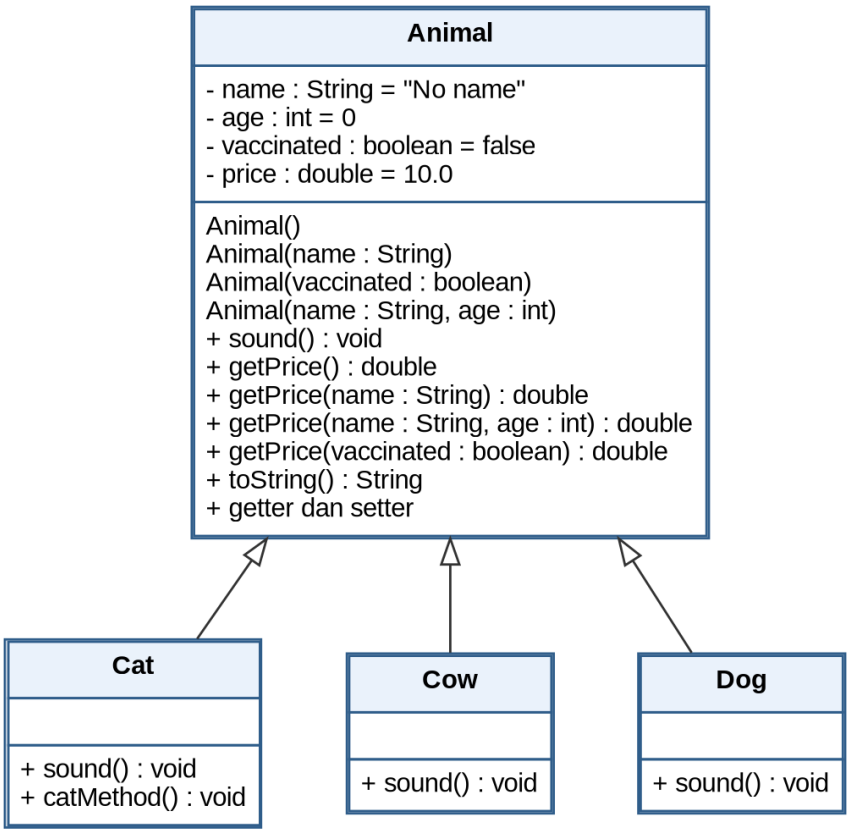
Dua blok pertama identik dengan milik Java, dan kodenya berpadanan satu lawan satu: `super().__init__(color + " shirt")` adalah `super(color + " shirt")` versi Python. Namun ada satu aturan yang berbeda, dan blok ketiga memperlihatkan akibatnya. **Python tidak pernah menyisipkan panggilan itu untuk Anda.** Bila sebuah subkelas mendefinisikan `__init__` dan tidak memanggil `super().__init__(...)`, `__init__` milik superkelas sama sekali tidak berjalan. `ForgetfulGarment` membuat kesalahan itu: objeknya mendapat `kind` tetapi tidak pernah mendapat `tag`, dan program baru menemukan masalahnya belakangan, ketika `describe()` mencoba membaca `self.tag` dan memunculkan `AttributeError`. Galat itu muncul jauh dari penyebabnya, dan itulah yang menjadikannya salah satu *bug* pewarisan Python yang paling sering terjadi. Kebiasaan yang perlu dibentuk mudah saja: **baris pertama setiap `__init__` subkelas adalah `super().__init__(...)`**.

(Sebaliknya, bila sebuah subkelas Python sama sekali tidak mendefinisikan `__init__`, ia mewarisi `__init__` superkelasnya seperti metode lainnya. `Cat()` di subbab berikut bekerja dengan cara itu.)

7.6 Polimorfisme: Satu Panggilan, Banyak Perilaku

Kata *polimorfisme* berasal dari bahasa Yunani dan berarti “banyak bentuk”. Dalam pemrograman berorientasi objek, kata itu menggambarkan keadaan yang sudah beberapa kali kita lihat: banyak kelas, yang terhubung melalui pewarisan, masing-masing memberi bentuknya sendiri kepada metode yang sama, sehingga satu panggilan di dalam program dapat mengambil banyak bentuk ketika program berjalan. Pewarisan memungkinkan sebuah kelas *menerima* atribut dan metode dari kelas lain; polimorfisme memungkinkannya *memakai* metode-metode itu untuk melakukan tindakan yang sama dengan caranya sendiri.

Contoh klasiknya adalah superkelas `Animal` dengan metode `sound()`, dan subkelas `Cat`, `Cow`, serta `Dog` yang masing-masing mengeluarkan suaranya sendiri. **Gambar 7.3** memperlihatkan rancangannya, dan **Kode 7.7** programnya secara lengkap. Program ini layak dipelajari pelan-pelan, karena ia menempatkan dua fitur yang paling sering tertukar oleh pemula — *overloading* dan *overriding* — berdampingan di dalam satu kelas.



Gambar 7.3 Diagram kelas UML program Animal. `Animal` memiliki empat konstruktor yang di-*overload* dan empat metode `getPrice` yang di-*overload*; `Cat`, `Cow`, dan `Dog` masing-masing meng-*override* `sound()`, dan `Cat` menambahkan satu metode miliknya sendiri. Segitiga kosong menunjuk dari subkelas ke superkelasnya; `-` menandai anggota private dan `+` anggota public.

Kode 7.7 `animal/AnimalTest.java`: `Animal` dengan konstruktor dan metode `getPrice` yang di-*overload*, tiga subkelas yang meng-*override* `sound()`, dan program uji yang menjalankan *overloading* maupun *overriding*.

```
package animal;

public class AnimalTest {
    public static void main(String[] args) {
        Animal animal = new Animal();  // Membuat objek Animal
        // Overloading
        Animal animal2 = new Animal("Bibi");  // Dengan nama
        Animal animal3 = new Animal(true);    // Dengan status vaksinasi
        Animal animal4 = new Animal("Chiki", 3);  // Dengan nama & umur
        System.out.println();

        Animal cat = new Cat();  // Membuat objek Cat
        Animal cow = new Cow();  // Membuat objek Cow
        Animal dog = new Dog();  // Membuat objek Dog
        System.out.println();

        // Polimorfisme
        animal.sound();
        cat.sound();
        cow.sound();
        dog.sound();
        System.out.println();

        // Overloading
        System.out.println(animal);
        System.out.println("Age: " + animal.getAge());
        System.out.println("Price: " + animal.getPrice());
        System.out.println("Vaccinated: " + animal.isVaccinated());
        System.out.println();

        System.out.println(animal2);
        System.out.println("Age: " + animal2.getAge());
        System.out.println("Price: " + animal2.getPrice(animal2.getName()));
        System.out.println("Vaccinated: " + animal2.isVaccinated());
        System.out.println();

        System.out.println(animal3);
        System.out.println("Age: " + animal3.getAge());
        System.out.println("Price: " + animal3.getPrice(animal3.isVaccinated()));
        System.out.println("Vaccinated: " + animal3.isVaccinated());
        System.out.println();
```

Kode 7.7 (Lanjutan) *animal/AnimalTest.java: Animal dengan konstruktor dan metode getPrice yang di-overload, tiga subkelas yang meng-override sound(), dan program uji yang menjalankan overloading maupun overriding.*

```
        System.out.println(animal4);
        System.out.println("Age: " + animal4.getAge());
        System.out.println("Price: " + animal4.getPrice(animal4.getName(),
                animal4.getAge()));
        System.out.println("Vaccinated: " + animal4.isVaccinated());
        System.out.println();

        System.out.print("Showing that a superclass can call ");
        System.out.println("a subclass method");
        System.out.println("---------------------------------------------------");
        System.out.println("a1 is defined by superclass: Animal, instantiated by
superclass: Animal");
        Animal a1 = new Animal();
        System.out.println("b1 is defined by subclass: Cat, instantiated by subclass:
Cat");
        Cat b1 = new Cat();
        System.out.println("b1 calls the Cat's method: catMethod()");
        b1.catMethod();
        // System.out.println("Try: a1 calls the subclass' method: catMethod() by casting
to Cat");
        // ((Cat) a1).catMethod(); // Eksepsi akan terjadi
        System.out.println("c1 is defined by superclass: Animal, instantiated by subclass:
Cat");
        Animal c1 = new Cat();
        // c1.catMethod(); // Metode subkelas: catMethod() tidak terdefinisi untuk tipe
Animal
        ((Cat) c1).catMethod();  // Perlu cast, agar instans bertipe superkelas dapat
memanggil
                                 // metode milik subkelas
    }
}

class Animal {
    // Penyembunyian informasi = enkapsulasi
    // Properti kelas ini dibuat private -> tidak dapat diakses
    // secara langsung. Hanya dapat diakses lewat getter & setter
    private String name = "No name";  // Nama bawaan: No name
    private int age = 0;  // Umur bawaan: 0
    private boolean vaccinated = false;  // Bawaan: false
    private double price = 10.0;  // Bawaan: 10.0

    // Overloading: sebuah kelas memiliki lebih dari satu metode
    // bernama sama & parameternya berbeda

    // Konstruktor: kasus khusus fungsi, yang namanya
    // sama dengan nama kelasnya
    Animal() {  // Overloading
        System.out.println("Animal: Default constructor");
    }
```

Kode 7.7 (Lanjutan) *animal/AnimalTest.java: Animal dengan konstruktor dan metode getPrice yang di-overload, tiga subkelas yang meng-override sound(), dan program uji yang menjalankan overloading maupun overriding.*

```
    Animal(String name) {  // Overloading
        this.name = name;
        System.out.println("Constructor with name");
    }

    Animal(boolean vaccinated) {  // Overloading
        this.vaccinated = vaccinated;
        System.out.println("Constructor with vaccinated status");
    }

    Animal(String name, int age) {  // Overloading
        this.name = name;
        this.age = age;
        System.out.println("Constructor with name & age");
    }

    // Overriding: nama metode sama & parameter sama
    // Dengan overriding, polimorfisme waktu jalan dapat dicapai
    public void sound() {  // Polimorfisme
        System.out.println("Animal makes a sound");
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public boolean isVaccinated() {
        return vaccinated;
    }

    public void setVaccinated(boolean vaccinated) {
        this.vaccinated = vaccinated;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }
```

Kode 7.7 (Lanjutan) *animal/AnimalTest.java: Animal dengan konstruktor dan metode getPrice yang di-overload, tiga subkelas yang meng-override sound(), dan program uji yang menjalankan overloading maupun overriding.*

```
    public double getPrice() {  // Overloading
        return price;
    }

    public double getPrice(String name) {  // Overloading
        if (name.equals("No name")) {
            return price + 100;
        } else {
            return price + 110;
        }
    }

    public double getPrice(String name, int age) {  // Overloading
        if (name.equals("No name")) {
            if (age > 1) {
                return price + 200;
            } else {
                return price + 150;
            }
        } else {
            if (age > 1) {
                return price + 300;
            } else {
                return price + 200;
            }
        }
    }

    public double getPrice(boolean vaccinated) {  // Overloading
        if (vaccinated) {
            return price + 400;
        } else {
            return price;
        }
    }

    public void setPrice(double price) {
        this.price = price;
    }

    @Override
    public String toString() {  // Dipakai saat mencetak sebuah instans
        return "[" + name + "]";
    }
}
```

Kode 7.7 (Lanjutan) *animal/AnimalTest.java: Animal dengan konstruktor dan metode getPrice yang di-overload, tiga subkelas yang meng-override sound(), dan program uji yang menjalankan overloading maupun overriding.*

```
class Cat extends Animal {
    // Overriding: nama metode sama & parameter sama
    // Dengan overriding, polimorfisme waktu jalan dapat dicapai
    @Override
    public void sound() {  // Polimorfisme
        System.out.println("Cat says: meow");
    }

    // Contoh metode/fungsi milik Cat
    public void catMethod() {
        System.out.println("Cat: catMethod");
    }
}

class Cow extends Animal {
    @Override
    public void sound() {  // Polimorfisme
        System.out.println("Cow says: moo");
    }
}

class Dog extends Animal {
    @Override
    public void sound() {  // Polimorfisme
        System.out.println("Dog says: woof");
    }
}
```

Kompilasi dan jalankan dari folder `java` dengan `javac -d out animal/AnimalTest.java` dan `java -cp out animal.AnimalTest`:

```
Animal: Default constructor
Constructor with name
Constructor with vaccinated status
Constructor with name & age

Animal: Default constructor
Animal: Default constructor
Animal: Default constructor

Animal makes a sound
Cat says: meow
Cow says: moo
Dog says: woof

[No name]
Age: 0
Price: 10.0
Vaccinated: false

[Bibi]
Age: 0
Price: 120.0
Vaccinated: false

[No name]
Age: 0
Price: 410.0
Vaccinated: true

[Chiki]
Age: 3
Price: 310.0
Vaccinated: false

Showing that a superclass can call a subclass method
--------------------------------------------------
a1 is defined by superclass: Animal, instantiated by superclass: Animal
Animal: Default constructor
b1 is defined by subclass: Cat, instantiated by subclass: Cat
Animal: Default constructor
b1 calls the Cat's method: catMethod()
Cat: catMethod
c1 is defined by superclass: Animal, instantiated by subclass: Cat
Animal: Default constructor
Cat: catMethod
```

7.6.1 Membaca kelas Animal

- **Field yang terenkapsulasi.** Keempat field bersifat `private`: tidak ada kode di luar `Animal` yang boleh menyentuhnya. Masing-masing punya nilai bawaan — `"No name"`, `0`, `false`, `10.0` — yang berlaku kecuali diubah oleh konstruktor. Dunia luar membaca dan mengubahnya hanya melalui metode *getter* dan *setter* publik di bagian bawah (`getName`, `setName`, dan seterusnya). Menyembunyikan data di balik metode dengan cara ini disebut *enkapsulasi*, atau *penyembunyian informasi*, dan Subbab 7.14 menjelaskan mengapa hal itu penting.
- **Konstruktor yang di-overload.** `Animal` memiliki empat konstruktor: tanpa argumen, dengan sebuah `String`, dengan sebuah `boolean`, serta dengan sebuah `String` dan sebuah `int`. Namanya sama dan daftar parameternya berbeda: *overloading*, seperti di Bab 6. Kompilator memilih konstruktor yang parameternya cocok dengan argumen, sehingga `new Animal("Bibi")` memanggil `Animal(String name)` dan `new Animal(true)` memanggil `Animal(boolean vaccinated)`. Empat baris pertama keluaran adalah keempat konstruktor itu yang mengumumkan dirinya.

- **`getPrice` yang di-overload.** Cara yang sama menghasilkan empat metode `getPrice`. Harga dasarnya 10.0; versi yang menerima nama menambahkan 100 untuk hewan tanpa nama dan 110 untuk hewan bernama; versi yang menerima nama dan umur menambahkan antara 150 dan 300; versi yang menerima penanda vaksinasi menambahkan 400 bila hewan itu sudah divaksin. Periksalah keluarannya: Bibi, yang bernama, seharga $10 + 110 = 120$; hewan yang divaksin seharga $10 + 400 = 410$; Chiki, bernama dan berumur tiga tahun, seharga $10 + 300 = 310$.
- **`sound()` memang dimaksudkan untuk di-override.** `Animal.sound()` mencetak baris yang umum. Setiap subkelas menggantinya. Itulah *overriding*.
- **`toString()`** juga sebuah *override* — dari metode yang diwarisi setiap kelas Java dari `Object` (Subbab 7.8). `System.out.println(animal)` memanggilnya secara otomatis, itulah sebabnya keluaran memperlihatkan `[No name]` dan `[Bibi]`.

Ketiga subkelasnya mungil. `Cat`, `Cow`, dan `Dog` masing-masing mendeklarasikan `extends Animal` dan meng-*override* `sound()`. Tidak satu pun mendeklarasikan konstruktor, sehingga masing-masing mendapat *konstruktor bawaan* dari kompilator — yang tidak menerima argumen dan hanya memanggil `super()`. Itulah sebabnya pembuatan kucing, sapi, dan anjing mencetak `Animal: Default constructor` tiga kali: satu-satunya konstruktor yang mencetak sesuatu adalah `Animal()`, yang dicapai lewat `super()` yang tak terlihat. `Cat` juga menambahkan `catMethod()`, metode yang tidak dimiliki `Animal`.

7.6.2 Membaca program uji

Blok yang diberi komentar `// Polimorfisme` adalah inti program ini. Keempat variabel `animal`, `cat`, `cow`, dan `dog` semuanya dideklarasikan bertipe `Animal`. Keempat panggilan tampak identik — `animal.sound()`, `cat.sound()`, dan seterusnya — namun mencetak empat baris yang berbeda. Kompilator tidak memutuskan `sound()` mana yang dijalankan; ia tidak mungkin melakukannya, karena yang ia ketahui hanyalah bahwa setiap variabel adalah `Animal`. Keputusan itu diambil saat program berjalan dengan melihat objek yang sebenarnya. Itulah **polimorfisme waktu jalan** (*run-time polymorphism*), dan subbab berikut membongkarnya.

Blok terakhir memperlihatkan sisi lain mata uang itu. `a1` dideklarasikan `Animal` dan memegang `Animal`; `b1` dideklarasikan `Cat` dan memegang `Cat`, sehingga `b1.catMethod()` tidak bermasalah. `c1` dideklarasikan `Animal` tetapi memegang `Cat`. Bolehkah kita menulis `c1.catMethod()`? Objeknya jelas memiliki metode itu. Namun hapuslah `//` di depan baris itu dan kompilator menolak:

```
animal/AnimalTest.java:63: error: cannot find symbol
        c1.catMethod(); // The subclass' method: catMethod() is undefined for the type
Animal
          ^
  symbol:   method catMethod()
  location: variable c1 of type Animal
1 error
```

Kompilator bernalar dari tipe yang dideklarasikan, `Animal`, dan `Animal` tidak punya `catMethod`. Untuk memanggilnya kita harus melakukan *cast*: `((Cat) c1).catMethod()` berkata kepada kompilator "percayalah, `Animal` ini sebenarnya `Cat`". *Cast* itu diperiksa ketika program berjalan. Bila kita berbohong — seperti pada baris yang dijadikan komentar `((Cat) a1).catMethod()`, dengan `a1` sebuah `Animal` biasa — program berhenti dengan `ClassCastException`. Subbab 7.7 memperlihatkan cara yang lebih aman untuk menulis *cast* semacam itu.

7.6.3 Overloading versus overriding

Kedua kata itu terdengar mirip dan sama-sama melibatkan "beberapa metode dengan satu nama". **Tabel 7.1** menyandingkan keduanya.

Tabel 7.1 Perbandingan overloading dan overriding.

	Overloading	Overriding
Di mana	Dalam satu kelas (atau sebuah kelas dan subkelasnya)	Di subkelas, menggantikan metode superkelas
Nama sama?	Ya	Ya
Tipe parameter sama?	**Tidak** — harus berbeda	**Ya** — harus cocok
Siapa yang memilih metodenya	Kompilator, dari tipe argumen yang dideklarasikan	Program yang berjalan, dari kelas objek yang sebenarnya
Kapan dipilih	Saat kompilasi (*static binding*)	Saat program berjalan (*dynamic binding*)
Contoh di **Kode 7.7**	Empat konstruktor `Animal(...)`, empat `getPrice(...)`	`sound()` di `Cat`, `Cow`, `Dog`; `toString()` di `Animal`
Python	Tidak didukung: `def` kedua dengan nama sama menggantikan yang pertama; pakai argumen bawaan atau argumen kata kunci	Didukung: nama yang sama di subkelas

Baris terakhir patut ditekankan: Python **tidak punya *overloading***. `def get_price` kedua di dalam kelas yang sama hanya menggantikan yang pertama. Pemrogram Python memperoleh keluwesan yang sama dari argumen bawaan dan argumen kata kunci, seperti diperlihatkan **Kode 7.8**.

Kode 7.8 `animal_test.py`: program Animal dalam Python; satu `__init__` dan satu `get_price` dengan parameter kata kunci opsional masing-masing menggantikan empat *overload* Java.

```python
"""animal_test.py: Animal, Cat, Cow dan Dog; overriding, dan jawaban Python atas
overloading."""


class Animal:
    # Satu __init__ dengan parameter opsional menggantikan empat konstruktor Java.
    def __init__(self, name=None, age=None, vaccinated=None):
        self._name = "No name"   # Nama bawaan: No name
        self._age = 0            # Umur bawaan: 0
        self._vaccinated = False  # Bawaan: False
        self._price = 10.0       # Bawaan: 10.0
        if name is not None and age is not None:
            self._name, self._age = name, age
            print("Constructor with name & age")
        elif name is not None:
            self._name = name
            print("Constructor with name")
        elif vaccinated is not None:
            self._vaccinated = vaccinated
            print("Constructor with vaccinated status")
        else:
            print("Animal: Default constructor")

    # Overriding: nama metode sama & parameter sama
    def sound(self):
        print("Animal makes a sound")

    @property
    def name(self):
        return self._name

    @property
    def age(self):
        return self._age

    @property
    def vaccinated(self):
        return self._vaccinated
```

Kode 7.8 (Lanjutan) *animal_test.py: program Animal dalam Python; satu __init__ dan satu get_price dengan parameter kata kunci opsional masing-masing menggantikan empat overload Java.*

```
    # Satu metode dengan parameter kata kunci menggantikan empat overload getPrice di Java.
    def get_price(self, name=None, age=None, vaccinated=None):
        if name is not None and age is not None:
            if name == "No name":
                return self._price + (200 if age > 1 else 150)
            return self._price + (300 if age > 1 else 200)
        if name is not None:
            return self._price + (100 if name == "No name" else 110)
        if vaccinated is not None:
            return self._price + 400 if vaccinated else self._price
        return self._price

    def __str__(self):  # Dipakai saat mencetak sebuah instans
        return "[" + self._name + "]"


class Cat(Animal):
    def sound(self):  # Polimorfisme
        print("Cat says: meow")

    def cat_method(self):  # Contoh metode milik Cat sendiri
        print("Cat: catMethod")


class Cow(Animal):
    def sound(self):
        print("Cow says: moo")


class Dog(Animal):
    def sound(self):
        print("Dog says: woof")


def show(animal, price):
    print(animal)
    print("Age:", animal.age)
    print("Price:", price)
    print("Vaccinated:", str(animal.vaccinated).lower())
    print()
```

Kode 7.8 (Lanjutan) *animal_test.py: program Animal dalam Python; satu __init__ dan satu get_price dengan parameter kata kunci opsional masing-masing menggantikan empat overload Java.*

```
def main():
    animal = Animal()
    animal2 = Animal(name="Bibi")
    animal3 = Animal(vaccinated=True)
    animal4 = Animal(name="Chiki", age=3)
    print()

    cat, cow, dog = Cat(), Cow(), Dog()
    print()

    for a in (animal, cat, cow, dog):  # Polimorfisme
        a.sound()
    print()

    show(animal, animal.get_price())
    show(animal2, animal2.get_price(name=animal2.name))
    show(animal3, animal3.get_price(vaccinated=animal3.vaccinated))
    show(animal4, animal4.get_price(name=animal4.name, age=animal4.age))

    print("A variable has no declared type: only the object decides")
    print("-----------------------------------------------------")
    a1 = Animal()
    c1 = Cat()
    c1.cat_method()  # tidak perlu cast: objeknya memang Cat
    try:
        a1.cat_method()  # objeknya Animal biasa
    except AttributeError as e:
        print("AttributeError:", e)


if __name__ == "__main__":
    main()
```

```
Animal: Default constructor
Constructor with name
Constructor with vaccinated status
Constructor with name & age

Animal: Default constructor
Animal: Default constructor
Animal: Default constructor

Animal makes a sound
Cat says: meow
Cow says: moo
Dog says: woof

[No name]
Age: 0
Price: 10.0
Vaccinated: false

[Bibi]
Age: 0
Price: 120.0
Vaccinated: false

[No name]
Age: 0
Price: 410.0
Vaccinated: true

[Chiki]
Age: 3
Price: 310.0
Vaccinated: false

A variable has no declared type: only the object decides
--------------------------------------------------
Animal: Default constructor
Animal: Default constructor
Cat: catMethod
AttributeError: 'Animal' object has no attribute 'cat_method'
```

Semua keluaran sampai blok terakhir sama persis dengan keluaran Java. `__init__` yang tunggal menerima tiga parameter opsional, masing-masing bernilai bawaan `None` (“tidak diberikan”), dan memutuskan konstruktor Java mana yang ditirunya dari argumen yang datang. Tempat pemanggilannya menjelaskan dirinya sendiri — `Animal(vaccinated=True)` berkata lebih banyak daripada `new Animal(true)` di Java. `get_price` bekerja dengan cara yang sama. *Getter*-nya menjadi *property* (`@property`), yang memungkinkan pemanggil menulis `animal.age` alih-alih `animal.get_age()`; Subbab 7.14 menjelaskannya.

Blok terakhir adalah tempat kedua bahasa berpisah jalan. Variabel Python tidak punya tipe yang dideklarasikan, sehingga tidak ada yang perlu di-*cast*: `c1.cat_method()` langsung berjalan, karena objeknya memang `Cat`. Dan `a1.cat_method()`, yang di Java bahkan tidak lolos kompilasi, di Python dijalankan dan gagal dengan `AttributeError` tepat pada saat pemanggilan. Semboyan Python untuk gaya ini adalah “bila ia berjalan seperti bebek dan berkuwek seperti bebek, ia adalah bebek” — *duck typing*: satu-satunya pertanyaan adalah apakah objek itu memiliki metodenya, bukan tipe apa yang dideklarasikan untuk variabelnya.

7.7 Cara Kerja Dynamic Dispatch yang Sebenarnya: Gambaran Memori

Sudah beberapa kali kita katakan bahwa "objek yang sebenarnya menentukan metode mana yang berjalan". Kalimat itu benar tetapi samar, dan kesamaran adalah tempat berkembang biaknya salah paham. Di subbab ini kita membuatnya tepat, pertama di Java lalu di Python, dengan menggambar apa yang ada di dalam memori komputer selama program berjalan.

7.7.1 Dua tipe untuk setiap variabel

Di Java setiap ekspresi memiliki dua tipe, dan keduanya harus benar-benar dibedakan:

> **DEFINISI. Tipe statis** (*static type*, atau *tipe yang dideklarasikan*) sebuah variabel adalah tipe yang tertulis dalam deklarasinya; tipe ini tetap sejak program dikompilasi. **Tipe dinamis** (*dynamic type*, atau *tipe waktu jalan*) adalah kelas objek yang dirujuk variabel itu pada suatu saat ketika program berjalan. Tipe dinamis selalu sama dengan tipe statis atau salah satu subkelasnya.

Setelah `Machine m = new Washer("W-01");`, tipe statis `m` adalah `Machine` dan tipe dinamisnya `Washer`. Memberikan objek subkelas kepada variabel superkelas, seperti di sini, disebut **upcasting**; ini selalu aman dan terjadi otomatis, karena `Washer` adalah sejenis `Machine`. Arah sebaliknya — memperlakukan variabel `Machine` sebagai `Washer` — disebut **downcasting**; ini tidak otomatis, dan tidak selalu aman.

Kompilator dan program yang berjalan memakai kedua tipe itu untuk tugas yang berbeda:

- **Kompilator memakai tipe statis** untuk memutuskan apakah sebuah panggilan *diizinkan*. `m.start()` lolos kompilasi karena `Machine` memiliki `start()`. `m.setHeat(3)` tidak akan lolos, sekalipun `m` memegang `Dryer`, karena `Machine` tidak punya `setHeat`.
- **Program yang berjalan memakai tipe dinamis** untuk memutuskan metode *override mana* yang dijalankan. `m.start()` menjalankan `Washer.start()` karena objeknya `Washer`.

Pembagian kerja ini punya nama: **dynamic dispatch** (pengiriman dinamis, juga disebut *late binding*). Ia berlaku untuk *metode instans yang dapat di-override*. Ia **tidak** berlaku untuk field, metode `static`, atau metode `private`, yang semuanya dipilih kompilator dari tipe statis. **Kode 7.9** memperlihatkan setiap kasus dengan hierarki kecil mesin binatu.

Kode 7.9 `DispatchDemo.java`: *dynamic dispatch* pada `start()`, tidak adanya *dispatch* untuk field yang tersembunyi dan metode statis yang tersembunyi, *downcast* yang aman dengan `instanceof`, dan *downcast* yang tidak aman.

```
public class DispatchDemo {
    public static void main(String[] args) {
        Machine m = new Washer("W-01");  // tipe statis Machine, tipe dinamis Washer
        m.start();                       // dipilih saat program berjalan: Washer.start()
        System.out.println("m.label  = " + m.label);   // field: dipilih saat kompilasi
        System.out.println("m.kind() = " + m.kind());  // static: dipilih saat kompilasi
        System.out.println("m.describe() = " + m.describe());
        System.out.println();

        Machine[] floor = {new Washer("W-02"), new Dryer("D-01"), new Machine("M-00")};
        for (Machine x : floor) {
            x.start();  // satu baris kode, tiga perilaku berbeda
        }
        System.out.println();

        for (Machine x : floor) {
            if (x instanceof Dryer d) {  // periksa dulu, lalu pakai sebagai Dryer
                d.setHeat(3);
            }
        }
        try {
            Dryer wrong = (Dryer) floor[0];  // floor[0] adalah Washer
            wrong.setHeat(1);
        } catch (ClassCastException e) {
            System.out.println("Casting a Washer to Dryer: " +
e.getClass().getSimpleName());
        }
        System.out.println("floor[1] is really a " + floor[1].getClass().getSimpleName());
    }
}

class Machine {
    protected final String id;
    String label = "machine";

    Machine(String id) {
        this.id = id;
    }

    void start() {
        System.out.println(id + ": generic machine starts");
    }

    static String kind() {
        return "Machine.kind()";
    }
```

Kode 7.9 (Lanjutan) *DispatchDemo.java: dynamic dispatch pada start(), tidak adanya dispatch untuk field yang tersembunyi dan metode statis yang tersembunyi, downcast yang aman dengan instanceof, dan downcast yang tidak aman.*

```
    String describe() {
        return id + " is a " + label;  // membaca field label milik Machine
    }
}

class Washer extends Machine {
    String label = "washer";  // menyembunyikan Machine.label; BUKAN meng-override-nya

    Washer(String id) {
        super(id);
    }

    @Override
    void start() {
        System.out.println(id + ": fill water, spin at 1200 rpm");
    }

    static String kind() {  // menyembunyikan Machine.kind(); metode static tidak meng-
override
        return "Washer.kind()";
    }
}

class Dryer extends Machine {
    private int heat = 2;

    Dryer(String id) {
        super(id);
    }

    @Override
    void start() {
        System.out.println(id + ": tumble with heat level " + heat);
    }

    void setHeat(int heat) {
        this.heat = heat;
        System.out.println(id + ": heat set to " + heat);
    }
}
```

```
W-01: fill water, spin at 1200 rpm
m.label  = machine
m.kind() = Machine.kind()
m.describe() = W-01 is a machine

W-02: fill water, spin at 1200 rpm
D-01: tumble with heat level 2
M-00: generic machine starts

D-01: heat set to 3
Casting a Washer to Dryer: ClassCastException
floor[1] is really a Dryer
```

Telusurilah blok pertama, yang berisi tiga kejutan bagi yang lengah.

1. `m.start()` mencetak baris milik *mesin cuci*: *dynamic dispatch*, sesuai harapan.
2. `m.label` mencetak `machine`, bukan `washer`. `Washer` mendeklarasikan field miliknya sendiri bernama `label`, tetapi field yang namanya sama dengan field superkelas tidak *meng-override*-nya; ia *menyembunyikan*-nya. Objek itu sebenarnya memuat **dua** field `label`, satu dari setiap kelas, dan kompilator memilih salah satunya berdasarkan tipe statis ekspresinya, di sini

`Machine`. Bahkan di dalam `describe()` — metode milik `Machine` — `label` berarti field milik `Machine`, itulah sebabnya `describe()` berkata `W-01 is a machine`. Field yang tersembunyi hampir selalu merupakan kesalahan; obatnya adalah jangan pernah mendeklarasikan ulang field superkelas di dalam subkelas.

3. `m.kind()` mencetak `Machine.kind()`. `kind` bersifat `static`: ia milik kelas, bukan milik objek mana pun, sehingga tidak ada objek yang menjadi dasar *dispatch*, dan kompilator mengikat panggilan itu pada tipe statisnya. (Memanggil metode statis melalui variabel memang sah tetapi menyesatkan; `javac -Xlint` memperingatkan "static method should be qualified by type name", dan Anda pun sebaiknya demikian.)

Blok kedua adalah polimorfisme yang sedang menjalankan pekerjaan sehari-harinya. Larik `floor` memiliki tipe elemen statis `Machine`, tetapi memuat sebuah `Washer`, sebuah `Dryer`, dan sebuah `Machine` biasa. Badan perulangannya hanya satu baris, `x.start()`, dan menghasilkan tiga perilaku. Menambahkan jenis mesin keempat tahun depan tidak akan menuntut perubahan pada perulangan ini.

Blok ketiga memperlihatkan *downcasting*. Kita ingin mengatur panas setiap mesin pengering, tetapi `setHeat` hanya ada di `Dryer`. Uji `x instanceof Dryer d` bertanya "apakah tipe dinamis `x` adalah `Dryer` (atau subkelasnya)?", dan bila ya, ia mendeklarasikan variabel baru `d` bertipe statis `Dryer` yang merujuk objek yang sama. Di dalam `if`, `d.setHeat(3)` lolos kompilasi dan aman. Bentuk gabungan uji-dan-*cast* ini, yang disebut *pattern matching for* `instanceof`, sudah menjadi standar sejak Java 16. Blok `try` melakukan hal yang tidak boleh Anda lakukan: ia meng-*cast* `floor[0]` ke `Dryer` tanpa memeriksa. Objeknya `Washer`, *cast* itu gagal saat program berjalan, dan Java melempar `ClassCastException`. Terakhir, `getClass()` — metode yang diwarisi setiap objek dari `Object` — melaporkan tipe dinamisnya.

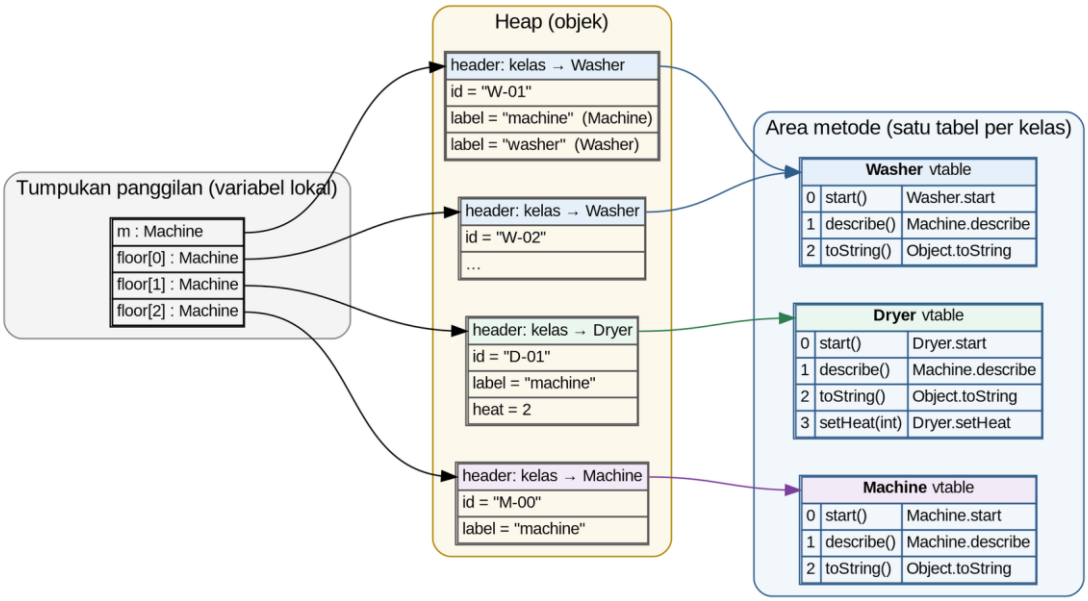
Tabel 7.2 merangkum anggota mana dipilih oleh tipe yang mana.

Tabel 7.2 Apa yang diputuskan kapan: hanya metode instans yang dapat di-override yang di-dispatch berdasarkan tipe dinamis.

Anggota	Dipilih oleh	Pada saat	Di
Metode instans yang dapat di-override	Tipe dinamis objek	Program berjalan	`m.start()` menjalankan `Washer.start`
Metode `private`	Kelas yang mendeklarasikannya	Kompilasi	—
Metode `static`	Tipe statis ekspresi	Kompilasi	`m.kind()` menjalankan `Machine.kind`
Field (instans atau statis)	Tipe statis ekspresi	Kompilasi	`m.label` membaca `Machine.label`
Overload di antara beberapa metode	Tipe statis argumen	Kompilasi	keempat `getPrice` di **Kode 7.7**

7.7.2 Gambaran memori: tabel metode

Bagaimana program yang berjalan dapat menemukan `Washer.start` secepat itu? Ia tidak mampu menelusuri definisi kelas pada setiap panggilan; perulangan harga bisa saja memanggil `price()` jutaan kali. **Gambar 7.4** memperlihatkan gagasan yang dipakai mesin virtual Java dan kompilator C++.



Gambar 7.4 Memori selama **Kode 7.9** berjalan. Variabel lokal di tumpukan panggilan (*stack*) memegang referensi ke objek di *heap*; *header* setiap objek menunjuk ke kelasnya; setiap kelas memiliki satu tabel metode (*vtable*) yang slot ke-*k*-nya memuat kode metode ke-*k* yang dapat di-override. Subkelas menyalin slot-slot induknya dan menimpa slot yang di-override-nya.

Bacalah dari kiri ke kanan.

- **Tumpukan panggilan** (*stack*) memuat variabel lokal `main`: `m` dan elemen-elemen `floor`. Masing-masing adalah *referensi*, panah menuju sebuah objek; semuanya dideklarasikan `Machine`.
- ***Heap*** memuat objek-objeknya. Setiap objek diawali *header* kecil yang, di antara hal lain, berisi penunjuk ke kelas objek itu. Sesudah *header* datang field-field-nya — semuanya, termasuk yang dideklarasikan di superkelas. Lihatlah `Washer` yang pertama: ia membawa `id`, `label` yang dideklarasikan di `Machine`, *dan* `label` yang dideklarasikan di `Washer`. Itulah kejutan field tersembunyi yang dibuat kasatmata.
- **Area metode** memuat satu tabel per kelas, yang disebut **tabel metode virtual** atau **vtable**. Setiap metode yang dapat di-*override* diberi *nomor slot* yang tetap ketika kelas-kelas dimuat: dalam hierarki ini `start()` adalah slot 0, `describe()` slot 1, `toString()` slot 2. Tabel subkelas bermula sebagai salinan tabel superkelasnya; setiap metode yang di-*override* subkelas menggantikan isi slotnya, dan setiap metode baru ditambahkan di akhir. Jadi slot 0 milik `Washer` berisi `Washer.start`, sementara slot 1-nya masih berisi `Machine.describe` warisan. `Dryer` menimpa slot 0 dan menambahkan `setHeat` sebagai slot 3.

Kini panggilan `x.start()` dikompilasi menjadi tiga langkah tingkat mesin, dan tidak satu pun berupa pencarian:

$$\text{kode yang dijalankan} = \text{vtable}(\text{kelas}(x))[\,\text{slot}(\text{start})\,] \tag{7.1}$$

Ikuti referensi `x` sampai ke objeknya; ikuti *header* objek itu sampai ke vtable kelasnya; lompat ke kode di slot yang sudah ditetapkan kompilator sebelumnya. Persamaan (7.1) memerlukan beberapa pembacaan memori yang sama, entah hierarkinya berisi dua kelas atau dua ratus. Itulah sebabnya *dynamic dispatch* cukup murah untuk dipakai di mana-mana. (Spesifikasi Mesin Virtual Java menguraikan *perilaku* yang dituntut dari instruksi `invokevirtual`, bukan struktur data khusus ini [5]; JVM sungguhan memakai vtable, dan bahkan melangkah lebih jauh dengan

memperhatikan kapan sebuah tempat pemanggilan hanya pernah menjumpai satu kelas lalu memanggil metode itu secara langsung.)

Gambaran ini juga menjelaskan aturan-aturan **Tabel 7.2**. Sebuah field dibaca pada posisi (*offset*) tetap di dalam objek yang dipilih kompilator, sehingga pandangan statis kompilatorlah yang menentukan `label` mana yang dibaca. Metode statis atau private tidak pernah masuk ke vtable, sehingga dipanggil secara langsung. Dan pemeriksaan kompilator bahwa `m.setHeat(3)` tidak sah tidak lain adalah pengamatan bahwa tabel milik `Machine` tidak punya slot bernama `setHeat`.

7.7.3 Gagasan yang sama dalam Python

Python tidak punya tipe statis yang perlu dirisaukan, dan pencariannya diatur dengan cara lain, tetapi hasilnya sama. **Kode 7.10** mengulang percobaan itu.

Kode 7.10 `dispatch_demo.py`: pencarian dinamis di Python, dengan fungsi pembantu yang menelusuri urutan resolusi metode persis seperti yang dilakukan Python.

```
"""dispatch_demo.py: bagaimana Python menemukan metode yang dijalankan."""


class Machine:
    label = "machine"  # atribut kelas

    def __init__(self, machine_id):
        self.id = machine_id

    def start(self):
        print(f"{self.id}: generic machine starts")

    def describe(self):
        return f"{self.id} is a {self.label}"  # self.label dicari secara dinamis


class Washer(Machine):
    label = "washer"

    def start(self):
        print(f"{self.id}: fill water, spin at 1200 rpm")


class Dryer(Machine):
    def __init__(self, machine_id):
        super().__init__(machine_id)
        self.heat = 2

    def start(self):
        print(f"{self.id}: tumble with heat level {self.heat}")

    def set_heat(self, heat):
        self.heat = heat
        print(f"{self.id}: heat set to {heat}")


def where_is(cls, name):
    """Telusuri MRO persis seperti Python dan laporkan kelas yang mendefinisikan name."""
    for klass in cls.__mro__:
        if name in klass.__dict__:
            return klass.__name__
    return None
```

Kode 7.10 (Lanjutan) *dispatch_demo.py: pencarian dinamis di Python, dengan fungsi pembantu yang menelusuri urutan resolusi metode persis seperti yang dilakukan Python.*

```
def main():
    m: Machine = Washer("W-01")  # anotasi hanyalah petunjuk, bukan aturan
    m.start()
    print("m.label =", m.label)  # atribut juga dinamis, tidak seperti field Java
    print("m.describe() =", m.describe())
    print()

    floor = [Washer("W-02"), Dryer("D-01"), Machine("M-00")]
    for x in floor:
        x.start()  # satu baris kode, tiga perilaku berbeda
    print()

    for x in floor:
        if isinstance(x, Dryer):
            x.set_heat(3)
    try:
        floor[0].set_heat(1)  # Washer tidak punya set_heat
    except AttributeError as e:
        print("AttributeError:", e)
    print()

    print("Washer MRO:", [k.__name__ for k in Washer.__mro__])
    for name in ("start", "describe", "label", "set_heat"):
        print(f"  Washer.{name:8} found in {where_is(Washer, name)}")


if __name__ == "__main__":
    main()
```

```
W-01: fill water, spin at 1200 rpm
m.label = washer
m.describe() = W-01 is a washer

W-02: fill water, spin at 1200 rpm
D-01: tumble with heat level 2
M-00: generic machine starts

D-01: heat set to 3
AttributeError: 'Washer' object has no attribute 'set_heat'

Washer MRO: ['Washer', 'Machine', 'object']
  Washer.start    found in Washer
  Washer.describe found in Machine
  Washer.label    found in Washer
  Washer.set_heat found in None
```

Ketika Python mengevaluasi `m.start`, ia melakukan pencarian yang dapat ditiru fungsi pembantu `where_is` dalam beberapa baris. Setiap kelas memiliki kamus, `__dict__`, yang memetakan nama-nama yang didefinisikan *di dalam kelas itu* ke nilainya, dan sebuah tupel, `__mro__` (*method resolution order*, urutan resolusi metode), yang mendaftar kelas itu, superkelasnya, dan seterusnya sampai `object`. Python mula-mula memeriksa kamus milik objek itu sendiri (untuk metode, pemeriksaan ini tidak menemukan apa-apa), lalu menelusuri `type(m).__mro__` dan mengambil kamus kelas pertama yang memuat nama itu. Untuk `Washer` urutannya `Washer`, `Machine`, `object`; `start` ditemukan di `Washer`, `describe` di `Machine`. Tidak ada nomor slot dan tidak ada tabel yang disalin: "tabel"-nya adalah rantai kamus yang ditelusuri berurutan, dengan *cache* sehingga pencarian itu jarang diulang.

Dua akibatnya berbeda dari Java. Pertama, `m.label` mencetak `washer`: di Python `label` adalah *atribut kelas*, yang ditemukan melalui penelusuran MRO yang sama seperti metode, sehingga nilai

milik subkelas yang menang dan `describe()` pun melihatnya. Python tidak punya jebakan field tersembunyi. Kedua, tidak ada yang diperiksa sebelum program berjalan: `floor[0].set_heat(1)` baru gagal ketika dieksekusi, dengan `AttributeError`. `isinstance(x, Dryer)` memainkan peran `instanceof` di Java, tetapi tidak diikuti *cast*: `x.set_heat(3)` berjalan karena objeknya memiliki metode itu. Anotasi `m: Machine` adalah *petunjuk* bagi pembaca dan alat pemeriksa; interpreter mengabaikannya.

> **WAWASAN KUNCI:** Java menjawab "bolehkah saya memanggil ini?" saat kompilasi berdasarkan tipe *statis*, dan "versi mana yang berjalan?" saat program berjalan berdasarkan tipe *dinamis*. Python menjawab kedua pertanyaan itu saat program berjalan, berdasarkan objeknya saja. Java menangkap lebih banyak kesalahan sebelum program berjalan; Python memungkinkan Anda menulis lebih sedikit. Tidak satu pun jawaban itu mengubah fakta pokoknya: **objeknya, bukan kode yang memanggilnya, yang menentukan metode mana yang berjalan**.

7.8 Akar Setiap Hierarki: `Object` dan `object`

Setiap kelas Java yang tidak menyatakan `extends` memperluas `java.lang.Object`, sehingga, langsung maupun tidak langsung, *setiap* kelas adalah subkelas `Object`. Di Python setiap kelas mewarisi `object`. Itulah sebabnya `Animal` dapat meng-*override* `toString()` tanpa mewarisi apa pun yang tampak, dan mengapa vtable di **Gambar 7.4** berakhir dengan `Object.toString`. Kelas-kelas akar itu menyediakan segelintir metode yang dimiliki setiap objek; tiga di antaranya penting setiap hari, dan ketiganya memang dimaksudkan untuk di-*override*:

- **Teks.** `toString()` di Java serta `__str__()` (untuk pengguna) dan `__repr__()` (untuk pemrogram) di Python menghasilkan teks yang tampil ketika sebuah objek dicetak. Versi warisannya mencetak nama kelas dan sebuah nomor identitas, misalnya `PlainCustomer@1b6d3586`; berguna bagi mesin, tak berguna bagi manusia.
- **Kesamaan.** `equals(Object)` di Java dan `__eq__` di Python memutuskan apakah dua objek *sama*. Versi warisannya berkata "sama hanya bila keduanya objek yang itu-itu juga", yang juga ditawarkan Java sebagai `==` dan Python sebagai `is`.
- **Hash.** `hashCode()` di Java dan `__hash__` di Python mengembalikan bilangan bulat yang dipakai tabel hash (`HashSet`, `HashMap`, serta `set` dan `dict` di Python) untuk memutuskan di mana sebuah objek disimpan. Nilainya harus selaras dengan kesamaan.

Kode 7.11 memperlihatkan mengapa perilaku bawaan itu biasanya keliru bagi kelas yang menggambarkan benda di dunia nyata, misalnya pelanggan binatu yang dikenali dari nomor teleponnya.

Kode 7.11 `ObjectRoot.java`: kelas yang mewarisi `toString`, `equals`, dan `hashCode` dari `Object`, dan kelas yang meng-*override* ketiganya.

```
import java.util.HashSet;
import java.util.Objects;
import java.util.Set;

public class ObjectRoot {
    public static void main(String[] args) {
        PlainCustomer p1 = new PlainCustomer("Siti Aminah", "081234567890");
        PlainCustomer p2 = new PlainCustomer("Siti Aminah", "081234567890");
        System.out.println("PlainCustomer text starts with 'PlainCustomer@'? "
                + p1.toString().startsWith("PlainCustomer@"));
        System.out.println("p1.equals(p2) = " + p1.equals(p2));

        Customer c1 = new Customer("Siti Aminah", "081234567890");
        Customer c2 = new Customer("Siti A.", "081234567890");  // nomor telepon sama
        Customer c3 = new Customer("Budi Santoso", "085712340000");
        System.out.println(c1);
        System.out.println("c1.equals(c2) = " + c1.equals(c2));
        System.out.println("c1 == c2      = " + (c1 == c2));

        Set<PlainCustomer> plain = new HashSet<>();
        plain.add(p1);
        plain.add(p2);
        Set<Customer> real = new HashSet<>();
        real.add(c1);
        real.add(c2);
        real.add(c3);
        System.out.println("distinct plain customers: " + plain.size());
        System.out.println("distinct customers:       " + real.size());
    }
}

class PlainCustomer {  // mewarisi toString, equals, hashCode dari Object
    private final String name;
    private final String phone;

    PlainCustomer(String name, String phone) {
        this.name = name;
        this.phone = phone;
    }
}

class Customer {
    private final String name;
    private final String phone;  // nomor telepon mengidentifikasi pelanggan
```

Kode 7.11 (Lanjutan) *ObjectRoot.java: kelas yang mewarisi toString, equals, dan hashCode dari Object, dan kelas yang meng-override ketiganya.*

```
    Customer(String name, String phone) {
        this.name = name;
        this.phone = phone;
    }

    @Override
    public String toString() {
        return "Customer[" + name + ", " + phone + "]";
    }

    @Override
    public boolean equals(Object other) {
        if (this == other) {
            return true;
        }
        if (!(other instanceof Customer that)) {
            return false;
        }
        return phone.equals(that.phone);
    }

    @Override
    public int hashCode() {  // objek yang sama WAJIB memiliki kode hash yang sama
        return Objects.hash(phone);
    }
}
```

```
PlainCustomer text starts with 'PlainCustomer@'? true
p1.equals(p2) = false
Customer[Siti Aminah, 081234567890]
c1.equals(c2) = true
c1 == c2      = false
distinct plain customers: 2
distinct customers:       2
```

`PlainCustomer` tidak meng-*override* apa pun. Dua objek yang dibangun dari nama dan nomor telepon yang sama tidak `equals`, sehingga `HashSet` menyimpan keduanya: binatu akan menghitung Siti dua kali. (Kita hanya mencetak apakah teksnya *diawali* `PlainCustomer@`, karena bilangan setelah `@` berbeda pada setiap kali program dijalankan.) `Customer` meng-*override* ketiganya. `equals` miliknya mengikuti pola yang patut dihafal: `true` seketika bila objek lain itu adalah objek ini sendiri; `false` bila objek lain itu sama sekali bukan `Customer` (pola `instanceof` sekaligus memberi kita `that`, bertipe `Customer`); selain itu, bandingkan field yang menentukan identitas, di sini nomor telepon. `hashCode` miliknya meng-*hash* field yang persis sama. Kini `c1` dan `c2` sama tetapi bukan objek yang sama (`==` bernilai `false`), dan himpunan berisi tiga pelanggan itu hanya beranggota dua.

> **PERINGATAN:** Bila Anda meng-*override* `equals`, Anda **wajib** meng-*override* `hashCode` sehingga objek-objek yang sama memiliki kode hash yang sama. Java tidak memeriksa hal ini. Kelas yang melanggar aturan itu lolos kompilasi dan berjalan, lalu `HashSet` diam-diam menyimpan "duplikat" dan `HashMap` diam-diam gagal menemukan kunci yang sebenarnya ada [6].

Python membuat aturan itu lebih sulit dilanggar, seperti diperlihatkan **Kode 7.12**.

Kode 7.12 `object_root.py`: ketiga metode yang sama dalam Python, dan sebuah kelas yang mendefinisikan `__eq__` tanpa `__hash__`.

```
"""object_root.py: setiap kelas mewarisi object; override __str__, __eq__, __hash__."""


class PlainCustomer:  # mewarisi __repr__, __eq__ dan __hash__ dari object
    def __init__(self, name, phone):
        self.name = name
        self.phone = phone


class Customer:
    def __init__(self, name, phone):
        self.name = name
        self.phone = phone  # nomor telepon mengidentifikasi pelanggan

    def __str__(self):
        return f"Customer[{self.name}, {self.phone}]"

    def __eq__(self, other):
        if not isinstance(other, Customer):
            return NotImplemented
        return self.phone == other.phone

    def __hash__(self):  # objek yang sama WAJIB memiliki hash yang sama
        return hash(self.phone)


class HalfDoneCustomer:
    def __init__(self, phone):
        self.phone = phone

    def __eq__(self, other):  # __eq__ tanpa __hash__ ...
        return isinstance(other, HalfDoneCustomer) and self.phone == other.phone


def main():
    p1 = PlainCustomer("Siti Aminah", "081234567890")
    p2 = PlainCustomer("Siti Aminah", "081234567890")
    print("PlainCustomer text starts with '<__main__.PlainCustomer object'?",
          repr(p1).startswith("<__main__.PlainCustomer object"))
    print("p1 == p2 =", p1 == p2)
```

Kode 7.12 (Lanjutan) *object_root.py: ketiga metode yang sama dalam Python, dan sebuah kelas yang mendefinisikan __eq__ tanpa __hash__.*

```
    c1 = Customer("Siti Aminah", "081234567890")
    c2 = Customer("Siti A.", "081234567890")  # nomor telepon sama
    c3 = Customer("Budi Santoso", "085712340000")
    print(c1)
    print("c1 == c2 =", c1 == c2)
    print("c1 is c2 =", c1 is c2)
    print("distinct plain customers:", len({p1, p2}))
    print("distinct customers:      ", len({c1, c2, c3}))
    try:
        {HalfDoneCustomer("0812")}
    except TypeError as e:
        print("TypeError:", e)  # ... membuat kelas tidak dapat di-hash


if __name__ == "__main__":
    main()
```

```
PlainCustomer text starts with '<__main__.PlainCustomer object'? True
p1 == p2 = False
Customer[Siti Aminah, 081234567890]
c1 == c2 = True
c1 is c2 = False
distinct plain customers: 2
distinct customers:       2
TypeError: unhashable type: 'HalfDoneCustomer'
```

Kode Python lebih pendek tetapi mengikuti pola yang sama. `__eq__` mengembalikan nilai khusus `NotImplemented`, bukan `False`, ketika objek lain itu bertipe yang tidak berhubungan; dengan begitu Python secara santun diberi kesempatan menanyakan kepada objek lain itu apakah *ia* tahu cara membandingkannya. Kelas terakhir memperlihatkan jaring pengaman Python: kelas yang mendefinisikan `__eq__` tetapi tidak `__hash__` otomatis mendapat `__hash__` bernilai `None`, sehingga objeknya sama sekali tidak dapat dimasukkan ke dalam himpunan. `TypeError` yang lantang jauh lebih baik daripada perilaku keliru Java yang diam-diam.

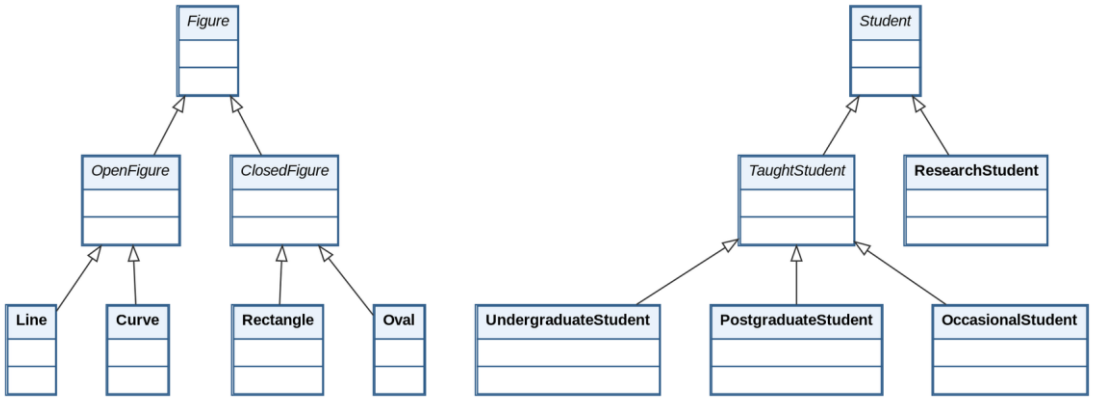
7.9 Kelas Abstrak

Kita sudah memakai kelas abstrak sejak **Kode 7.1**. Sudah waktunya menyatakan gagasan itu dengan benar dan melihat apa manfaatnya.

> **DEFINISI. Metode abstrak** adalah metode yang dideklarasikan tanpa badan. **Kelas abstrak** adalah kelas yang tidak dapat diinstansiasi; di Java ia ditandai `abstract`, dan ia harus abstrak bila mendeklarasikan atau mewarisi metode abstrak yang tidak diimplementasikannya. **Kelas konkret** adalah kelas yang dapat diinstansiasi; ia harus mengimplementasikan setiap metode abstrak yang diwarisinya.

Kelas abstrak berada di antara antarmuka, yang (secara tradisional) hanya berisi deklarasi metode, dan kelas konkret, yang hanya berisi metode yang lengkap. Ia dapat memuat field, konstruktor, metode lengkap, dan metode abstrak sekaligus. Tujuannya adalah menangkap *apa yang dimiliki bersama oleh sebuah keluarga kelas*, termasuk kodenya, sambil menyerahkan *bagian-bagian yang berbeda* kepada subkelas. Tidak ada objek `DB` — `new DB()` tidak lolos kompilasi — tetapi ada objek `LinearDB` dan `TreeDB`, dan variabel bertipe `DB` dapat merujuk keduanya.

Sebuah hierarki dapat memiliki beberapa tingkat abstrak. Tinjaulah bangun geometri. Setiap bangun memiliki panjang (panjang sebuah garis, keliling sebuah bangun tertutup). Sebagian bangun bersifat *terbuka* — garis, kurva — dan tidak melingkupi luas; sebagian lagi *tertutup* — persegi panjang, oval — dan melingkupi luas. Pohon kiri pada **Gambar 7.5** menggambar hierarki itu; pohon kanan adalah hierarki serupa untuk mahasiswa, yang diminta untuk Anda implementasikan dalam Soal Latihan 7.4.



Gambar 7.5 Dua hierarki dengan tingkat dalam yang abstrak (nama bercetak miring). Kiri: bangun geometri, terbagi menjadi bangun terbuka dan tertutup. Kanan: mahasiswa, terbagi menjadi mahasiswa perkuliahan dan mahasiswa riset.

Kode 7.13 mengimplementasikan bangun-bangun itu. Kedua bangun tertutup yang konkret masing-masing memerlukan dua rumus. Persegi panjang dengan lebar w dan tinggi h memiliki keliling $2(w + h)$ dan luas wh. Oval (elips) dengan setengah sumbu a dan b — setengah lebarnya dan setengah tingginya — memiliki luas

$$A = \pi ab \tag{7.2}$$

tetapi kelilingnya tidak punya rumus eksak dalam fungsi elementer. Aproksimasi yang indah, ditemukan matematikawan Srinivasa Ramanujan pada tahun 1914, adalah

$$P \approx \pi\left[3(a + b) - \sqrt{(3a + b)(a + 3b)}\right] \tag{7.3}$$

yang eksak untuk lingkaran ($a = b = r$ menghasilkan $\pi\left[6r - \sqrt{16r^2}\right] = 2\pi r$) dan meleset hanya sepersekian persen untuk oval apa pun yang mungkin Anda gambar.

Kode 7.13 `FigureDemo.java`: `Figure` abstrak dengan subkelas abstrak `OpenFigure` dan `ClosedFigure` serta empat bangun konkret.

```
import java.util.Locale;

public class FigureDemo {
    public static void main(String[] args) {
        Figure[] figures = {
            new Line(0, 0, 3, 4),
            new Curve(new double[] {0, 1, 2}, new double[] {0, 1, 0}),
            new Rectangle(4, 2.5),
            new Oval(3, 2)
        };
        for (Figure f : figures) {
            System.out.println(f.describe());
        }
        double totalArea = 0;
        for (Figure f : figures) {
            if (f instanceof ClosedFigure c) {
                totalArea += c.area();
            }
        }
        System.out.printf(Locale.US, "Total area of the closed figures: %.3f%n",
totalArea);
    }
}

abstract class Figure {
    private final String name;

    protected Figure(String name) {
        this.name = name;
    }

    String getName() {
        return name;
    }

    abstract double length();  // setiap bangun memiliki panjang (atau keliling)

    // Metode konkret yang dibangun di atas metode abstrak.
    String describe() {
        return String.format(Locale.US, "%-9s length %7.3f", name, length());
    }
}
```

Kode 7.13 (Lanjutan) *FigureDemo.java: Figure abstrak dengan subkelas abstrak OpenFigure dan ClosedFigure serta empat bangun konkret.*

```
abstract class OpenFigure extends Figure {
    protected OpenFigure(String name) {
        super(name);
    }
}

abstract class ClosedFigure extends Figure {
    protected ClosedFigure(String name) {
        super(name);
    }

    abstract double area();  // hanya bangun tertutup yang melingkupi luas

    @Override
    String describe() {
        return super.describe() + String.format(Locale.US, "   area %7.3f", area());
    }
}

class Line extends OpenFigure {
    private final double x1, y1, x2, y2;

    Line(double x1, double y1, double x2, double y2) {
        super("Line");
        this.x1 = x1;
        this.y1 = y1;
        this.x2 = x2;
        this.y2 = y2;
    }

    @Override
    double length() {
        return Math.hypot(x2 - x1, y2 - y1);
    }
}

class Curve extends OpenFigure {
    private final double[] xs, ys;  // kurva digambar sebagai garis patah melalui titik-
titik

    Curve(double[] xs, double[] ys) {
        super("Curve");
        this.xs = xs;
        this.ys = ys;
    }
```

Kode 7.13 (Lanjutan) *FigureDemo.java: Figure abstrak dengan subkelas abstrak OpenFigure dan ClosedFigure serta empat bangun konkret.*

```
    @Override
    double length() {
        double sum = 0;
        for (int i = 1; i < xs.length; i++) {
            sum += Math.hypot(xs[i] - xs[i - 1], ys[i] - ys[i - 1]);
        }
        return sum;
    }
}

class Rectangle extends ClosedFigure {
    private final double width, height;

    Rectangle(double width, double height) {
        super("Rectangle");
        this.width = width;
        this.height = height;
    }

    @Override
    double length() {
        return 2 * (width + height);
    }

    @Override
    double area() {
        return width * height;
    }
}

class Oval extends ClosedFigure {
    private final double a, b;  // kedua setengah sumbu

    Oval(double a, double b) {
        super("Oval");
        this.a = a;
        this.b = b;
    }

    @Override
    double length() {  // aproksimasi keliling dari Ramanujan
        return Math.PI * (3 * (a + b) - Math.sqrt((3 * a + b) * (a + 3 * b)));
    }
```

Kode 7.13 (Lanjutan) *FigureDemo.java: Figure abstrak dengan subkelas abstrak OpenFigure dan ClosedFigure serta empat bangun konkret.*

```
    @Override
    double area() {
        return Math.PI * a * b;
    }
}
```

```
Line      length   5.000
Curve     length   2.828
Rectangle length  13.000   area  10.000
Oval      length  15.865   area  18.850
Total area of the closed figures: 28.850
```

Rancangan ini layak dicermati, karena ia memperlihatkan tiga hal yang dapat dilakukan kelas abstrak.

1. **Menyatakan apa yang wajib disediakan setiap subkelas.** `Figure` mendeklarasikan `abstract double length()`. Setiap bangun konkret wajib menyediakannya, dan kompilator memeriksa: kelas konkret yang melupakan metode abstrak warisan tidak lolos kompilasi.
2. **Menyediakan kode yang bergantung pada apa yang disediakan subkelas.** `Figure.describe()` lengkap. Ia memanggil `length()` tanpa mengetahui bangun mana yang sedang diukurnya, dan *dynamic dispatch* memasok rumus yang tepat. Metode seperti ini — kerangka tetap dengan satu langkah yang diisi subkelas — disebut **metode templat** (*template method*), salah satu pola desain klasik [7].
3. **Menambahkan persyaratan bagi sebagian keluarga.** `ClosedFigure` juga abstrak. Ia menambahkan `abstract double area()`, yang hanya dimiliki bangun tertutup, dan meng-*override* `describe()` untuk menambahkan luasnya, dengan memanggil `super.describe()` untuk bagian yang sudah dapat ditulis oleh `Figure`.

Periksalah angka-angkanya. Garis dari (0,0) ke (3,4) panjangnya $\sqrt{3^2 + 4^2} = 5$; `Math.hypot` menghitung persis ini. Kurva yang melalui (0,0), (1,1), (2,0) terdiri atas dua ruas sepanjang $\sqrt{2}$, totalnya $2\sqrt{2} \approx 2.828$. Persegi panjang berkeliling $2(4 + 2.5) = 13$ dan berluas 10. Untuk oval, Persamaan (7.2) memberikan $\pi \cdot 3 \cdot 2 \approx 18.850$ dan Persamaan (7.3) memberikan $\pi\left[15 - \sqrt{11 \cdot 9}\right] = \pi(15 - 9.950) \approx 15.865$. Baris terakhir menjumlahkan luas bangun tertutup saja, dengan `instanceof ClosedFigure` karena `area()` tidak ada untuk bangun terbuka: $10 + 18.850 = 28.850$.

Satu rincian layak disebut: `String.format` dan `printf` mengikuti pengaturan regional komputer, sehingga pada komputer yang diatur ke bahasa Indonesia `%.3f` mencetak `5,000` dengan koma desimal. Memberikan `Locale.US` membuat keluarannya sama di mana-mana; aplikasi untuk pelanggan Indonesia akan memberikan `Locale.forLanguageTag("id-ID")`. (Karena alasan yang sama, angka dalam rumus, keluaran, dan nota bab ini ditulis dengan titik desimal dan koma pemisah ribuan, persis seperti yang dicetak program.)

Apa kata kompilator bila kita mencoba membuat bangun polos, atau melupakan sebuah metode? Menambahkan `Figure plain = new Figure("?");` di awal `main` menghasilkan

```
FigureDemo.java:5: error: Figure is abstract; cannot be instantiated
        Figure plain = new Figure("?");
                       ^
1 error
```

dan subkelas konkret yang melupakan sebuah metode abstrak ditolak dengan pesan yang menyebut metode yang hilang itu; kita akan melihat satu contohnya di Subbab 7.15.

Kode 7.14 adalah versi Python-nya.

Kode 7.14 `figure_demo.py`: hierarki bangun dalam Python; `OpenFigure` dan `ClosedFigure` mencantumkan `ABC` sekali lagi untuk menegaskan bahwa keduanya pun abstrak.

```python
"""figure_demo.py: hierarki Figure abstrak dengan bangun terbuka dan tertutup."""
import math
from abc import ABC, abstractmethod


class Figure(ABC):
    def __init__(self, name):
        self._name = name

    @abstractmethod
    def length(self):  # setiap bangun memiliki panjang (atau keliling)
        ...

    # Metode konkret yang dibangun di atas metode abstrak.
    def describe(self):
        return f"{self._name:<9} length {self.length():7.3f}"


class OpenFigure(Figure, ABC):
    pass


class ClosedFigure(Figure, ABC):
    @abstractmethod
    def area(self):  # hanya bangun tertutup yang melingkupi luas
        ...

    def describe(self):
        return super().describe() + f"   area {self.area():7.3f}"


class Line(OpenFigure):
    def __init__(self, x1, y1, x2, y2):
        super().__init__("Line")
        self._p, self._q = (x1, y1), (x2, y2)

    def length(self):
        return math.dist(self._p, self._q)


class Curve(OpenFigure):
    def __init__(self, xs, ys):  # kurva digambar sebagai garis patah melalui titik-titik
        super().__init__("Curve")
        self._points = list(zip(xs, ys))
```

Kode 7.14 (Lanjutan) *figure_demo.py: hierarki bangun dalam Python; OpenFigure dan ClosedFigure mencantumkan ABC sekali lagi untuk menegaskan bahwa keduanya pun abstrak.*

```
    def length(self):
        return sum(math.dist(a, b) for a, b in zip(self._points, self._points[1:]))


class Rectangle(ClosedFigure):
    def __init__(self, width, height):
        super().__init__("Rectangle")
        self._w, self._h = width, height

    def length(self):
        return 2 * (self._w + self._h)

    def area(self):
        return self._w * self._h


class Oval(ClosedFigure):
    def __init__(self, a, b):  # kedua setengah sumbu
        super().__init__("Oval")
        self._a, self._b = a, b

    def length(self):  # aproksimasi keliling dari Ramanujan
        a, b = self._a, self._b
        return math.pi * (3 * (a + b) - math.sqrt((3 * a + b) * (a + 3 * b)))

    def area(self):
        return math.pi * self._a * self._b


def main():
    figures = [Line(0, 0, 3, 4), Curve([0, 1, 2], [0, 1, 0]), Rectangle(4, 2.5), Oval(3,
2)]
    for f in figures:
        print(f.describe())
    total_area = sum(f.area() for f in figures if isinstance(f, ClosedFigure))
    print(f"Total area of the closed figures: {total_area:.3f}")
    try:
        Figure("?")
    except TypeError as e:
        print("Figure():", e)

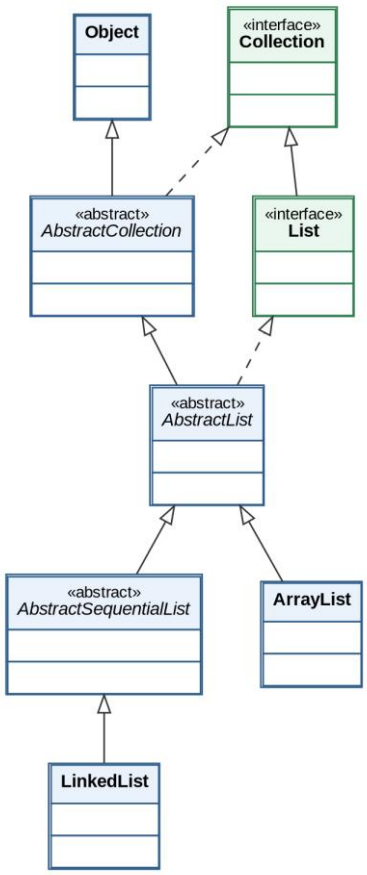

if __name__ == "__main__":
    main()
```

```
Line      length   5.000
Curve     length   2.828
Rectangle length  13.000   area  10.000
Oval      length  15.865   area  18.850
Total area of the closed figures: 28.850
Figure(): Can't instantiate abstract class Figure with abstract method length
```

`math.dist` menghitung jarak antara dua titik, memainkan peran `Math.hypot`. `Curve.length` memasangkan setiap titik dengan titik berikutnya melalui `zip(self._points, self._points[1:])`, lalu menjumlahkan jarak-jaraknya. `sum(...)` terakhir dengan `if` di dalamnya adalah *generator expression*, bentuk ringkas Python untuk "ulangi, saring, jumlahkan". Python memeriksa metode abstrak tepat ketika sebuah objek dibuat, dan baris terakhir keluaran memperlihatkan penolakannya.

7.9.1 Pustaka Java dibangun dengan cara yang sama

Kelas abstrak bukan sekadar alat akademis. **Gambar 7.6** memperlihatkan satu sudut pustaka koleksi milik Java sendiri, yang dipakai di Bab 5 dan dipelajari mendalam di Bab 9.



Gambar 7.6 Sebagian hierarki koleksi Java. Garis utuh dengan segitiga kosong adalah `extends`; garis putus-putus adalah `implements`. Kelas-kelas abstrak menyediakan kode bersama, antarmuka-antarmuka menyatakan kontraknya, dan `ArrayList` serta `LinkedList` adalah kelas konkret yang benar-benar dibuat oleh program.

`Collection` dan `List` adalah antarmuka: keduanya menjanjikan metode seperti `add`, `get`, dan `size`. `AbstractCollection` dan `AbstractList` adalah kelas abstrak yang mengimplementasikan sebagian besar metode itu berdasarkan beberapa metode yang dibiarkannya abstrak — persis seperti `DB` mengimplementasikan `findOneOf` berdasarkan `search`. `AbstractList`, misalnya, mengimplementasikan `contains`, `indexOf`, `iterator`, dan banyak lagi hanya berdasarkan `get(int)` dan `size()`. `ArrayList` memperluas `AbstractList` dan menyediakan sebuah larik; `LinkedList` memperluas `AbstractSequentialList` dan menyediakan rantai simpul. Pemrogram yang menginginkan jenis list baru cukup menulis dua metode dan mewarisi puluhan metode lainnya.

7.10 Mewarisi Kelas Konkret, dan Mengapa Harus Berhati-hati

Tidak ada yang memaksa sebuah superkelas bersifat abstrak. Sebuah kelas boleh memperluas kelas biasa yang *konkret*, dan mekanismenya persis sama: subkelas mendapat setiap field dan metode dan boleh meng-*override* yang mana pun. Satu-satunya perbedaan adalah bahwa superkelas itu dapat memiliki objeknya sendiri. Jadi kita dapat saja melewatkan `DB` dan

membiarkan `TreeDB` mewarisi `LinearDB`, seperti yang dilakukan **Kode 7.15**. Listing itu berada di paket `dbconcrete` agar nama-nama kelasnya tidak bertabrakan dengan milik paket `db`.

Kode 7.15 `dbconcrete/DBTest.java`: `TreeDB` mewarisi kelas konkret `LinearDB`.

```
package dbconcrete;

public class DBTest {
    public static void main(String[] args) {
        LinearDB ldb = new LinearDB();
        TreeDB tdb = new TreeDB();
        ldb.addKey(1);
        if (ldb.search(1)) {
            System.out.println("It's found!");
        } else {
            System.out.println("It's not found!");
        }
        tdb.addKey(2);
        if (tdb.search(1)) {
            System.out.println("It's found!");
        } else {
            System.out.println("It's not found!");
        }
        // Apa yang diterima TreeDB dari LinearDB tanpa memintanya:
        System.out.println("-- what TreeDB inherited --");
        System.out.println("tdb.table.length = " + tdb.table.length);
        System.out.println("tdb is a LinearDB? " + (tdb instanceof LinearDB));
        LinearDB surprise = new TreeDB();  // sah, tetapi apakah pohon itu "DB linear"?
        surprise.addSeveral(new int[] {7, 8});
    }
}

class LinearDB {
    int table[] = new int[10];
    int size = 0;

    void addKey(int key) {
        System.out.println("LinearDB: addKey()");
    }

    boolean search(int key) {
        boolean result = true;
        System.out.println("LinearDB: searchKey()");
        return result;
    }
```

Kode 7.15 (Lanjutan) *dbconcrete/DBTest.java: TreeDB mewarisi kelas konkret LinearDB.*

```
    void addSeveral(int keys[]) {
        for (int i = 0; i < keys.length; i++) {
            addKey(keys[i]);
        }
    }

    boolean findOneOf(int keys[]) {
        for (int i = 0; i < keys.length; i++) {
            if (search(keys[i])) {
                return true;
            }
        }
        return false;
    }
}

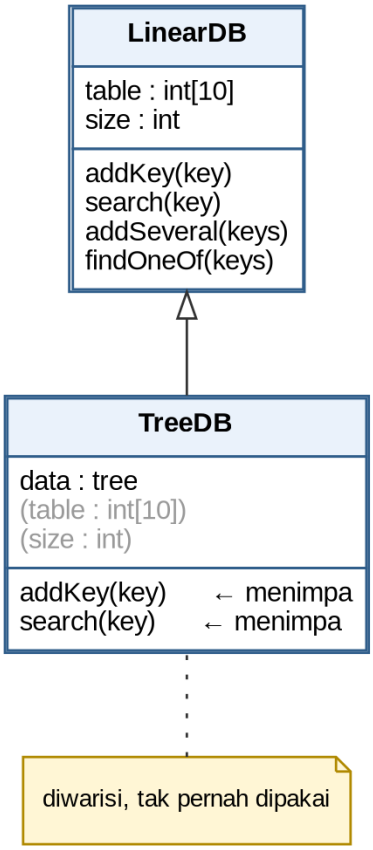
class TreeDB extends LinearDB {
    Object data;  // pohon pencarian biner akan disimpan di sini

    void addKey(int key) {
        System.out.println("TreeDB: addKey()");
    }

    boolean search(int key) {
        boolean result = false;
        System.out.println("TreeDB: searchKey()");
        return result;
    }
}
```

```
LinearDB: addKey()
LinearDB: searchKey()
It's found!
TreeDB: addKey()
TreeDB: searchKey()
It's not found!
-- what TreeDB inherited --
tdb.table.length = 10
tdb is a LinearDB? true
TreeDB: addKey()
TreeDB: addKey()
```

Program itu bekerja; enam baris pertama sama seperti sebelumnya. `addSeveral` dan `findOneOf` ditulis sekali, di `LinearDB`, dan `TreeDB` mewarisinya. Lalu mengapa harus berhati-hati? Blok keluaran terakhir menjawabnya, dan **Gambar 7.7** menggambar jawaban itu.



Gambar 7.7 Mewarisi kelas konkret. `TreeDB` menerima larik dan pencacah milik `LinearDB` walaupun pohon tidak pernah memakainya, dan setiap `TreeDB` kini mengaku sebagai `LinearDB`.

- **Beban bawaan.** Setiap objek `TreeDB` membawa larik berisi sepuluh bilangan bulat dan pencacah `size` yang tidak pernah dipakainya: `tdb.table.length = 10`. Di sini pemborosannya kecil. Dalam hierarki sungguhan beban itu bisa berupa koneksi basis data, berkas yang terbuka, atau *cache*.
- **Janji palsu.** `tdb instanceof LinearDB` bernilai `true`, dan `LinearDB surprise = new TreeDB();` lolos kompilasi. Kode apa pun yang ditulis untuk "basis data linear" — yang wajar saja berasumsi, misalnya, bahwa kunci dikembalikan sesuai urutan penyisipan — kini akan menerima sebuah pohon dan diam-diam keliru. Pohon *bukan* basis data linear. Uji is-a dari Subbab 7.2 gagal, dan kompilator tidak dapat menyadarinya.
- **Keterikatan pada rincian.** `TreeDB` kini bergantung pada segala sesuatu yang dilakukan `LinearDB`, termasuk hal-hal yang mungkin diubah `LinearDB` besok. Subbab 7.11 memperlihatkan bagaimana perubahan semacam itu dapat merusak subkelas yang tidak berbuat salah apa pun.

Versi dengan `DB` abstrak tidak punya satu pun masalah ini: `DB` hanya memuat apa yang dimiliki bersama oleh *setiap* basis data, sedangkan `LinearDB` dan `TreeDB` bersaudara, tidak ada yang berpura-pura menjadi yang lain. Itulah pelajaran umumnya: **bila dua kelas berbagi kode, ekstraklah superkelas abstrak yang diperluas keduanya, alih-alih membuat yang satu memperluas yang lain.**

Kode 7.16 mengulang percobaan itu dalam Python. Perhatikan komentar di `TreeDB.__init__`: karena Python tidak pernah memanggil inisialisator superkelas dengan sendirinya, `TreeDB` versi Python yang melupakan `super().__init__()` *tidak* akan punya larik itu — beban bawaan hanya

muncul bila kita memintanya, dan `add_several` warisan tetap berjalan, karena ia tidak pernah menyentuh larik itu. Namun janji palsunya tetap ada, bagaimanapun juga.

Kode 7.16 `db_concrete.py`: versi superkelas konkret dalam Python.

```
"""db_concrete.py: TreeDB mewarisi kelas konkret LinearDB."""


class LinearDB:
    def __init__(self):
        self.table = [0] * 10
        self.size = 0

    def add_key(self, key):
        print("LinearDB: addKey()")

    def search(self, key):
        result = True
        print("LinearDB: searchKey()")
        return result

    def add_several(self, keys):
        for key in keys:
            self.add_key(key)

    def find_one_of(self, keys):
        for key in keys:
            if self.search(key):
                return True
        return False


class TreeDB(LinearDB):
    def __init__(self):
        super().__init__()  # wajib dipanggil, jika tidak table dan size tidak pernah ada
        self.data = None    # pohon pencarian biner akan disimpan di sini

    def add_key(self, key):
        print("TreeDB: addKey()")

    def search(self, key):
        result = False
        print("TreeDB: searchKey()")
        return result
```

Kode 7.16 (Lanjutan) *db_concrete.py: versi superkelas konkret dalam Python.*

```
def main():
    ldb = LinearDB()
    tdb = TreeDB()
    ldb.add_key(1)
    if ldb.search(1):
        print("It's found!")
    else:
        print("It's not found!")
    tdb.add_key(2)
    if tdb.search(1):
        print("It's found!")
    else:
        print("It's not found!")
    # Apa yang diterima TreeDB dari LinearDB tanpa memintanya:
    print("-- what TreeDB inherited --")
    print("len(tdb.table) =", len(tdb.table))
    print("tdb is a LinearDB?", isinstance(tdb, LinearDB))
    surprise: LinearDB = TreeDB()  # sah, tetapi apakah pohon itu "DB linear"?
    surprise.add_several([7, 8])


if __name__ == "__main__":
    main()
```

```
LinearDB: addKey()
LinearDB: searchKey()
It's found!
TreeDB: addKey()
TreeDB: searchKey()
It's not found!
-- what TreeDB inherited --
len(tdb.table) = 10
tdb is a LinearDB? True
TreeDB: addKey()
TreeDB: addKey()
```

7.11 Komposisi versus Pewarisan

Pewarisan bukan satu-satunya cara memakai ulang kode. Cara lainnya adalah *memakai* objek dari kelas lain alih-alih *menjadi* objek kelas itu: sebuah kelas menyimpan referensi ke objek pembantu di dalam sebuah field dan memanggil metode-metodenya. Cara ini disebut **komposisi** (*composition*; atau *agregasi* bila objek pembantu itu dapat berdiri sendiri), dan ia menyatakan hubungan **has-a** (*memiliki*): sebuah `Order` memiliki layanan-layanan; sebuah mobil memiliki mesin.

> **DEFINISI. Pewarisan** memakai ulang kode melalui hubungan *is-a*: subkelas *adalah* sejenis superkelas dan menerima anggota-anggotanya. **Komposisi** memakai ulang kode melalui hubungan *has-a*: kelas itu *memuat* objek kelas lain dan mendelegasikan pekerjaan kepadanya dengan memanggil metode-metode publiknya.

Nasihat hampir setiap perancang berpengalaman, yang pertama kali dituliskan dalam buku yang memberi nama pada pola-pola desain klasik, adalah: **utamakan komposisi objek daripada pewarisan kelas** [7]. Pewarisan tidak buruk, tetapi ia menciptakan keterikatan paling erat yang mungkin ada di antara dua kelas: subkelas bergantung bukan hanya pada apa yang *dijanjikan* superkelasnya, melainkan juga pada *bagaimana superkelas itu diimplementasikan*. Alan Snyder, yang menganalisis generasi pertama bahasa berorientasi objek, mengamati bahwa pewarisan memungkinkan subkelas bergantung pada pilihan-pilihan internal superkelas, sehingga melemahkan enkapsulasi yang semestinya diberikan objek [8].

7.11.1 Kelas dasar yang rapuh

Inilah peragaan klasiknya, diadaptasi dari Bloch [6]. Binatu menginginkan himpunan tag pakaian (`T-101`, `T-102`, ...) yang juga menghitung berapa banyak tag yang pernah ditambahkan. Pendekatan yang tampak jelas adalah memperluas `HashSet<String>` dan meng-*override* kedua metode penambah untuk menaikkan sebuah pencacah. **Kode 7.17** menuliskannya, dan di sebelahnya kelas yang sama yang dibangun dengan komposisi.

Kode 7.17 `FragileBase.java`: himpunan tag yang menghitung, dibangun dengan pewarisan, yang menghitung setiap tag dua kali, dan yang dibangun dengan komposisi, yang menghitung dengan benar.

```
import java.util.Collection;
import java.util.HashSet;
import java.util.List;
import java.util.Set;

public class FragileBase {
    public static void main(String[] args) {
        List<String> tags = List.of("T-101", "T-102", "T-103");

        InheritedTagSet inherited = new InheritedTagSet();
        inherited.addAll(tags);
        System.out.println("inherited: size = " + inherited.size()
                + ", tags added = " + inherited.getAddCount());

        ComposedTagSet composed = new ComposedTagSet();
        composed.addAll(tags);
        System.out.println("composed:  size = " + composed.size()
                + ", tags added = " + composed.getAddCount());
    }
}

// Pewarisan: IS-A HashSet, dan bergantung pada cara kerja internal HashSet.
class InheritedTagSet extends HashSet<String> {
    private int addCount = 0;

    @Override
    public boolean add(String tag) {
        addCount++;
        return super.add(tag);
    }

    @Override
    public boolean addAll(Collection<? extends String> tags) {
        addCount += tags.size();
        return super.addAll(tags);  // HashSet.addAll memanggil add() untuk setiap tag!
    }

    int getAddCount() {
        return addCount;
    }
}
```

Kode 7.17 (Lanjutan) *FragileBase.java: himpunan tag yang menghitung, dibangun dengan pewarisan, yang menghitung setiap tag dua kali, dan yang dibangun dengan komposisi, yang menghitung dengan benar.*

```
// Komposisi: HAS-A himpunan, dan hanya memakai kontrak publiknya.
class ComposedTagSet {
    private final Set<String> tags = new HashSet<>();
    private int addCount = 0;

    boolean add(String tag) {
        addCount++;
        return tags.add(tag);
    }

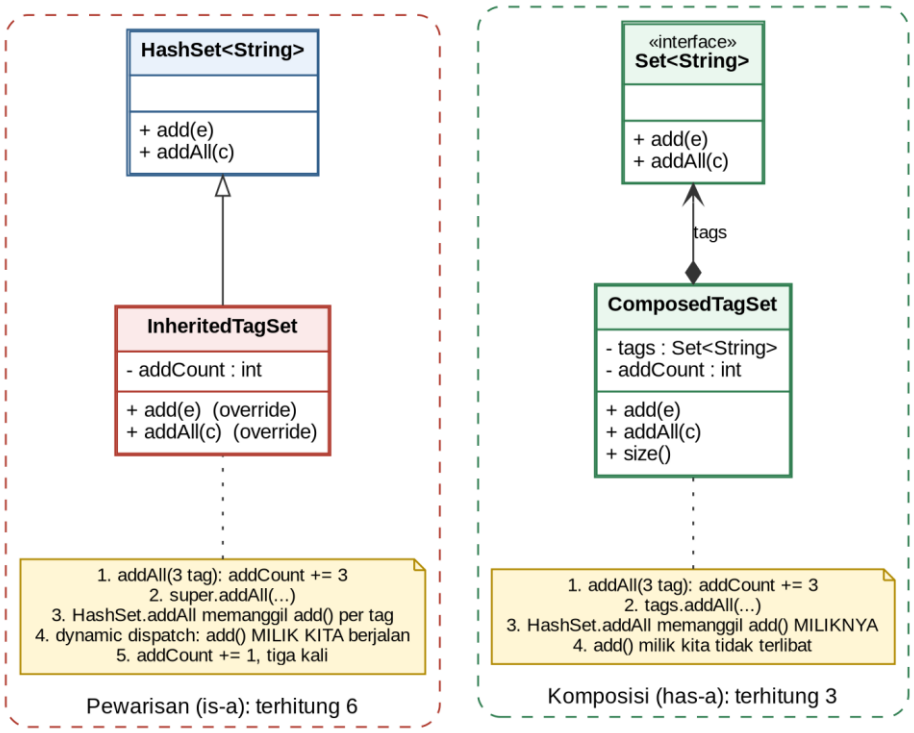
    boolean addAll(Collection<? extends String> more) {
        addCount += more.size();
        return tags.addAll(more);  // add() milik himpunan di dalam bukan milik kita: tidak
terhitung ganda
    }

    int size() {
        return tags.size();
    }

    int getAddCount() {
        return addCount;
    }
}
```

```
inherited: size = 3, tags added = 6
composed:  size = 3, tags added = 3
```

Kita menambahkan tiga tag, dan versi pewarisan berkata enam. Tidak ada yang tampak salah di `InheritedTagSet`; setiap metode menambahkan jumlah yang benar. Masalahnya tersembunyi di dalam `HashSet`: `addAll` miliknya, yang diwarisi dari `AbstractCollection`, diimplementasikan dengan memanggil `add` sekali untuk setiap elemen. Ikutilah **Gambar 7.8**, sisi kiri. `addAll` milik kita menambahkan 3 ke pencacah, lalu memanggil `super.addAll`, yang memanggil `add` tiga kali — dan karena *dynamic dispatch*, panggilan-panggilan itu menuju `add` *milik kita* yang meng-*override*, yang menambahkan 1 setiap kali. Totalnya enam.



Gambar 7.8 Kelas dasar yang rapuh. Kiri: `addAll` milik subkelas memanggil `addAll` milik superkelas, yang memanggil `add`, yang oleh *dynamic dispatch* dikirim kembali ke subkelas: setiap tag terhitung dua kali. Kanan: kelas komposisi memanggil metode objek yang sekadar dipegangnya; `add` miliknya sendiri tidak pernah dimasuki ulang.

Bagian yang menakutkan adalah bahwa apakah `addAll` memanggil `add` merupakan *rincian implementasi* `HashSet`. Hal itu tidak tercantum dalam dokumentasi `Set`; versi Java mendatang dapat mengubahnya, dan subkelas yang kemarin benar akan rusak tanpa satu baris pun darinya berubah. Dari situlah namanya: kelas dasarnya *rapuh* (*fragile base class*), karena subkelas-subkelasnya bergantung pada isi dalamnya.

`ComposedTagSet`, di sisi kanan **Gambar 7.8**, menyimpan `Set<String>` di dalam field private dan meneruskan panggilan kepadanya. `addAll` miliknya memanggil `tags.addAll(more)`, yang memanggil `add` milik himpunan di dalam itu sendiri; kelas kita bukan `HashSet`, sehingga *dispatch* tidak dapat memantul kembali ke dalamnya, dan hitungannya 3. Kelas komposisi juga hanya menampakkan metode yang dipilihnya, sedangkan kelas pewarisan menampakkan setiap metode `HashSet`, termasuk `remove` dan `clear`.

Python memiliki jebakan yang persis sama, dan **Kode 7.18** memicunya dengan `collections.UserDict`, kelas kamus yang memang dirancang untuk diturunkan.

Kode 7.18 `fragile_base.py`: kelas dasar yang rapuh dalam Python; `UserDict.update` memanggil `__setitem__` untuk setiap pasangan.

```
"""fragile_base.py: menghitung tag dengan mewarisi UserDict, dan dengan komposisi."""
from collections import UserDict


# Pewarisan: IS-A UserDict, dan bergantung pada cara kerja internal UserDict.
class InheritedTagRegistry(UserDict):
    def __init__(self):
        super().__init__()
        self.add_count = 0

    def __setitem__(self, tag, garment):
        self.add_count += 1
        super().__setitem__(tag, garment)

    def update(self, pairs):
        self.add_count += len(pairs)
        super().update(pairs)  # UserDict.update memanggil __setitem__ untuk setiap
pasangan!


# Komposisi: HAS-A dict, dan hanya memakai kontrak publiknya.
class ComposedTagRegistry:
    def __init__(self):
        self._tags = {}
        self.add_count = 0

    def __setitem__(self, tag, garment):
        self.add_count += 1
        self._tags[tag] = garment

    def update(self, pairs):
        self.add_count += len(pairs)
        self._tags.update(pairs)  # metode dict di dalam bukan milik kita: tidak terhitung
ganda

    def __len__(self):
        return len(self._tags)


def main():
    pairs = {"T-101": "shirt", "T-102": "towel", "T-103": "sock"}

    inherited = InheritedTagRegistry()
    inherited.update(pairs)
    print(f"inherited: size = {len(inherited)}, tags added = {inherited.add_count}")
```

Kode 7.18 (Lanjutan) *fragile_base.py: kelas dasar yang rapuh dalam Python; UserDict.update memanggil __setitem__ untuk setiap pasangan.*

```
    composed = ComposedTagRegistry()
    composed.update(pairs)
    print(f"composed:  size = {len(composed)}, tags added = {composed.add_count}")


if __name__ == "__main__":
    main()
```

```
inherited: size = 3, tags added = 6
composed:  size = 3, tags added = 3
```

(`dict` bawaan kebetulan *tidak* menyalurkan `update` melalui `__setitem__` — satu lagi rincian tak terdokumentasi yang harus Anda ketahui, yang justru membuktikan pokok persoalannya.)

7.11.2 Kapan pewarisan tepat? Prinsip substitusi

Komposisi adalah pilihan bawaan yang aman, tetapi pewarisan tepat sekali bila subkelas benar-benar *merupakan* kasus khusus dari superkelasnya — bila setiap potong kode yang ditulis untuk superkelas akan tetap benar seandainya diberi objek subkelas. Barbara Liskov dan Jeannette Wing mengubah kalimat itu menjadi aturan yang tepat, yang kini disebut **prinsip substitusi Liskov** (*Liskov substitution principle*) [9]:

$$\forall x\colon T.\ \phi(x) \ \Rightarrow\ \forall y\colon S.\ \phi(y) \qquad \text{bilamana } S \text{ adalah subtipe dari } T \tag{7.4}$$

Bacalah Persamaan (7.4) sebagai: "bila suatu sifat ϕ dapat dibuktikan untuk setiap objek x bertipe T, maka sifat itu juga harus berlaku untuk setiap objek y bertipe subtipe S". Sifat-sifat yang penting adalah janji-janji yang diandalkan para pemanggil. Bila para pemanggil `Service.price()` mengandalkan bahwa harga adalah bilangan rupiah yang positif, maka tidak ada subkelas yang boleh mengembalikan nol atau bilangan negatif. Bila para pemanggil "basis data linear" mengandalkan urutan penyisipan, maka `TreeDB` tidak boleh menyamar sebagai basis data linear — dan tepat itulah yang salah di Subbab 7.10.

Dalam praktik, tiga pertanyaan menuntaskan sebagian besar kasus:

1. **Apakah kalimat is-a terdengar wajar bagi orang yang bukan pemrogram?** "Layanan kilat adalah sejenis layanan": ya. "Himpunan tag adalah sejenis *hash set*": hanya bagi pemrogram.
2. **Apakah setiap pemanggil superkelas akan puas dengan objek subkelas?** Bila subkelas harus menolak atau mengabaikan suatu metode warisan, jawabannya tidak.
3. **Apakah Anda ingin subkelas berubah ketika isi dalam superkelas berubah?** Bila tidak, pakailah komposisi.

Mesin harga binatu di Subbab 7.15 memakai kedua alat itu, masing-masing di tempatnya. `WashService` *adalah sejenis* `Service`, sehingga ia memperluas `Service`. Pesanan kilat juga *adalah sejenis* `Service` — sebuah pesanan dapat memuatnya — tetapi ia *memiliki* sebuah layanan di dalamnya, yaitu layanan yang dipercepatnya. Maka `ExpressService` memperluas `Service` **dan** memegang sebuah `Service` di dalam field. Paduan itu adalah pola desain yang dikenal, *decorator* [7], dan memungkinkan layanan kilat membungkus layanan apa saja, termasuk yang belum diciptakan.

7.12 Pewarisan Jamak: MRO Python dan Antarmuka Java

Bolehkah sebuah kelas memiliki dua superkelas? Layanan "cuci dan setrika", bagaimanapun, adalah layanan cuci sekaligus layanan setrika. Setiap bahasa menjawab pertanyaan ini secara berbeda, dan perbedaan itu mencerahkan.

7.12.1 Java: satu superkelas, banyak antarmuka

Java mengizinkan sebuah kelas memperluas (`extends`) tepat **satu** kelas. Namun kelas itu boleh mengimplementasikan (`implements`) sebanyak apa pun **antarmuka**. Bab 6 memperkenalkan antarmuka sebagai daftar deklarasi metode; **Tabel 7.3** membandingkannya dengan kelas abstrak.

Tabel 7.3 Kelas abstrak dan antarmuka di Java.

	Kelas abstrak	Antarmuka
Dapat mendeklarasikan metode abstrak	Ya	Ya (semua metode tanpa badan bersifat abstrak)
Dapat memuat kode metode	Ya	Hanya metode `default`, `static`, dan `private` (sejak Java 8 dan 9)
Dapat memegang field instans (keadaan)	Ya	Tidak — hanya konstanta
Memiliki konstruktor	Ya	Tidak
Berapa banyak yang dapat dipakai sebuah kelas	Satu (`extends`)	Banyak (`implements A, B, C`)
Menyatakan	"adalah sejenis", dengan kode dan keadaan bersama	"mampu melakukan", sebuah kemampuan atau kontrak

Pembatasan pada satu superkelas itu disengaja: bahaya sesungguhnya pewarisan jamak terletak pada pewarisan *keadaan* dan *kode* melalui dua jalur. Antarmuka tidak membawa keadaan, sehingga mengimplementasikan beberapa antarmuka aman. Namun sejak Java 8 sebuah antarmuka boleh membawa metode `default` yang berkode, sehingga pertanyaan itu kembali dalam bentuk yang lebih kecil: bagaimana bila dua antarmuka menyediakan metode default dengan signature yang sama? **Kode 7.19** mencobanya.

Kode 7.19 `DiamondDemo.java`: dua antarmuka dengan metode default yang bertentangan; kelas pengimplementasinya wajib menyelesaikan pertentangan itu, dan metode yang diwarisi dari kelas mengalahkan default mana pun.

```
public class DiamondDemo {
    public static void main(String[] args) {
        PremiumCare premium = new PremiumCare();
        System.out.println("PremiumCare: " + premium.finish());

        Foldable asFoldable = premium;    // satu objek, dilihat melalui dua antarmuka
        Steamable asSteamable = premium;
        System.out.println("as Foldable:  " + asFoldable.finish());
        System.out.println("as Steamable: " + asSteamable.finish());

        HotelCare hotel = new HotelCare();
        System.out.println("HotelCare:    " + hotel.finish());  // kelas yang menang
    }
}

interface Foldable {
    default String finish() {
        return "folded";
    }
}

interface Steamable {
    default String finish() {
        return "steamed";
    }
}

// Dua default dengan signature sama: kelasnya WAJIB memutuskan.
class PremiumCare implements Foldable, Steamable {
    @Override
    public String finish() {
        return Steamable.super.finish() + ", then " + Foldable.super.finish();
    }
}

class BasicCare {
    public String finish() {
        return "hung on a rack";
    }
}

// Metode yang diwarisi dari kelas mengalahkan default dari antarmuka.
class HotelCare extends BasicCare implements Foldable {
}
```

```
PremiumCare: steamed, then folded
as Foldable:  steamed, then folded
as Steamable: steamed, then folded
HotelCare:    hung on a rack
```

`Foldable` dan `Steamable` masing-masing menawarkan `finish()` default. `PremiumCare` mengimplementasikan keduanya. Java menolak menebak mana yang semestinya didapatnya; bila `PremiumCare` tidak meng-*override* `finish()`, kompilator akan berhenti dengan:

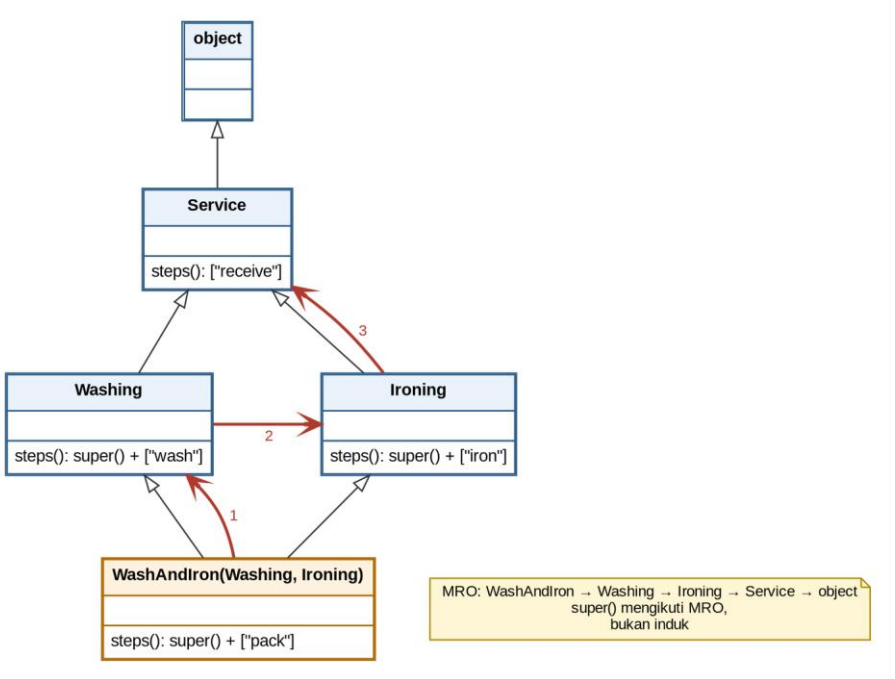
```
DiamondDemo.java:29: error: types Foldable and Steamable are incompatible;
class PremiumCare implements Foldable, Steamable {
^
  class PremiumCare inherits unrelated defaults for finish() from types Foldable and
Steamable
1 error
```

Maka `PremiumCare` meng-*override* `finish()`, dan di dalamnya memakai sintaksis khusus `Steamable.super.finish()` dan `Foldable.super.finish()` untuk memanggil default setiap antarmuka secara eksplisit. Perhatikan bahwa objek itu menjawab sama saja, baik dilihat sebagai

`Foldable` maupun sebagai `Steamable`: *dynamic dispatch* selalu menemukan *override* milik kelasnya. `HotelCare` menggambarkan aturan yang lain: ia mewarisi `finish()` dari *kelas* `BasicCare` dan `finish()` default dari *antarmuka* `Foldable`, dan kelasnya menang tanpa keluhan apa pun. Tiga aturan Java untuk metode default, menurut Spesifikasi Bahasa [3], adalah: (1) metode dari kelas mengalahkan default dari antarmuka; (2) default dari antarmuka yang lebih khusus mengalahkan default dari antarmuka yang diperluasnya; (3) selain itu, kelas harus meng-*override* dan memilih.

7.12.2 Python: pewarisan jamak sejati dan MRO

Python mengizinkan sebuah kelas mencantumkan beberapa superkelas, lengkap dengan keadaan dan kodenya. Masalah yang dihindari Java karenanya harus diselesaikan, dan ia diselesaikan dengan satu urutan yang dirancang dengan cermat. **Gambar 7.9** memperlihatkan *berlian* (*diamond*) klasik: `Washing` dan `Ironing` sama-sama mewarisi `Service`, dan `WashAndIron` mewarisi keduanya.



Gambar 7.9 Berlian. `WashAndIron` memiliki dua jalur naik ke `Service`. Python melinearkan hierarki itu menjadi satu urutan resolusi metode, dan setiap panggilan `super()` berpindah ke kelas berikutnya dalam urutan itu (panah bernomor), sehingga `Service.steps` berjalan tepat satu kali.

Kode 7.20 `diamond_mro.py`: hierarki berlian dengan panggilan `super()` kooperatif, MRO yang dihasilkannya, pertentangan yang diselesaikan oleh urutan, dan hierarki yang tidak memiliki urutan yang konsisten.

```
"""diamond_mro.py: pewarisan jamak, MRO, dan super() kooperatif."""


class Service:
    def steps(self):
        return ["receive"]


class Washing(Service):
    def steps(self):
        return super().steps() + ["wash"]


class Ironing(Service):
    def steps(self):
        return super().steps() + ["iron"]


class WashAndIron(Washing, Ironing):  # berlian: dua jalur naik ke Service
    def steps(self):
        return super().steps() + ["pack"]


class Foldable:
    def finish(self):
        return "folded"


class Steamable:
    def finish(self):
        return "steamed"


class PremiumCare(Steamable, Foldable):  # tidak ada galat: MRO langsung memilih Steamable
    pass


def names(cls):
    return [k.__name__ for k in cls.__mro__]
```

Kode 7.20 (Lanjutan) *diamond_mro.py: hierarki berlian dengan panggilan super() kooperatif, MRO yang dihasilkannya, pertentangan yang diselesaikan oleh urutan, dan hierarki yang tidak memiliki urutan yang konsisten.*

```
def main():
    print("WashAndIron MRO:", names(WashAndIron))
    print("steps():", WashAndIron().steps())
    print("PremiumCare MRO:", names(PremiumCare))
    print("finish():", PremiumCare().finish())
    try:
        class Confused(Service, Washing):  # meminta Service SEBELUM subkelasnya sendiri
            pass
    except TypeError as e:
        print("TypeError:", e)


if __name__ == "__main__":
    main()
```

```
WashAndIron MRO: ['WashAndIron', 'Washing', 'Ironing', 'Service', 'object']
steps(): ['receive', 'iron', 'wash', 'pack']
PremiumCare MRO: ['PremiumCare', 'Steamable', 'Foldable', 'object']
finish(): steamed
TypeError: Cannot create a consistent method resolution
order (MRO) for bases Service, Washing
```

Lihatlah baris kedua keluaran. `WashAndIron().steps()` mengembalikan `['receive', 'iron', 'wash', 'pack']`. Dari mana `iron` datang? `WashAndIron.steps` memanggil `super().steps()`. Kebanyakan pemula mengira `super()` berarti "induk saya", dan akan meramalkan bahwa `Washing.steps` yang berjalan, lalu `super()`-nya mencapai `Service`, sehingga hasilnya `['receive', 'wash', 'pack']` — dengan setrika yang hilang. Bukan itu yang terjadi. **Di Python, `super()` berarti "kelas berikutnya dalam MRO milik kelas objek itu"**, bukan "induk saya". Objeknya adalah `WashAndIron`, yang MRO-nya `WashAndIron, Washing, Ironing, Service, object`. Maka `WashAndIron.steps` memanggil `Washing.steps` (panah 1 di **Gambar 7.9**); `super()` milik `Washing` berpindah ke *kelas berikutnya setelah `Washing` dalam MRO itu*, yaitu `Ironing` (panah 2) — sekalipun `Ironing` bukan superkelas `Washing`; `super()` milik `Ironing` berpindah ke `Service` (panah 3). `Service` mengembalikan `['receive']`, dan hasil-hasilnya diperpanjang dalam perjalanan kembali: `iron`, lalu `wash`, lalu `pack`. Setiap kelas berjalan sekali, dan leluhur bersamanya pun berjalan sekali. Gaya ini, ketika setiap kelas memanggil `super()` dan memercayai MRO, disebut **pewarisan jamak kooperatif** (*cooperative multiple inheritance*) [10].

Bagaimana MRO dihitung? Python memakai **linearisasi C3** (*C3 linearization*), yang dirancang untuk bahasa Dylan [11] dan diadopsi oleh Python 2.3 [12]. Tuliskan $L[C]$ untuk linearisasi (daftar MRO) kelas C, dan misalkan C dideklarasikan dengan basis $B_1, \ldots, B_n$ dalam urutan itu. Maka

$$L[C] = C + \text{merge}(L[B_1], \ldots, L[B_n],\ [B_1, \ldots, B_n]) \tag{7.5}$$

dengan merge berulang kali mengambil elemen pertama (*head*, kepala) dari daftar pertama yang kepalanya tidak muncul di *tail* (ekor: posisi mana pun selain yang pertama) daftar lain mana pun, menambahkannya ke hasil, dan menghapusnya dari semua daftar. Bila kepala semacam itu tidak ada, hierarki itu tidak memiliki urutan yang konsisten.

Mari kita terapkan Persamaan (7.5) dengan tangan pada `WashAndIron(Washing, Ironing)`. Mula-mula linearisasi yang lebih kecil: L[Service] = [Service,object]; L[Washing] = [Washing,Service,object]; L[Ironing] = [Ironing,Service,object]. Sekarang gabungkan ketiga daftar [Washing,Service,object], [Ironing,Service,object], [Washing,Ironing]:

1. Kepala daftar pertama adalah `Washing`. Apakah ia ada di ekor mana pun? Tidak. Ambil. Daftar-daftarnya: [Service,object], [Ironing,Service,object], [Ironing].
2. Kepala daftar pertama adalah `Service`. Ia ada di ekor daftar kedua. Lewati; coba kepala daftar kedua, `Ironing`. Tidak ada di ekor mana pun. Ambil. Daftar-daftarnya: [Service,object], [Service,object], [ ].
3. `Service` tidak lagi ada di ekor mana pun. Ambil; lalu `object`.

Hasilnya: L[WashAndIron] = [WashAndIron,Washing,Ironing,Service,object] — persis seperti yang dicetak program. Langkah 2 adalah jantung C3: `Service` harus menunggu sampai *setiap* subkelasnya sudah terdaftar, dan itulah yang menjamin bahwa leluhur bersama selalu datang sesudah semua keturunannya.

Baris ketiga dan keempat keluaran memperlihatkan bagaimana Python menuntaskan pertentangan yang ditolak Java. `PremiumCare(Steamable, Foldable)` mewarisi dua metode

`finish`; MRO mendaftar `Steamable` lebih dahulu karena ia disebut lebih dahulu, sehingga `Steamable.finish` yang menang. Python memilih secara diam-diam, berdasarkan urutan; Java memaksa Anda memilih secara eksplisit, yang lebih sulit keliru secara tidak sengaja.

Baris terakhir memperlihatkan C3 yang menolak. `class Confused(Service, Washing)` meminta `Service` *sebelum* `Washing`, padahal `Washing` adalah subkelas `Service` dan harus datang sebelumnya. Penggabungan pada langkah 2 tidak menemukan kepala yang dapat diterima, dan Python memunculkan `TypeError` tepat ketika pernyataan kelas itu dijalankan. (Pesannya terpecah menjadi dua baris di Python 3.11; versi yang lebih baru mencetaknya dalam satu baris.)

> **PERINGATAN:** Pewarisan jamak di Python ampuh tetapi mudah disalahgunakan. Pola yang aman adalah satu hierarki utama ditambah kelas-kelas **mixin** kecil — kelas yang menambahkan satu kemampuan, memegang sedikit atau sama sekali tidak memegang keadaan, dan memanggil `super()` secara kooperatif. Bila Anda tergoda mewarisi dua kelas yang "besar", biasanya itu tanda bahwa salah satunya semestinya menjadi field (Subbab 7.11).

7.13 Kontrol Akses

Sejauh ini kita memakai `private`, `protected`, dan `public` tanpa penjelasan yang tuntas. Pewarisan membuat penjelasan itu mendesak: subkelas adalah "orang luar" jenis baru yang berdiri di antara kelas itu sendiri dan seluruh dunia lainnya, dan kita harus memutuskan apa yang boleh dilihatnya.

7.13.1 Empat tingkat akses Java

Access level modifier (pengubah tingkat akses) menentukan apakah kelas lain dapat memakai suatu field atau memanggil suatu metode. Java memilikinya pada dua tingkat:

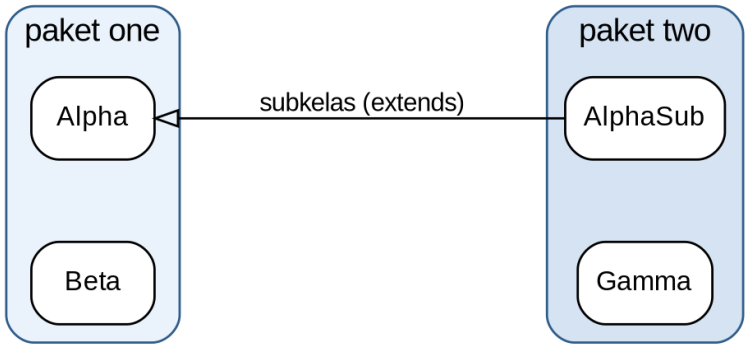
- **Pada tingkat teratas** — untuk kelas itu sendiri — sebuah kelas dapat bersifat `public` (dapat dipakai dari paket mana pun) atau tanpa pengubah, yang disebut **package-private** (hanya dapat dipakai di dalam paketnya sendiri). Semua kelas pembantu dalam listing bab ini, seperti `DB` dan `LinearDB`, bersifat package-private; hanya kelas yang memuat `main` yang publik.
- **Pada tingkat anggota** — untuk field, metode, dan konstruktor — ada empat pilihan: `public`, `protected`, package-private (tanpa pengubah), dan `private`.

Tabel 7.4 adalah tabel rujukannya, dalam bentuk yang diberikan tutorial resmi Java [13]. "Y" berarti kode di tempat itu dapat mengakses anggota yang dideklarasikan dengan pengubah tersebut.

Tabel 7.4 Tingkat akses Java. *Kelas*: kode di kelas yang sama; *Paket*: kode di kelas mana pun dalam paket yang sama; *Subkelas*: kode di subkelas yang dideklarasikan di paket lain; *Dunia*: kode lain mana pun.

Pengubah	Kelas	Paket	Subkelas	Dunia
`public`	Y	Y	Y	Y
`protected`	Y	Y	Y	N
tanpa pengubah	Y	Y	N	N
`private`	Y	N	N	N

Kolom-kolomnya paling mudah dipahami dengan empat kelas konkret yang tersusun dalam dua paket, seperti pada **Gambar 7.10**: `Alpha` dan `Beta` di paket `one`, serta `AlphaSub` (subkelas `Alpha`) dan `Gamma` di paket `two`.



Gambar 7.10 Empat kelas dalam dua paket. `Beta` sepaket dengan `Alpha`; `AlphaSub` berada di paket lain tetapi memperluas `Alpha`; `Gamma` sama sekali tidak berhubungan dengan `Alpha`.

Membaca **Tabel 7.4** dari sudut pandang `Alpha` menghasilkan **Tabel 7.5**: anggota `Alpha` mana saja yang dapat dilihat oleh masing-masing dari keempat kelas itu.

Tabel 7.5 Keterlihatan anggota `Alpha` dari setiap kelas pada **Gambar 7.10**.

Pengubah	Alpha	Beta	AlphaSub	Gamma
`public`	Y	Y	Y	Y
`protected`	Y	Y	Y	N
tanpa pengubah	Y	Y	N	N
`private`	Y	N	N	N

Tabel mudah dihafal dan mudah pula keliru diingat, jadi mari kita minta kompilator membuktikannya. **Kode 7.21**, **Kode 7.22**, **Kode 7.23**, dan **Kode 7.24** adalah keempat kelas itu, masing-masing dalam berkasnya sendiri, karena kelas `public` harus tinggal di berkas yang bernama sama, dan sebuah paket adalah sebuah folder.

Kode 7.21 `one/Alpha.java`: satu field untuk setiap tingkat akses.

```
package one;

public class Alpha {
    public String publicField = "public";
    protected String protectedField = "protected";
    String packageField = "package";  // tanpa modifier: package-private
    private String privateField = "private";

    public String allFour() {  // di dalam Alpha setiap anggota terlihat
        return publicField + " " + protectedField + " " + packageField + " " +
privateField;
    }
}
```

Kode 7.22 `one/Beta.java`: sepaket dengan `Alpha`, bukan subkelas.

```
package one;

public class Beta {  // paket yang sama dengan Alpha, bukan subkelas
    public static String look(Alpha a) {
        return a.publicField + " " + a.protectedField + " " + a.packageField;
        // a.privateField tidak akan lolos kompilasi: private milik Alpha
    }
}
```

Kode 7.23 `two/AlphaSub.java`: paket lain, tetapi subkelas `Alpha`.

```
package two;

import one.Alpha;

public class AlphaSub extends Alpha {  // paket lain, tetapi subkelas
    public String look() {
        return publicField + " " + protectedField;  // anggota warisan milik objek ini
        // packageField tidak akan lolos kompilasi: paket berbeda
    }
}
```

Kode 7.24 `two/Gamma.java`: paket lain, tidak berhubungan; memuat `main`.

```
package two;

import one.Alpha;
import one.Beta;

public class Gamma {  // paket lain, bukan subkelas
    public static void main(String[] args) {
        Alpha a = new Alpha();
        System.out.println("Alpha    sees: " + a.allFour());
        System.out.println("Beta     sees: " + Beta.look(a));
        System.out.println("AlphaSub sees: " + new AlphaSub().look());
        System.out.println("Gamma    sees: " + a.publicField);
    }
}
```

Kompilasi keempatnya dari folder `java` dengan `javac -d out one/*.java two/*.java` dan jalankan dengan `java -cp out two.Gamma`:

```
Alpha    sees: public protected package private
Beta     sees: public protected package
AlphaSub sees: public protected
Gamma    sees: public
```

Setiap baris adalah satu kolom **Tabel 7.5**, dibaca dari atas ke bawah. Untuk melihat isian "N", tambahkan ketiga baris ini ke `Gamma.main` setelah baris terakhirnya:

```
        System.out.println(a.protectedField);
        System.out.println(a.packageField);
        System.out.println(a.privateField);
```

dan kompilator menjawab dengan tiga alasan yang berbeda:

```
two/Gamma.java:13: error: protectedField has protected access in Alpha
        System.out.println(a.protectedField);
                           ^
two/Gamma.java:14: error: packageField is not public in Alpha; cannot be accessed from
outside package
        System.out.println(a.packageField);
                           ^
two/Gamma.java:15: error: privateField has private access in Alpha
        System.out.println(a.privateField);
                           ^
3 errors
```

Kolom `protected` menyimpan satu kehalusan yang mengejutkan bahkan pemrogram berpengalaman. `AlphaSub` boleh memakai `protectedField` **milik objeknya sendiri** (ia menulis `protectedField`, yang berarti `this.protectedField`). Ia *tidak* boleh menjangkau sembarang objek `Alpha` yang bukan `AlphaSub`. Tambahkan metode ini sebagai metode pertama `AlphaSub`:

```
    public String peek(Alpha other) {
        return other.protectedField;
    }
```

dan `javac` berkata:

```
two/AlphaSub.java:7: error: protectedField has protected access in Alpha
        return other.protectedField;
                    ^
1 error
```

Aturannya [3]: di luar paketnya, anggota protected hanya boleh diakses subkelas melalui referensi yang bertipe subkelas itu (atau salah satu subkelasnya sendiri). Gagasannya, `protected` ada agar sebuah kelas dapat berbagi rincian dengan *keturunannya*, bukan agar setiap keturunan dapat mencampuri setiap objek dalam keluarga itu.

7.13.2 Memilih tingkat akses

Bila pemrogram lain memakai kelas Anda, Anda ingin memastikan bahwa kesalahan akibat salah pakai tidak mungkin terjadi. Tingkat akses adalah alatnya. Dua pedoman praktis mencakup hampir segalanya [6], [13]:

- **Pakailah tingkat akses paling ketat yang masuk akal untuk suatu anggota. Pakailah `private` kecuali Anda punya alasan kuat untuk tidak memakainya.** Setiap anggota yang Anda tampakkan adalah janji yang harus Anda tepati selamanya, karena kode seseorang mungkin menjadi bergantung padanya.
- **Hindari field publik, kecuali konstanta** (`public static final`). Field publik mengikat Anda pada satu representasi. Bila `WashingMachine` menampakkan `public double loadKg`, Anda tidak akan pernah dapat menambahkan pemeriksaan bahwa muatan muat di dalam drum, karena siapa saja dapat menulis `m.loadKg = 99`.

`protected` memerlukan kewaspadaan khusus. Anggota protected adalah bagian dari janji kelas itu kepada *setiap subkelas yang akan pernah ditulis siapa pun*, di paket mana pun; ia hampir sepublik `public`. Pakailah untuk hal-hal yang benar-benar diperlukan subkelas, seperti `comparisons` di **Kode 7.3** atau `id` di **Kode 7.9**, dan sedapat mungkin utamakan field `private` dengan metode `protected`.

7.13.3 Python: konvensi, bukan aturan

Python tidak punya pengubah akses. Semuanya, secara teknis, publik. Yang dimiliki Python adalah seperangkat konvensi penamaan, yang diuraikan dalam panduan gayanya [14], dan dibaca dengan cara yang sama oleh setiap pemrogram Python:

- `name` — publik: bagian dari janji kelas itu.
- `_name` (satu garis bawah di depan) — internal: "mohon jangan dipakai dari luar; subkelas dipersilakan". Interpreter tidak menegakkannya; alat bantu dan pembacalah yang menegakkannya. Inilah padanan sehari-hari `protected` atau akses paket di Java, dan inilah yang dipakai listing Python bab ini untuk field.
- `__name` (dua garis bawah di depan, paling banyak satu di belakang) — *name mangling* (pengubahan nama): di dalam kelas `Alpha`, Python diam-diam mengganti nama `__name` menjadi `_Alpha__name`. Tujuannya bukan kerahasiaan melainkan *menghindari tabrakan*: subkelas yang kebetulan memakai nama yang sama mendapat atributnya sendiri yang diubah secara berbeda, dan tidak dapat menimpa milik induknya secara tidak sengaja.

Kode 7.25 memperlihatkan ketiganya.

Kode 7.25 `access_demo.py`: tiga konvensi penamaan Python, dan apa yang sebenarnya dilakukan *name mangling*.

```
"""access_demo.py: konvensi akses Python: public, _protected, __private."""


class Alpha:
    def __init__(self):
        self.public_field = "public"
        self._protected_field = "protected"  # konvensi: internal, subkelas dipersilakan
        self.__private_field = "private"     # namanya diubah (mangled) menjadi
_Alpha__private_field

    def all_three(self):  # di dalam Alpha setiap nama berlaku seperti tertulis
        return f"{self.public_field} {self._protected_field} {self.__private_field}"


class AlphaSub(Alpha):
    def look(self):
        text = f"{self.public_field} {self._protected_field}"
        try:
            text += " " + self.__private_field  # menjadi self._AlphaSub__private_field
        except AttributeError as e:
            text += f"  (AttributeError: {e})"
        return text


def main():
    a = Alpha()
    print("Alpha    sees:", a.all_three())
    print("AlphaSub sees:", AlphaSub().look())
    print("outsider sees:", a.public_field, a._protected_field, "(allowed, but impolite)")
    print("attribute names:", list(vars(a)))
    print("mangled name works:", a._Alpha__private_field, "(never do this)")


if __name__ == "__main__":
    main()
```

```
Alpha    sees: public protected private
AlphaSub sees: public protected  (AttributeError: 'AlphaSub' object has no attribute
'_AlphaSub__private_field')
outsider sees: public protected (allowed, but impolite)
attribute names: ['public_field', '_protected_field', '_Alpha__private_field']
mangled name works: private (never do this)
```

Di dalam `Alpha`, `self.__private_field` berlaku seperti tertulis. Di dalam `AlphaSub`, teks yang persis sama diubah dengan nama *subkelas* menjadi `self._AlphaSub__private_field`, yang tidak ada — karena itulah muncul `AttributeError`. Orang luar dapat membaca `_protected_field`, dan program ini melakukannya, disertai catatan soal sopan santun. `vars(a)` mendaftar atribut objek dan menyingkap nama hasil pengubahan `_Alpha__private_field`, dan baris terakhir memperlihatkan bahwa siapa pun yang mengetahui nama itu dapat memakainya. Falsafah Python sering diringkas sebagai “kita semua orang dewasa yang tahu apa yang kita lakukan”: bahasa ini menandai batasnya dengan jelas dan memercayai Anda untuk tidak melanggarnya. **Tabel 7.6** menyandingkan kedua bahasa.

Tabel 7.6 Kontrol akses di Java dan padanan konvensionalnya di Python.

Maksud	Java	Python
Siapa pun boleh memakainya	`public`	`name`
Untuk subkelas (dan, di Java, paket)	`protected`	`_name` (konvensi)
Hanya untuk paket / modul ini	tanpa pengubah	`_name` di tingkat modul; tidak ikut dalam `from m import *`
Hanya untuk kelas ini	`private` (ditegakkan kompilator)	`__name` (diubah namanya, tidak ditegakkan)
Nilai hanya-baca dengan akses terhitung	field `private` + getter publik	`@property` tanpa setter

7.14 Enkapsulasi: Objek yang Menepati Janjinya

Enkapsulasi (*encapsulation*) berarti membungkus data bersama metode-metode yang mengolahnya, dan menyembunyikan data itu sehingga hanya metode-metode tersebut yang dapat mengubahnya. Tujuannya bukan kerahasiaan demi kerahasiaan. Tujuannya adalah agar kelas dapat memberikan **janji** tentang datanya — yang disebut **invarian** — dan menepatinya, karena tidak ada kode lain yang dapat melanggarnya. Gagasan ini berakar pada makalah David Parnas tahun 1972 tentang *information hiding* (penyembunyian informasi), yang berpendapat bahwa setiap modul semestinya menyembunyikan satu keputusan desain yang kemungkinan besar berubah, sehingga ketika keputusan itu berubah, hanya modul itulah yang perlu ditulis ulang [15]; Bertrand Meyer kemudian menjadikan invarian kelas pusat metodenya, *design by contract* (desain berdasarkan kontrak) [16].

Sebuah mesin cuci memiliki kapasitas, yang ditetapkan ketika mesin itu dibuat, dan muatan saat ini. Invariannya adalah

$$0 \le \ell \le c \tag{7.6}$$

dengan ℓ muatan dalam kilogram dan c kapasitasnya. Bila sembarang kode dapat memberi nilai kepada muatan secara langsung, Persamaan (7.6) hanyalah harapan. Dengan enkapsulasi ia menjadi jaminan, seperti diperlihatkan **Kode 7.26**.

Kode 7.26 `EncapsulationDemo.java`: `WashingMachine` yang field-field private-nya hanya dapat berubah melalui metode yang menjaga invarian $0 \le \ell \le c$.

```java
import java.util.Locale;

public class EncapsulationDemo {
    public static void main(String[] args) {
        WashingMachine m = new WashingMachine("W-03", 8.0);
        System.out.println(m);
        m.addLoad(5.5);
        System.out.println(m);
        try {
            m.addLoad(3.0);  // akan menjadi 8,5 kg di drum 8 kg
        } catch (IllegalArgumentException e) {
            System.out.println("Refused: " + e.getMessage());
        }
        System.out.println(m);
        m.unload();
        System.out.println(m);
        // m.loadKg = 99;  // tidak lolos kompilasi: loadKg berakses private
    }
}

class WashingMachine {
    private final String id;
    private final double capacityKg;  // ditetapkan ketika mesin dibuat
    private double loadKg = 0.0;      // invarian: 0 <= loadKg <= capacityKg

    WashingMachine(String id, double capacityKg) {
        if (capacityKg <= 0) {
            throw new IllegalArgumentException("capacity must be positive");
        }
        this.id = id;
        this.capacityKg = capacityKg;
    }

    public double getLoadKg() {
        return loadKg;
    }

    public double getCapacityKg() {
        return capacityKg;
    }

    public double freeKg() {
        return capacityKg - loadKg;
    }
```

Kode 7.26 (Lanjutan) *EncapsulationDemo.java: WashingMachine yang field-field private-nya hanya dapat berubah melalui metode yang menjaga invarian 0 \le \ell \le c.*

```
    public void addLoad(double kg) {
        if (kg <= 0) {
            throw new IllegalArgumentException("load must be positive");
        }
        if (kg > freeKg()) {
            throw new IllegalArgumentException(
                    String.format(Locale.US, "%.1f kg does not fit, only %.1f kg free", kg,
freeKg()));
        }
        loadKg += kg;
    }

    public void unload() {
        loadKg = 0.0;
    }

    @Override
    public String toString() {
        return String.format(Locale.US, "%s: %.1f / %.1f kg", id, loadKg, capacityKg);
    }
}
```

```
W-03: 0.0 / 8.0 kg
W-03: 5.5 / 8.0 kg
Refused: 3.0 kg does not fit, only 2.5 kg free
W-03: 5.5 / 8.0 kg
W-03: 0.0 / 8.0 kg
```

Setiap field bersifat `private`; `capacityKg` juga `final`, sehingga kelas itu sendiri pun tidak dapat mengubahnya setelah konstruksi. Konstruktor menolak kapasitas yang tidak masuk akal. Satu-satunya metode yang menambah muatan, `addLoad`, menolak jumlah yang tidak positif dan menolak jumlah yang tidak muat, dengan melempar `IllegalArgumentException` (Bab 5) disertai pesan yang menjelaskan alasannya. Jadi, apa pun urutan panggilan yang dilakukan bagian lain program, Persamaan (7.6) berlaku setelah setiap panggilan. Upaya `m.addLoad(3.0)` ketika sudah ada 5.5 kg di dalamnya ditolak, dan muatannya tetap 5.5 kg. Bila Anda menghapus `//` di depan `m.loadKg = 99;` di `main`, kompilator melindungi invarian itu untuk Anda:

```
EncapsulationDemo.java:17: error: loadKg has private access in WashingMachine
        m.loadKg = 99;
         ^
1 error
```

Ada manfaat kedua, yang kurang tampak tetapi sama pentingnya. Karena tidak seorang pun di luar dapat melihat `loadKg`, kita bebas mengubah cara muatan disimpan — dalam gram sebagai `long`, misalnya, untuk menghindari pembulatan — tanpa ada kelas lain yang harus berubah. Itulah persis maksud Parnas.

7.14.1 Property di Python

Di Python, konvensi `_load_kg` meminta orang luar menjauh, dan metode seperti `add_load` menegakkan invarian persis seperti di Java. Python menambahkan alat yang tidak dimiliki Java: **property**. Property adalah metode yang *dipanggil seperti field*. Tujuannya adalah membiarkan sebuah kelas bermula dengan atribut biasa dan, kemudian, menempatkan pemeriksaan di baliknya tanpa mengubah satu baris pun kode yang memakainya. **Kode 7.27** memperlihatkan property hanya-baca maupun property dengan setter yang memvalidasi.

Kode 7.27 `encapsulation_demo.py`: mesin yang sama dalam Python; `capacity_kg` adalah property hanya-baca dan `load_kg` property yang setter-nya menegakkan invarian.

```
"""encapsulation_demo.py: mesin cuci yang menjaga invariannya sendiri."""


class WashingMachine:
    def __init__(self, machine_id, capacity_kg):
        if capacity_kg <= 0:
            raise ValueError("capacity must be positive")
        self._id = machine_id
        self._capacity_kg = capacity_kg  # ditetapkan ketika mesin dibuat
        self._load_kg = 0.0              # invarian: 0 <= muatan <= kapasitas

    @property
    def capacity_kg(self):  # hanya-baca: tidak ada setter
        return self._capacity_kg

    @property
    def load_kg(self):
        return self._load_kg

    @load_kg.setter
    def load_kg(self, kg):  # penugasan melewati pemeriksaan yang sama
        if kg < 0 or kg > self._capacity_kg:
            raise ValueError(f"{kg:.1f} kg is outside 0..{self._capacity_kg:.1f} kg")
        self._load_kg = kg

    def free_kg(self):
        return self._capacity_kg - self._load_kg

    def add_load(self, kg):
        if kg <= 0:
            raise ValueError("load must be positive")
        if kg > self.free_kg():
            raise ValueError(f"{kg:.1f} kg does not fit, only {self.free_kg():.1f} kg
free")
        self._load_kg += kg

    def unload(self):
        self._load_kg = 0.0

    def __str__(self):
        return f"{self._id}: {self._load_kg:.1f} / {self._capacity_kg:.1f} kg"
```

Kode 7.27 (Lanjutan) *encapsulation_demo.py: mesin yang sama dalam Python; capacity_kg adalah property hanya-baca dan load_kg property yang setter-nya menegakkan invarian.*

```
def main():
    m = WashingMachine("W-03", 8.0)
    print(m)
    m.add_load(5.5)
    print(m)
    try:
        m.add_load(3.0)  # akan menjadi 8,5 kg di drum 8 kg
    except ValueError as e:
        print("Refused:", e)
    print(m)
    try:
        m.load_kg = 99  # tampak seperti field, tetapi menjalankan setter
    except ValueError as e:
        print("Refused:", e)
    try:
        m.capacity_kg = 20
    except AttributeError as e:
        print("AttributeError:", e)
    m.unload()
    print(m)


if __name__ == "__main__":
    main()
```

```
W-03: 0.0 / 8.0 kg
W-03: 5.5 / 8.0 kg
Refused: 3.0 kg does not fit, only 2.5 kg free
W-03: 5.5 / 8.0 kg
Refused: 99.0 kg is outside 0..8.0 kg
AttributeError: property 'capacity_kg' of 'WashingMachine' object has no setter
W-03: 0.0 / 8.0 kg
```

Dekorator `@property` di atas `def capacity_kg(self)` mengubah metode itu menjadi atribut yang dapat dibaca: pemanggil menulis `m.capacity_kg`, tanpa tanda kurung. Karena tidak ada setter yang didefinisikan, `m.capacity_kg = 20` gagal dengan `AttributeError`. Untuk `load_kg`, dekorator kedua `@load_kg.setter` mendefinisikan apa yang terjadi saat penugasan: `m.load_kg = 99` *tampak* seperti penugasan field tetapi menjalankan setter, yang memeriksa Persamaan (7.6) dan memunculkan `ValueError`. Itulah sebabnya pemrogram Python jarang menulis metode `get_` dan `set_`: mereka mulai dengan atribut biasa dan beralih ke property hanya ketika muncul sebuah aturan, dan para pemanggil tidak pernah menyadarinya.

7.15 Profitina Laundry: Hierarki Layanan dan Penetapan Harga Polimorfis

Kini kita dapat membangun ulang penetapan harga binatu sehingga menambah jenis layanan baru tidak pernah lagi berarti menyunting sebuah `switch`. Persyaratannya, dibawa dari Bab 3 dan diperluas:

- **Wash & Fold** (`WF`, cuci lipat): Rp7,000 per kilogram, minimum 3 kg, selesai dalam 48 jam.
- **Iron Only** (`IR`, setrika saja): Rp5,000 per kilogram, minimum 2 kg, selesai dalam 24 jam.
- **Wash, Dry & Iron** (`WDI`, cuci, kering, dan setrika): Rp10,000 per kilogram, minimum 3 kg, selesai dalam 72 jam.
- **Dry Clean** (`DC`, cuci kering): harga *per potong* — kemeja Rp15,000, celana panjang Rp18,000, kebaya Rp40,000, jas Rp45,000 — selesai dalam 72 jam.

- **Express** (kilat): layanan apa pun dapat dipesan kilat, dengan biaya tambahan 50%, selesai dalam seperempat waktu biasanya tetapi tidak pernah kurang dari 6 jam.
- **Anggota** mendapat potongan 10% dari pesanan.

Layanan berbasis berat mengenakan biaya atas *berat terhitung* $\widehat{w}$: berat sebenarnya w, dinaikkan ke minimum $w_{\min}$ bila lebih kecil, lalu dibulatkan ke atas ke setengah kilogram berikutnya:

$$\widehat{w} = \frac{\lceil 2\max(w, w_{\min}) \rceil}{2} \tag{7.7}$$

dan harganya adalah berat terhitung dikalikan tarif r, dalam rupiah bulat:

$$P_{\text{berat}} = \text{round}(r \cdot \widehat{w}) \tag{7.8}$$

Sebagai contoh, 4.2 kg Wash & Fold: $\max(4.2,3) = 4.2$, $2 \times 4.2 = 8.4$, dibulatkan ke atas menjadi 9, dibagi dua menjadi $\widehat{w} = 4.5$ kg, sehingga $P = 7000 \times 4.5 = 31{,}500$. Versi kilat dari layanan apa pun dengan harga dasar P berharga, dengan pembagian bilangan bulat,

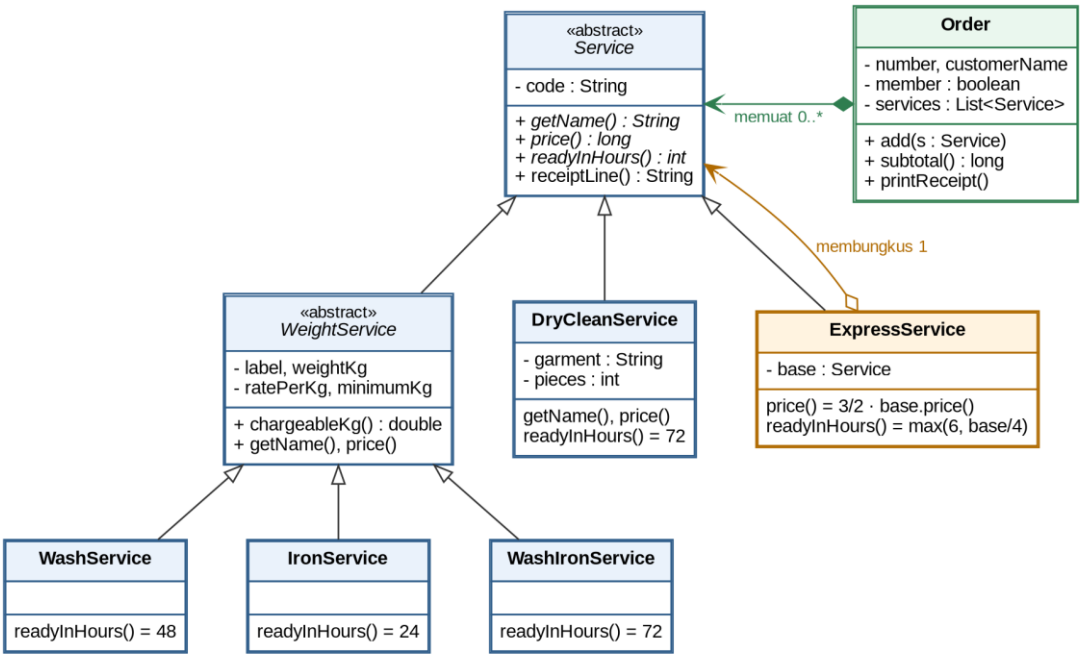
$$P_{\text{kilat}} = \lfloor \frac{3P}{2} \rfloor \tag{7.9}$$

dan sebuah pesanan berisi layanan $s_1, \ldots, s_n$ memiliki subtotal dan total anggota

$$T = \sum_{i=1}^{n} P\,(s_i), \qquad T_{\text{anggota}} = T - \lfloor \frac{T}{10} \rfloor \tag{7.10}$$

Penjumlahan pada Persamaan (7.10) adalah tempat polimorfisme membayar lunas jasanya: $P(s_i)$ dihitung dengan rumus yang berbeda untuk setiap jenis layanan, namun kode yang menjumlahkannya adalah satu perulangan yang tidak pernah menanyakan jenis apa yang sedang dipegangnya.

Gambar 7.11 adalah rancangannya.



Gambar 7.11 Diagram kelas UML mesin harga Profitina Laundry. `Service` dan `WeightService` abstrak; lima layanan konkret memperluas keduanya; `ExpressService` sekaligus *adalah* `Service` dan *membungkus* sebuah `Service` (belah ketupat kosong); sebuah `Order` memegang layanan dalam jumlah berapa pun (belah ketupat penuh).

- **`Service`** (abstrak) memuat apa yang dimiliki setiap layanan: sebuah kode. Ia mendeklarasikan tiga metode abstrak — `getName()`, `price()`, `readyInHours()` — dan menyediakan satu metode templat, `receiptLine()`, yang menyusun satu baris nota darinya.
- **`WeightService`** (abstrak) memuat segala sesuatu yang dimiliki bersama oleh layanan per kilogram: label, berat, tarif, dan minimum, aturan berat terhitung pada Persamaan (7.7), dan aturan harga pada Persamaan (7.8). Ia hanya membiarkan `readyInHours()` tetap abstrak.
- **`WashService`**, **`IronService`**, **`WashIronService`** kini hampir kosong: sebuah konstruktor yang meneruskan angka-angka yang tepat ke `super(...)`, dan sebuah waktu selesai. `switch` dari Bab 3 telah menjadi tiga *override* satu baris.
- **`DryCleanService`** memperluas `Service` secara langsung, karena harganya bergantung pada jumlah potong, bukan berat.
- **`ExpressService`** memperluas `Service` dan memegang sebuah `Service` bernama `base`. `price()` miliknya menerapkan Persamaan (7.9) pada `base.price()`; waktu selesainya seperempat waktu milik `base`. Karena ia menerima `Service` *apa pun*, layanan kilat berlaku untuk cuci kering, untuk layanan yang ditambahkan tahun depan, bahkan untuk layanan kilat lain (pertanyaan untuk Soal Latihan 7.9).
- **`Order`** memegang `List<Service>` dan menghitung subtotal, potongan, dan total pada Persamaan (7.10) dengan perulangan atas list itu.

Kode 7.28 adalah programnya secara lengkap.

Kode 7.28 `LaundryPricing.java`: hierarki layanan Profitina Laundry, *decorator* kilat, dan pesanan yang menghitung semua harga secara polimorfis.

```
import java.util.ArrayList;
import java.util.List;
import java.util.Locale;

public class LaundryPricing {
    public static void main(String[] args) {
        Order order = new Order("ORD-0712", "Siti Aminah", true);
        order.add(new WashService(4.2));
        order.add(new IronService(1.3));
        order.add(new DryCleanService("suit", 1));
        order.add(new DryCleanService("shirt", 3));
        order.add(new ExpressService(new WashIronService(2.6)));
        order.add(new ExpressService(new DryCleanService("kebaya", 1)));
        order.printReceipt();
    }
}

abstract class Service {
    private final String code;

    protected Service(String code) {
        this.code = code;
    }

    String getCode() {
        return code;
    }

    abstract String getName();       // yang dilihat pelanggan di nota
    abstract long price();           // dalam rupiah bulat
    abstract int readyInHours();     // waktu selesai yang dijanjikan

    // Metode templat: tata letak tetap, bagian yang berubah disediakan subkelas.
    String receiptLine() {
        return String.format(Locale.US, "%-7s %-32s %10s %4d h",
                code, getName(), rupiah(price()), readyInHours());
    }

    static String rupiah(long amount) {
        return String.format(Locale.US, "Rp%,d", amount);
    }
}
```

Kode 7.28 (Lanjutan) *LaundryPricing.java: hierarki layanan Profitina Laundry, decorator kilat, dan pesanan yang menghitung semua harga secara polimorfis.*

```
abstract class WeightService extends Service {
    private final String label;
    private final double weightKg;
    private final long ratePerKg;
    private final double minimumKg;

    protected WeightService(String code, String label, double weightKg,
                            long ratePerKg, double minimumKg) {
        super(code);
        if (weightKg <= 0) {
            throw new IllegalArgumentException("weight must be positive");
        }
        this.label = label;
        this.weightKg = weightKg;
        this.ratePerKg = ratePerKg;
        this.minimumKg = minimumKg;
    }

    // Paling sedikit minimum, dibulatkan ke atas ke 0,5 kg berikutnya.
    double chargeableKg() {
        return Math.ceil(Math.max(weightKg, minimumKg) * 2) / 2;
    }

    @Override
    String getName() {
        return String.format(Locale.US, "%s %.1f kg", label, weightKg);
    }

    @Override
    long price() {
        return Math.round(chargeableKg() * ratePerKg);
    }
}

class WashService extends WeightService {
    WashService(double weightKg) {
        super("WF", "Wash & Fold", weightKg, 7000, 3.0);
    }

    @Override
    int readyInHours() {
        return 48;
    }
}
```

Kode 7.28 (Lanjutan) *LaundryPricing.java: hierarki layanan Profitina Laundry, decorator kilat, dan pesanan yang menghitung semua harga secara polimorfis.*

```
class IronService extends WeightService {
    IronService(double weightKg) {
        super("IR", "Iron Only", weightKg, 5000, 2.0);
    }

    @Override
    int readyInHours() {
        return 24;
    }
}

class WashIronService extends WeightService {
    WashIronService(double weightKg) {
        super("WDI", "Wash, Dry & Iron", weightKg, 10000, 3.0);
    }

    @Override
    int readyInHours() {
        return 72;
    }
}

class DryCleanService extends Service {
    private final String garment;
    private final int pieces;

    DryCleanService(String garment, int pieces) {
        super("DC");
        if (pieces <= 0) {
            throw new IllegalArgumentException("pieces must be positive");
        }
        this.garment = garment;
        this.pieces = pieces;
    }

    private long pricePerPiece() {
        return switch (garment) {
            case "shirt" -> 15000;
            case "trousers" -> 18000;
            case "kebaya" -> 40000;
            case "suit" -> 45000;
            default -> throw new IllegalArgumentException("no dry-clean price for " +
garment);
        };
    }
```

Kode 7.28 (Lanjutan) *LaundryPricing.java: hierarki layanan Profitina Laundry, decorator kilat, dan pesanan yang menghitung semua harga secara polimorfis.*

```
    @Override
    String getName() {
        return "Dry Clean " + pieces + " x " + garment;
    }

    @Override
    long price() {
        return pieces * pricePerPiece();
    }

    @Override
    int readyInHours() {
        return 72;
    }
}

// Express ADALAH Service (sehingga Order menerimanya) dan MEMILIKI Service (yang
dipercepatnya).
class ExpressService extends Service {
    private final Service base;

    ExpressService(Service base) {
        super("EX-" + base.getCode());
        this.base = base;
    }

    @Override
    String getName() {
        return "Express " + base.getName();
    }

    @Override
    long price() {
        return base.price() * 3 / 2;  // biaya tambahan 50%
    }

    @Override
    int readyInHours() {
        return Math.max(6, base.readyInHours() / 4);
    }
}
```

Kode 7.28 (Lanjutan) *LaundryPricing.java: hierarki layanan Profitina Laundry, decorator kilat, dan pesanan yang menghitung semua harga secara polimorfis.*

```
class Order {
    private final String number;
    private final String customerName;
    private final boolean member;
    private final List<Service> services = new ArrayList<>();

    Order(String number, String customerName, boolean member) {
        this.number = number;
        this.customerName = customerName;
        this.member = member;
    }

    void add(Service s) {
        services.add(s);
    }

    long subtotal() {
        long sum = 0;
        for (Service s : services) {
            sum += s.price();  // dynamic dispatch memilih price() yang tepat
        }
        return sum;
    }

    long discount() {
        return member ? subtotal() * 10 / 100 : 0;
    }

    int readyInHours() {
        int latest = 0;
        for (Service s : services) {
            latest = Math.max(latest, s.readyInHours());
        }
        return latest;
    }
```

Kode 7.28 (Lanjutan) *LaundryPricing.java: hierarki layanan Profitina Laundry, decorator kilat, dan pesanan yang menghitung semua harga secara polimorfis.*

```
    void printReceipt() {
        System.out.println("PROFITINA LAUNDRY  order " + number + "  customer " +
customerName
                + (member ? " (member)" : ""));
        for (Service s : services) {
            System.out.println(s.receiptLine());
        }
        System.out.printf("%-40s %10s%n", "Subtotal", Service.rupiah(subtotal()));
        if (member) {
            System.out.printf("%-40s %10s%n", "Member discount 10%", "-" +
Service.rupiah(discount()));
        }
        System.out.printf("%-40s %10s%n", "TOTAL", Service.rupiah(subtotal() -
discount()));
        System.out.println("Everything ready in " + readyInHours() + " h");
    }
}
```

```
PROFITINA LAUNDRY  order ORD-0712  customer Siti Aminah (member)
WF      Wash & Fold 4.2 kg                Rp31,500   48 h
IR      Iron Only 1.3 kg                  Rp10,000   24 h
DC      Dry Clean 1 x suit                Rp45,000   72 h
DC      Dry Clean 3 x shirt               Rp45,000   72 h
EX-WDI  Express Wash, Dry & Iron 2.6 kg   Rp45,000   18 h
EX-DC   Express Dry Clean 1 x kebaya      Rp60,000   18 h
Subtotal                                 Rp236,500
Member discount 10%                      -Rp23,650
TOTAL                                    Rp212,850
Everything ready in 72 h
```

Periksalah setiap baris terhadap rumus-rumusnya. Wash & Fold 4.2 kg adalah Rp31,500, seperti dihitung di atas. Iron Only 1.3 kg dinaikkan ke minimum 2 kg: $5000 \times 2 = 10{,}000$. Satu jas Rp45,000; tiga kemeja $3 \times 15{,}000 = 45{,}000$. Wash, Dry & Iron 2.6 kg dinaikkan ke 3 kg, $10{,}000 \times 3 = 30{,}000$, dan layanan kilat menjadikannya $\lfloor 3 \times 30{,}000/2 \rfloor = 45{,}000$, selesai dalam $\max(6, 72/4) = 18$ jam. Kebaya kilat $\lfloor 3 \times 40{,}000/2 \rfloor = 60{,}000$. Subtotalnya $31{,}500 + 10{,}000 + 45{,}000 + 45{,}000 + 45{,}000 + 60{,}000 = 236{,}500$; potongan anggota $\lfloor 236{,}500/10 \rfloor = 23{,}650$; totalnya 212,850. Pesanan itu selesai ketika barang yang paling lambat selesai: 72 jam.

Sekarang lihatlah apa yang *tidak* dimuat kode itu. `Order.subtotal()` hanya empat baris dan tidak menyebut satu subkelas pun. `Service.receiptLine()` menyusun setiap baris dan tidak menyebut satu subkelas pun. Tidak ada `switch` atas kode dan tidak ada `if (express)` di mana pun. Satu-satunya `switch` yang tersisa, di dalam `DryCleanService`, memilih sebuah *angka* dari daftar harga, yang merupakan data, bukan perilaku. Ketika binatu menambahkan cuci sepatu, kita akan menulis satu kelas baru dan **tidak** mengubah apa pun yang sudah ada — seperti diperagakan Soal Latihan 7.9 dan penyelesaiannya. Sifat itu, terbuka untuk perluasan tetapi tertutup untuk perubahan, adalah buah praktis dari seluruh bab ini. Dan bila kita melupakan sebagian kontraknya, kompilator mengingatkan: hapus `readyInHours()` dari `IronService` dan `javac` menolak kelas itu, sambil menyebut metode yang hilang:

```
LaundryPricing.java:89: error: IronService is not abstract and does not override abstract
method readyInHours() in Service
class IronService extends WeightService {
^
1 error
```

Rancangan ini juga memakai setiap tingkat akses dengan sengaja. Field-field-nya `private`, sehingga tidak ada subkelas yang dapat merusak data kelas lain; subkelas menjangkau data induknya hanya melalui metode seperti `getCode()`. Kelas-kelasnya package-private, karena hanya program ini yang memakainya. Pembantu `rupiah` bersifat `static` dan memberikan `Locale.US`, sehingga pemisah ribuannya koma di setiap komputer, persis sama dengan keluaran Python.

Kode 7.29 adalah versi Python-nya, dan keluarannya identik.

Kode 7.29 `laundry_pricing.py`: mesin harga yang sama dalam Python.

```python
"""laundry_pricing.py: hierarki layanan Profitina Laundry dan penetapan harga
polimorfis."""
import math
from abc import ABC, abstractmethod


def rupiah(amount):
    return f"Rp{amount:,}"


class Service(ABC):
    def __init__(self, code):
        self._code = code

    @property
    def code(self):
        return self._code

    @abstractmethod
    def name(self): ...           # yang dilihat pelanggan di nota

    @abstractmethod
    def price(self): ...          # dalam rupiah bulat

    @abstractmethod
    def ready_in_hours(self): ...  # waktu selesai yang dijanjikan

    # Metode templat: tata letak tetap, bagian yang berubah disediakan subkelas.
    def receipt_line(self):
        return (f"{self._code:<7} {self.name():<32} {rupiah(self.price()):>10} "
                f"{self.ready_in_hours():4d} h")


class WeightService(Service, ABC):
    def __init__(self, code, label, weight_kg, rate_per_kg, minimum_kg):
        super().__init__(code)
        if weight_kg <= 0:
            raise ValueError("weight must be positive")
        self._label = label
        self._weight_kg = weight_kg
        self._rate_per_kg = rate_per_kg
        self._minimum_kg = minimum_kg

    def chargeable_kg(self):
        # Paling sedikit minimum, dibulatkan ke atas ke 0,5 kg berikutnya.
        return math.ceil(max(self._weight_kg, self._minimum_kg) * 2) / 2
```

Kode 7.29 (Lanjutan) *laundry_pricing.py: mesin harga yang sama dalam Python.*

```
    def name(self):
        return f"{self._label} {self._weight_kg:.1f} kg"

    def price(self):
        return round(self.chargeable_kg() * self._rate_per_kg)


class WashService(WeightService):
    def __init__(self, weight_kg):
        super().__init__("WF", "Wash & Fold", weight_kg, 7000, 3.0)

    def ready_in_hours(self):
        return 48


class IronService(WeightService):
    def __init__(self, weight_kg):
        super().__init__("IR", "Iron Only", weight_kg, 5000, 2.0)

    def ready_in_hours(self):
        return 24


class WashIronService(WeightService):
    def __init__(self, weight_kg):
        super().__init__("WDI", "Wash, Dry & Iron", weight_kg, 10000, 3.0)

    def ready_in_hours(self):
        return 72


class DryCleanService(Service):
    PRICE_PER_PIECE = {"shirt": 15000, "trousers": 18000, "kebaya": 40000, "suit": 45000}

    def __init__(self, garment, pieces):
        super().__init__("DC")
        if pieces <= 0:
            raise ValueError("pieces must be positive")
        if garment not in self.PRICE_PER_PIECE:
            raise ValueError(f"no dry-clean price for {garment}")
        self._garment = garment
        self._pieces = pieces

    def name(self):
        return f"Dry Clean {self._pieces} x {self._garment}"
```

Kode 7.29 (Lanjutan) *laundry_pricing.py: mesin harga yang sama dalam Python.*

```
    def price(self):
        return self._pieces * self.PRICE_PER_PIECE[self._garment]

    def ready_in_hours(self):
        return 72


# Express ADALAH Service (sehingga Order menerimanya) dan MEMILIKI Service (yang
dipercepatnya).
class ExpressService(Service):
    def __init__(self, base):
        super().__init__("EX-" + base.code)
        self._base = base

    def name(self):
        return "Express " + self._base.name()

    def price(self):
        return self._base.price() * 3 // 2  # biaya tambahan 50%

    def ready_in_hours(self):
        return max(6, self._base.ready_in_hours() // 4)


class Order:
    def __init__(self, number, customer_name, member):
        self._number = number
        self._customer_name = customer_name
        self._member = member
        self._services = []

    def add(self, service):
        self._services.append(service)

    def subtotal(self):
        return sum(s.price() for s in self._services)  # setiap s memilih price() miliknya
sendiri

    def discount(self):
        return self.subtotal() * 10 // 100 if self._member else 0

    def ready_in_hours(self):
        return max((s.ready_in_hours() for s in self._services), default=0)
```

Kode 7.29 (Lanjutan) *laundry_pricing.py: mesin harga yang sama dalam Python.*

```
    def print_receipt(self):
        member = " (member)" if self._member else ""
        print(f"PROFITINA LAUNDRY  order {self._number}  customer
{self._customer_name}{member}")
        for s in self._services:
            print(s.receipt_line())
        print(f"{'Subtotal':<40} {rupiah(self.subtotal()):>10}")
        if self._member:
            print(f"{'Member discount 10%':<40} {'-' + rupiah(self.discount()):>10}")
        print(f"{'TOTAL':<40} {rupiah(self.subtotal() - self.discount()):>10}")
        print(f"Everything ready in {self.ready_in_hours()} h")


def main():
    order = Order("ORD-0712", "Siti Aminah", True)
    order.add(WashService(4.2))
    order.add(IronService(1.3))
    order.add(DryCleanService("suit", 1))
    order.add(DryCleanService("shirt", 3))
    order.add(ExpressService(WashIronService(2.6)))
    order.add(ExpressService(DryCleanService("kebaya", 1)))
    order.print_receipt()


if __name__ == "__main__":
    main()
```

```
PROFITINA LAUNDRY  order ORD-0712  customer Siti Aminah (member)
WF      Wash & Fold 4.2 kg                Rp31,500   48 h
IR      Iron Only 1.3 kg                  Rp10,000   24 h
DC      Dry Clean 1 x suit                Rp45,000   72 h
DC      Dry Clean 3 x shirt               Rp45,000   72 h
EX-WDI  Express Wash, Dry & Iron 2.6 kg   Rp45,000   18 h
EX-DC   Express Dry Clean 1 x kebaya      Rp60,000   18 h
Subtotal                                 Rp236,500
Member discount 10%                      -Rp23,650
TOTAL                                    Rp212,850
Everything ready in 72 h
```

Strukturnya sama kelas demi kelas. `WeightService(Service, ABC)` mencantumkan `ABC` sekali lagi untuk menegaskan keabstrakannya. `//` adalah pembagian bilangan bulat di Python, yang berpadanan dengan `/` pada nilai `long` di Java. `round` di Python dan `Math.round` di Java memperlakukan nilai tepat setengah secara berbeda (Python membulatkannya ke bilangan genap terdekat), tetapi dengan harga-harga binatu ini hasil kali $r \cdot \hat{w}$ selalu bilangan bulat, sehingga keduanya sepakat. `f"{amount:,}"` menyisipkan koma di antara ribuan tanpa memedulikan pengaturan regional komputer, sehingga tidak diperlukan *locale*. Dan jantung program ini, `sum(s.price() for s in self._services)`, adalah penjumlahan pada Persamaan (7.10) yang ditulis hampir persis seperti matematika menuliskannya.

7.16 Profitina dalam Praktik

Binatu adalah ruang kelas kita; produk Profitina yang sebenarnya memakai gagasan yang sama dalam skala yang lebih besar. Profitina adalah platform visibilitas AI: ia memindai situs web sebuah merek dan mengukur seberapa siap situs itu untuk dikutip oleh mesin penjawab AI — disiplin yang disebut *generative engine optimization* (GEO). Sebuah pemindaian memeriksa banyak **sinyal** yang saling bebas, misalnya apakah kontennya dapat diekstraksi dengan bersih, apakah halamannya memuat data terstruktur schema.org, dan apakah `robots.txt` mengizinkan mesin penjawab masuk.

Bentuk pemindai itu adalah bentuk **Gambar 7.11**. Satu abstraksi — sebuah sinyal — menjanjikan beberapa operasi: mengukur sebuah halaman, memberi skor, dan menjelaskan skor itu dengan kata-kata yang dapat dipahami pemilik merek. Banyak kelas sinyal konkret mengimplementasikan janji itu, masing-masing diuji sendiri. Kode yang menjalankan pemindaian adalah satu perulangan atas daftar sinyal, persis seperti `Order.subtotal()` adalah satu perulangan atas layanan, dan ia tidak pernah menanyakan sinyal mana yang sedang dipegangnya. Sinyal baru berarti satu kelas baru, bukan suntingan pada perulangan itu; perilaku yang dimiliki bersama oleh beberapa sinyal ditempatkan dalam objek pembantu yang mereka pakai melalui komposisi, dengan alasan-alasan di Subbab 7.11.

Enkapsulasi pun berperan penting di sana. Profitina mematuhi Undang-Undang Pelindungan Data Pribadi (UU PDP No. 27/2022): data akun di-*hash* dengan Argon2id, persetujuan dinyatakan secara eksplisit, dan pengguna dapat mengekspor atau menghapus seluruh datanya. Kode semacam itu menyimpan data pribadi di balik sekumpulan kecil metode yang menegakkan aturannya, dalam semangat **Kode 7.26**. Dan, seperti di seluruh Profitina, skor-skor itu adalah nasihat: platform menjelaskan apa yang diukurnya; penggunalah yang memutuskan apa yang akan dilakukan.

7.17 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya

1. Mewarisi demi memakai ulang kode padahal kalimat is-a-nya salah. `class Order extends ArrayList<Service>` menghemat beberapa baris dan memberi setiap pesanan `clear()`, `remove()`, serta empat puluh metode lain yang tidak diinginkan siapa pun. *Hindari:* ucapkan kalimat is-a keras-keras; bila gagal, simpan list itu di dalam field private (Subbab 7.11).

2. Melakukan overloading padahal maksudnya overriding. Di Java, `public boolean equals(Customer other)` *tidak* meng-*override* `equals(Object)`; ia menambahkan *overload* yang tidak pernah dipanggil `HashSet`. *Hindari:* selalu tulis `@Override`. Dengan anotasi itu, kesalahannya menjadi galat kompilasi:

```
BadEquals.java:9: error: method does not override or implement a method from a supertype
    @Override
    ^
1 error
```

Nama yang salah eja (`Start` untuk `start`) tertangkap dengan cara yang sama.

3. Melupakan `super().__init__()` di Python. Field-field superkelas tidak pernah tercipta, dan galatnya baru muncul jauh kemudian sebagai `AttributeError` (blok ketiga keluaran **Kode 7.6**). *Hindari:* jadikan `super().__init__(...)` baris pertama setiap `__init__` subkelas.

4. Mendeklarasikan ulang field superkelas di subkelas Java. Objeknya lalu memegang dua field bernama sama, dan field mana yang Anda dapatkan bergantung pada tipe statis (**Kode 7.9**). *Hindari:* jangan pernah mendeklarasikan ulang; bila subkelas memerlukan nilai yang berbeda, berikan nilai itu ke konstruktor superkelas atau *override* sebuah getter.

5. Mengira metode atau field statis bersifat polimorfis. `m.kind()` memanggil `Machine.kind()` sekalipun `m` memegang `Washer`. *Hindari:* panggil metode statis melalui nama kelasnya, `Washer.kind()`, sehingga kode mengatakan apa yang dilakukannya.

6. Melakukan downcast tanpa memeriksa. `(Dryer) floor[0]` lolos kompilasi dan melempar `ClassCastException` saat program berjalan. *Hindari:* uji dengan `if (x instanceof Dryer d)`;

lebih baik lagi, tanyakan apakah metode itu semestinya dideklarasikan di superkelas sehingga tidak perlu *cast* sama sekali.

7. Meng-override `equals` tanpa `hashCode`. Koleksi berbasis hash lalu berperilaku keliru secara diam-diam di Java. *Hindari: override* keduanya, dari field-field yang sama (Subbab 7.8). Di Python, definisikan `__hash__` setiap kali Anda mendefinisikan `__eq__` pada kelas yang objeknya dimasukkan ke himpunan atau dijadikan kunci kamus.

8. Membuat field `protected` "agar subkelas dapat memakainya". Setiap subkelas, di paket mana pun, lalu dapat melanggar invariannya. *Hindari:* biarkan field tetap `private` dan berikan subkelas metode `protected` yang menjaga aturannya.

9. Membaca `super()` sebagai "induk saya" dalam pewarisan jamak Python. Pada berlian, `super()` menuju kelas berikutnya dalam MRO, yang bisa saja saudaranya (**Kode 7.20**). *Hindari:* cetak `Class.__mro__` bila ragu, dan rancang mixin agar memanggil `super()` secara kooperatif.

10. Memanggil metode yang dapat di-override dari konstruktor. Karena *dynamic dispatch*, konstruktor superkelas yang memanggil `describe()` menjalankan `describe()` milik *subkelas* — sebelum konstruktor subkelas mengisi field-field subkelas. *Hindari:* konstruktor sebaiknya hanya memanggil metode `private` atau `final` [6].

7.18 Ringkasan Bab

- **Pewarisan** menghapus kode kembar: bagian bersama beberapa kelas naik ke sebuah **superkelas**, dan setiap **subkelas** hanya mendeklarasikan apa yang berbeda (Java `class C extends B`, Python `class C(B)`). Uji rancangannya adalah kalimat "sebuah C *adalah sejenis* B".
- Subkelas **meng-*override*** metode warisan dengan mendefinisikannya ulang (Java: nama dan tipe parameter sama, diperiksa oleh `@Override`; Python: nama sama). ***Overloading*** — nama sama, parameter berbeda — diselesaikan saat kompilasi; Python menggantinya dengan argumen bawaan dan argumen kata kunci.
- **Konstruktor berantai ke atas**: bagian superkelas dibangun lebih dahulu. Java menyisipkan `super()` bila Anda tidak menulis `super(...)`; Python menjalankan `__init__` superkelas hanya bila Anda memanggil `super().__init__(...)`. `super.method()` / `super().method()` menjangkau versi yang di-*override*.
- **Polimorfisme**: satu panggilan menjalankan kode yang berbeda untuk objek yang berbeda. Di Java **tipe statis** menentukan apakah sebuah panggilan lolos kompilasi dan **tipe dinamis** menentukan *override* mana yang berjalan (***dynamic dispatch***, Persamaan (7.1)), yang diimplementasikan dengan satu tabel metode per kelas; field, metode statis, dan metode private tidak di-*dispatch*. Python mencari nama di sepanjang **MRO** saat program berjalan.
- Setiap kelas diturunkan dari `Object` / `object`; *override* `toString`/`__str__`, serta `equals` bersama `hashCode` (`__eq__` bersama `__hash__`).
- **Kelas abstrak** tidak dapat diinstansiasi; ia menghimpun keadaan dan kode bersama, menyatakan apa yang wajib diimplementasikan subkelas, dan memungkinkan **metode templat** seperti `describe()` dan `receiptLine()`. Utamakan superkelas bersama yang abstrak daripada mewarisi kelas konkret.
- **Utamakan komposisi daripada pewarisan**: pewarisan mengikat subkelas pada implementasi superkelasnya (kelas dasar yang rapuh). Mewarislah hanya bila **prinsip substitusi Liskov** (Persamaan (7.4)) berlaku. Layanan kilat melakukan keduanya — ia memperluas sekaligus membungkus `Service`, sebuah *decorator*.

- Java mengizinkan **satu superkelas dan banyak antarmuka**, dan default yang bertentangan harus diselesaikan secara eksplisit. Python mengizinkan **pewarisan jamak**, yang diurutkan oleh **linearisasi C3** (Persamaan (7.5)) dan diikuti oleh `super()`.
- **Kontrol akses**: `public`, `protected`, package-private, dan `private` di Java (**Tabel 7.4**) — pilihlah yang paling ketat yang masih berfungsi. Konvensi Python adalah `name`, `_name`, dan `__name`. **Enkapsulasi** memungkinkan kelas menjaga **invarian**-nya; `@property` menempatkan pemeriksaan di balik sintaksis atribut.
- Di binatu, satu perulangan kini menghitung harga setiap jenis layanan, dan layanan baru adalah kelas baru, bukan label `case` baru.

7.19 Soal Latihan

Soal Latihan 7.1 (is-a atau has-a). Untuk setiap pasangan, tentukan apakah yang pertama sebaiknya *mewarisi* yang kedua, *memegang* objek yang kedua, atau tidak keduanya, dan berikan alasan dalam satu kalimat. (a) `ExpressService`, `Service`. (b) `Order`, `Customer`. (c) `DryCleanService`, `Service`. (d) `WashingMachine`, `Machine`. (e) `Receipt`, `Order`. (f) `Customer`, `Order`. (g) `ProfitinaLaundry`, `WashingMachine`.

Soal Latihan 7.2 (penelusuran). Untuk `TreeIntDB` di **Kode 7.3**, gambarlah pohon setelah penyisipan 42, 7, 19, 3, 25, lalu hasilkan ulang jumlah perbandingan 22 dengan mendaftar, untuk setiap pencarian dari ketujuh pencarian yang dilakukan program, simpul-simpul yang dikunjungi. Kemudian ramalkan jumlahnya untuk `LinearIntDB` seandainya kunci-kunci ditambahkan dengan urutan 3, 7, 19, 25, 42, dan periksalah jawaban Anda dengan mengubah programnya.

Soal Latihan 7.3 (overloading atau overriding). Dalam kode Java berikut, tentukan untuk setiap baris bertanda (1)–(5) metode mana yang berjalan dan apa yang dicetak, lalu beri label setiap hubungan sebagai *overloading* atau *overriding*.

```
class Bag {
    String weigh()            { return "Bag.weigh()"; }
    String weigh(double kg)   { return "Bag.weigh(double)"; }
}
class LaundryBag extends Bag {
    @Override String weigh()  { return "LaundryBag.weigh()"; }
    String weigh(int grams)   { return "LaundryBag.weigh(int)"; }
}
Bag b = new LaundryBag();
LaundryBag lb = new LaundryBag();
b.weigh();        // (1)
b.weigh(2.5);     // (2)
b.weigh(2500);    // (3)
lb.weigh(2500);   // (4)
lb.weigh(2.5);    // (5)
```

Soal Latihan 7.4 (hierarki buatan Anda sendiri). Implementasikan hierarki mahasiswa di sisi kanan **Gambar 7.5** dalam Java dan dalam Python. `Student` dan `TaughtStudent` abstrak. Biaya satu semester adalah: mahasiswa sarjana, Rp300,000 per SKS; mahasiswa pascasarjana, Rp500,000 per SKS; mahasiswa tidak tetap, pendaftaran Rp250,000 ditambah Rp400,000 per SKS; mahasiswa riset, tetap Rp9,000,000. Cetak satu baris per mahasiswa beserta totalnya untuk Ayu (sarjana, 20 SKS), Bagus (pascasarjana, 12), Citra (tidak tetap, 6), dan Dewi (riset).

Soal Latihan 7.5 (ramalkan keluarannya). Tanpa menjalankannya, ramalkan setiap baris yang dicetak blok pertama **Kode 7.9** bila (a) `Washer` *tidak* mendeklarasikan ulang `label`; (b) `describe()` di `Machine` dijadikan `private` dan dipanggil dari metode publik baru `info()` di

`Machine`; (c) `kind()` adalah metode instans, bukan `static`. Dasarkan setiap jawaban pada **Tabel 7.2**.

Soal Latihan 7.6 (C3 dengan tangan). Diberikan

```
class A: ...
class B(A): ...
class C(A): ...
class D(B, C): ...
class E(C): ...
class F(D, E): ...
```

hitunglah $L[F]$ dengan Persamaan (7.5), dengan memperlihatkan setiap langkah penggabungannya. Lalu periksalah jawaban Anda dengan `print([k.__name__ for k in F.__mro__])`. Terakhir, jelaskan mengapa `class G(A, B)` memunculkan `TypeError`.

Soal Latihan 7.7 (akses). Dalam susunan **Gambar 7.10**, misalkan `Alpha` juga mendeklarasikan `protected void touch()` dan `static protected int counter`. Untuk setiap pernyataan berikut, tentukan apakah ia lolos kompilasi, dan mengapa: (a) di `Beta`, `new Alpha().touch();` (b) di `AlphaSub`, `touch();` (c) di `AlphaSub`, `new Alpha().touch();` (d) di `AlphaSub`, `new AlphaSub().touch();` (e) di `Gamma`, `new AlphaSub().touch();`.

Soal Latihan 7.8 (rancang ulang). Versi superkelas konkret dari basis data (**Kode 7.15**) membuat `TreeDB` mewarisi `LinearDB`. Daftarlah setiap masalah yang ditimbulkannya, lalu rancang ulang dengan dua cara: dengan superkelas abstrak, dan dengan antarmuka `KeyStore` ditambah kelas pembantu yang dipakai melalui komposisi. Rancangan mana yang akan Anda pilih bila implementasi ketiga, tabel hash, sudah direncanakan?

Soal Latihan 7.9 (perluas binatunya). Tanpa mengubah `LaundryPricing.java` atau `laundry_pricing.py` sedikit pun, tambahkan (a) `ShoeCleaningService` seharga Rp25,000 per pasang dan selesai dalam 72 jam, dan (b) `PromoService` yang membungkus layanan apa pun dan memotong harganya sebesar persentase tertentu. Hitunglah harga pesanan untuk Budi (bukan anggota) yang berisi 3.4 kg Wash & Fold, dua pasang sepatu, sepasang sepatu dengan potongan 20%, dan Wash & Fold kilat 5.0 kg dengan potongan 20%. Apakah berpengaruh apakah promosi diterapkan di dalam atau di luar layanan kilat? Berapa harga `new ExpressService(new ExpressService(s))`?

Soal Latihan 7.10 (*name mangling* di Python). Ramalkan keluaran `print(list(vars(Child())))` untuk

```
class Parent:
    def __init__(self):
        self.__secret = 1
        self._hint = 2
class Child(Parent):
    def __init__(self):
        super().__init__()
        self.__secret = 3
```

dan jelaskan mengapa penugasan oleh anak tidak menimpa nilai milik induknya.

Soal Latihan 7.11 (substitusi). Binatu diminta menambahkan `FlatRateWash`, subkelas `WashService` yang pelanggannya membayar tetap Rp50,000 per kantong “berapa pun beratnya”, dan `chargeableKg()` di-*override* agar mengembalikan 0 sehingga nota tidak menampilkan berat. Jelaskan, dengan Persamaan (7.4), pemanggil mana yang dapat dirusak oleh subkelas ini, dan usulkan rancangan yang lebih baik.

Soal Latihan 7.12 (property). Tulis ulang `Customer` dari **Kode 7.12** sehingga `phone` menjadi property hanya-baca, `name` menjadi property yang setter-nya membuang spasi di sekelilingnya

dan menolak nama kosong, dan kelas itu tetap berfungsi di dalam himpunan. Jelaskan mengapa khususnya `phone` tidak boleh dapat diubah begitu objeknya berada di dalam himpunan.

Soal Latihan 7.13 (konstruktor yang memanggil metode yang dapat di-override). Tulislah kelas Java `Item` yang konstruktornya mencetak `describe()`, dan subkelas `Towel` yang meng-*override* `describe()` untuk mencetak field `color` yang diisi di konstruktor `Towel`. Jalankan `new Towel("white")` dan jelaskan keluarannya dengan **Gambar 7.2**. Lakukan hal yang sama di Python dan bandingkan.

7.20 Penyelesaian Soal Latihan

Penyelesaian 7.1. (a) *Mewarisi dan memegang*: layanan kilat adalah sejenis layanan (sebuah pesanan harus menerimanya) dan ia memiliki layanan yang dipercepatnya — *decorator* dari Subbab 7.15. (b) *Memegang*: sebuah pesanan memiliki pelanggan; pesanan bukan pelanggan. (c) *Mewarisi*: cuci kering adalah sejenis layanan, dan setiap pemanggil `price()` puas dengannya. (d) *Mewarisi*, asalkan setiap operasi `Machine` masuk akal bagi mesin cuci. (e) *Memegang* (atau tidak keduanya): nota dihasilkan dari pesanan dan boleh merujuknya, tetapi nota bukan pesanan — tidak ada layanan yang dapat ditambahkan kepadanya. (f) *Tidak keduanya secara langsung*; paling jauh seorang pelanggan memegang daftar pesanan masa lalunya. (g) *Memegang*: usaha itu memiliki mesin-mesin. Dalam setiap kasus "memegang", kalimat is-a terdengar janggal bila diucapkan keras-keras, dan itulah uji tercepatnya.

Penyelesaian 7.2. Pohonnya berakar 42; anak kiri 42 adalah 7; anak kiri 7 adalah 3 dan anak kanannya 19; anak kanan 19 adalah 25 (42 tidak punya anak kanan). Pencarian-pencarian dan simpul-simpul yang dikunjunginya adalah: `search(19)`: 42, 7, 19 (3); `search(8)`: 42, 7, 19, lalu anak kiri 19 yang kosong (3); `findOneOf({8,9,25})`: 8 mengunjungi 42, 7, 19 (3), 9 mengunjungi 42, 7, 19 (3), 25 mengunjungi 42, 7, 19, 25 dan ditemukan (4); `findOneOf({1,2})`: 1 mengunjungi 42, 7, 3 (3), 2 mengunjungi 42, 7, 3 (3). Totalnya $3+3+3+3+4+3+3=22$. Untuk `LinearIntDB` dengan kunci tersimpan dalam urutan 3, 7, 19, 25, 42: `search(19)` berbiaya 3, `search(8)` berbiaya 5, lalu 8 dan 9 masing-masing 5 dan 25 berbiaya 4, lalu 1 dan 2 masing-masing 5, totalnya $3+5+5+5+4+5+5=32$. Ubahlah larik di `main` menjadi `{3, 7, 19, 25, 42}` untuk memastikannya. (Perhatikan bahwa pohon yang dibangun dari urutan terurut ini berupa rantai — setiap simpul hanya punya anak kanan — sehingga pohon kehilangan keunggulannya; Bab 9 dan buku-buku struktur data menjelaskan pohon yang seimbang.)

Penyelesaian 7.3. Menjalankan kode itu mencetak:

```
LaundryBag.weigh()
Bag.weigh(double)
Bag.weigh(double)
LaundryBag.weigh(int)
Bag.weigh(double)
```

(1)`weigh()` di-*override*, sehingga tipe dinamis `LaundryBag` yang menentukan: *overriding*. (2) Resolusi *overload* memakai tipe statis `Bag`, yang memiliki `weigh()` dan `weigh(double)`; `2.5` cocok dengan `weigh(double)`, yang tidak di-*override* `LaundryBag`. (3) Kejutannya: tipe statisnya tetap `Bag`, yang tidak tahu apa-apa tentang `weigh(int)`; kompilator memperlebar `2500` menjadi `2500.0` dan memilih `weigh(double)`. `LaundryBag.weigh(int)` adalah *overload* yang ditambahkan subkelas, bukan *override*, sehingga *dispatch* tidak pernah mempertimbangkannya. (4) Dengan tipe statis `LaundryBag`, `weigh(double)` (warisan) dan `weigh(int)` sama-sama menjadi kandidat, dan `int` cocok persis dengan `2500`. (5) `2.5` adalah

`double`, sehingga `Bag.weigh(double)` warisan yang berjalan. Pelajarannya: *overload* dipilih kompilator dari tipe statis; hanya *override* yang di-*dispatch*.

Penyelesaian 7.4. **Kode 7.30** dan **Kode 7.31** mengimplementasikan hierarki itu. `TaughtStudent` memegang jumlah SKS dan meng-*override* `kind()` untuk menambahkannya, dengan memanggil `super.kind()`; `ResearchStudent` memperluas `Student` secara langsung karena tidak punya SKS.

Kode 7.30 `StudentFees.java`: hierarki mahasiswa pada **Gambar 7.5** dengan biaya polimorfis.

```
import java.util.Locale;

public class StudentFees {
    public static void main(String[] args) {
        Student[] students = {
            new UndergraduateStudent("Ayu", 20),
            new PostgraduateStudent("Bagus", 12),
            new OccasionalStudent("Citra", 6),
            new ResearchStudent("Dewi")
        };
        long total = 0;
        for (Student s : students) {
            System.out.println(s.line());
            total += s.fee();
        }
        System.out.println(String.format(Locale.US, "%-40s Rp%,d", "Total", total));
    }
}

abstract class Student {
    private final String name;

    protected Student(String name) {
        this.name = name;
    }

    String getName() {
        return name;
    }

    abstract long fee();  // biaya kuliah satu semester, dalam rupiah

    String kind() {
        return getClass().getSimpleName();
    }

    String line() {
        return String.format(Locale.US, "%-6s %-33s Rp%,d", name, kind(), fee());
    }
}

abstract class TaughtStudent extends Student {
    private final int credits;
```

Kode 7.30 (Lanjutan) *StudentFees.java: hierarki mahasiswa pada Gambar 7.5 dengan biaya polimorfis.*

```
    protected TaughtStudent(String name, int credits) {
        super(name);
        this.credits = credits;
    }

    int getCredits() {
        return credits;
    }

    @Override
    String kind() {
        return super.kind() + " (" + credits + " credits)";
    }
}

class UndergraduateStudent extends TaughtStudent {
    UndergraduateStudent(String name, int credits) {
        super(name, credits);
    }

    @Override
    long fee() {
        return getCredits() * 300_000L;
    }
}

class PostgraduateStudent extends TaughtStudent {
    PostgraduateStudent(String name, int credits) {
        super(name, credits);
    }

    @Override
    long fee() {
        return getCredits() * 500_000L;
    }
}

class OccasionalStudent extends TaughtStudent {
    OccasionalStudent(String name, int credits) {
        super(name, credits);
    }
```

Kode 7.30 (Lanjutan) *StudentFees.java: hierarki mahasiswa pada Gambar 7.5 dengan biaya polimorfis.*

```
    @Override
    long fee() {
        return 250_000L + getCredits() * 400_000L;  // pendaftaran ditambah SKS
    }
}

class ResearchStudent extends Student {
    ResearchStudent(String name) {
        super(name);
    }

    @Override
    long fee() {
        return 9_000_000L;  // biaya tetap: tanpa SKS perkuliahan
    }
}
```

```
Ayu    UndergraduateStudent (20 credits) Rp6,000,000
Bagus  PostgraduateStudent (12 credits)  Rp6,000,000
Citra  OccasionalStudent (6 credits)     Rp2,650,000
Dewi   ResearchStudent                   Rp9,000,000
Total                                    Rp23,650,000
```

Kode 7.31 `student_fees.py`: hierarki mahasiswa dalam Python.

```python
"""student_fees.py: hierarki mahasiswa dengan biaya polimorfis."""
from abc import ABC, abstractmethod


class Student(ABC):
    def __init__(self, name):
        self._name = name

    @abstractmethod
    def fee(self): ...  # biaya kuliah satu semester, dalam rupiah

    def kind(self):
        return type(self).__name__

    def line(self):
        return f"{self._name:<6} {self.kind():<33} Rp{self.fee():,}"


class TaughtStudent(Student, ABC):
    def __init__(self, name, credits):
        super().__init__(name)
        self._credits = credits

    def kind(self):
        return super().kind() + f" ({self._credits} credits)"


class UndergraduateStudent(TaughtStudent):
    def fee(self):
        return self._credits * 300_000


class PostgraduateStudent(TaughtStudent):
    def fee(self):
        return self._credits * 500_000


class OccasionalStudent(TaughtStudent):
    def fee(self):
        return 250_000 + self._credits * 400_000  # pendaftaran ditambah SKS


class ResearchStudent(Student):
    def fee(self):
        return 9_000_000  # biaya tetap: tanpa SKS perkuliahan
```

Kode 7.31 (Lanjutan) *student_fees.py: hierarki mahasiswa dalam Python.*

```
def main():
    students = [UndergraduateStudent("Ayu", 20), PostgraduateStudent("Bagus", 12),
                OccasionalStudent("Citra", 6), ResearchStudent("Dewi")]
    for s in students:
        print(s.line())
    print(f"{'Total':<40} Rp{sum(s.fee() for s in students):,}")


if __name__ == "__main__":
    main()
```

```
Ayu    UndergraduateStudent (20 credits) Rp6,000,000
Bagus  PostgraduateStudent (12 credits)  Rp6,000,000
Citra  OccasionalStudent (6 credits)     Rp2,650,000
Dewi   ResearchStudent                   Rp9,000,000
Total                                    Rp23,650,000
```

Periksa: $20 \times 300{,}000 = 6{,}000{,}000$; $12 \times 500{,}000 = 6{,}000{,}000$; $250{,}000 + 6 \times 400{,}000 = 2{,}650{,}000$; totalnya $6 + 6 + 2.65 + 9 = 23.65$ juta rupiah. `getClass().getSimpleName()` dan `type(self).__name__` mengembalikan nama kelas dinamis, sehingga `kind()` tidak perlu di-*override* di kelas-kelas konkret.

Penyelesaian 7.5. (a) Tanpa deklarasi ulang hanya ada satu field `label`, sehingga `m.label` mencetak `machine` dan `describe()` mencetak `W-01 is a machine` — persis seperti sebelumnya, karena tipe statis `Machine` memang sudah memilih field milik `Machine`. Menghapus field tersembunyi tidak mengubah keluaran, tetapi menghapus jebakannya. (b) Metode private tidak di-*dispatch*, tetapi di sini memang tidak ada yang perlu di-*dispatch*: `Washer` tidak meng-*override* `describe`. `m.info()` memanggil `Machine.describe()` dan keluarannya tidak berubah; percobaan ini hanya membuktikan bahwa metode private dapat dipanggil dari dalam kelasnya sendiri. (c) Sebagai metode instans, `kind()` menjadi *override*, dan `m.kind()` kini mencetak `Washer.kind()`: tipe dinamis yang menentukan. Maka empat baris pertama berbunyi `W-01: fill water, spin at 1200 rpm`, `m.label  = machine`, `m.kind() = Washer.kind()`, `m.describe() = W-01 is a machine`.

Penyelesaian 7.6. Pertama bagian-bagiannya: $L[A] = [A, o]$ (dengan o untuk `object`); $L[B] = [B, A, o]$; $L[C] = [C, A, o]$; $L[E] = [E, C, A, o]$. Untuk $D(B, C)$: gabungkan $[B, A, o]$, $[C, A, o]$, $[B, C]$: ambil B; A ada di ekor $[C, A, o]$, jadi ambil C; lalu A, o. Jadi $L[D] = [D, B, C, A, o]$. Untuk $F(D, E)$: gabungkan $[D, B, C, A, o]$, $[E, C, A, o]$, $[D, E]$:

1. D: tidak ada di ekor mana pun; ambil. Daftar-daftarnya: $[B, C, A, o]$, $[E, C, A, o]$, $[E]$.
2. B: tidak ada di ekor mana pun; ambil. Daftar-daftarnya: $[C, A, o]$, $[E, C, A, o]$, $[E]$.
3. C: ada di ekor $[E, C, A, o]$; lewati. Kepala berikutnya E: tidak ada di ekor mana pun; ambil. Daftar-daftarnya: $[C, A, o]$, $[C, A, o]$, $[\,]$.
4. C, lalu A, lalu o.

$L[F] = [F, D, B, E, C, A, o]$. Python sepakat:

```
['F', 'D', 'B', 'E', 'C', 'A', 'object']
TypeError: Cannot create a consistent method resolution
order (MRO) for bases A, B
```

Untuk `G(A, B)` penggabungannya atas $[A, o]$, $[B, A, o]$, $[A, B]$. Kepala A ada di ekor $[B, A, o]$; kepala berikutnya, B, ada di ekor $[A, B]$. Tidak ada kepala yang dapat diterima, sehingga tidak ada urutan

yang dapat menempatkan `A` sebelum `B` (seperti diminta) sekaligus `B` sebelum `A` (sebagaimana seharusnya subkelas). Karena itulah muncul `TypeError`.

Penyelesaian 7.7. (a) Lolos kompilasi: `Beta` berada di paket yang sama, dan `protected` mencakup akses paket. (b) Lolos kompilasi: subkelas boleh memakai anggota protected milik objeknya sendiri. (c) Tidak lolos kompilasi: di luar paket `one`, `AlphaSub` boleh memakai `touch()` hanya melalui referensi bertipe `AlphaSub`, dan `new Alpha()` bukan referensi semacam itu. (d) Lolos kompilasi dengan alasan yang sama, karena `new AlphaSub()` bertipe `AlphaSub`; field statis `counter` dapat diakses dari `AlphaSub` sebagai `counter` maupun `Alpha.counter`. (e) Tidak lolos kompilasi: `Gamma` tidak berada di paket `one` dan bukan subkelas. Untuk memeriksanya, tambahkan `touch()` dan `counter` ke `Alpha`; letakkan (a) di metode statis baru di `Beta`; letakkan (b), (c), (d), `counter++`, dan `Alpha.counter++` di metode baru di akhir `AlphaSub`; dan letakkan (e) sebagai pernyataan terakhir `Gamma.main`. `javac` lalu melaporkan tepat dua galat:

```
two/AlphaSub.java:13: error: touch() has protected access in Alpha
        new Alpha().touch();         // (c)
                    ^
two/Gamma.java:13: error: touch() has protected access in Alpha
        new AlphaSub().touch();          // (e)
                       ^
2 errors
```

Penyelesaian 7.8. Masalah-masalahnya adalah yang diuraikan di Subbab 7.10: setiap `TreeDB` membawa larik dan pencacah yang tidak terpakai; `TreeDB` dapat diberikan di mana pun `LinearDB` diharapkan, padahal ia bukan `LinearDB`; dan `TreeDB` bergantung pada implementasi `LinearDB`, sehingga perubahan di sana (misalnya membuat `addSeveral` berhenti memanggil `addKey`) diam-diam mengubah `TreeDB`. *Rancangan 1* adalah **Kode 7.1**: `DB` abstrak dengan dua metode bersama dan dua metode abstrak. *Rancangan 2*: antarmuka `KeyStore` yang mendeklarasikan `addKey` dan `search`; kelas `ArrayKeyStore` dan `TreeKeyStore` yang mengimplementasikannya; dan kelas `KeyBatch` yang memegang sebuah `KeyStore` dan mengimplementasikan `addSeveral` serta `findOneOf` dengan memanggilnya. Untuk implementasi ketiga yang direncanakan, keduanya dapat dipakai: `HashDB extends DB` pada rancangan 1, atau `HashKeyStore implements KeyStore` pada rancangan 2. Rancangan 1 lebih sederhana bila setiap penyimpan memerlukan operasi borongan; rancangan 2 lebih luwes bila penyimpan juga harus memperluas kelas lain, karena Java hanya mengizinkan satu superkelas tetapi banyak antarmuka.

Penyelesaian 7.9. **Kode 7.32** menambahkan dua kelas dan tidak mengubah apa pun yang sudah ada. `ShoeCleaningService` memperluas `Service`; `PromoService` memperluas `Service` dan membungkus sebuah `Service`, persis seperti `ExpressService`. Kompilasi bersama mesin harga, `javac LaundryPricing.java ShoePromoDemo.java`, lalu jalankan `java ShoePromoDemo`.

Kode 7.32 `ShoePromoDemo.java`: cuci sepatu dan promosi ditambahkan ke mesin harga tanpa mengubahnya.

```
// Kompilasi bersama LaundryPricing.java:  javac LaundryPricing.java ShoePromoDemo.java
public class ShoePromoDemo {
    public static void main(String[] args) {
        Order order = new Order("ORD-0713", "Budi Santoso", false);
        order.add(new WashService(3.4));
        order.add(new ShoeCleaningService(2));
        order.add(new PromoService(new ShoeCleaningService(1), 20));
        order.add(new ExpressService(new PromoService(new WashService(5.0), 20)));
        order.printReceipt();
    }
}

class ShoeCleaningService extends Service {
    private final int pairs;

    ShoeCleaningService(int pairs) {
        super("SH");
        if (pairs <= 0) {
            throw new IllegalArgumentException("pairs must be positive");
        }
        this.pairs = pairs;
    }

    @Override
    String getName() {
        return "Shoe Cleaning " + pairs + " pair" + (pairs == 1 ? "" : "s");
    }

    @Override
    long price() {
        return pairs * 25000L;
    }

    @Override
    int readyInHours() {
        return 72;
    }
}

// Promosi membungkus layanan apa pun, persis seperti ExpressService.
class PromoService extends Service {
    private final Service base;
    private final int percentOff;
```

Kode 7.32 (Lanjutan) *ShoePromoDemo.java: cuci sepatu dan promosi ditambahkan ke mesin harga tanpa mengubahnya.*

```
    PromoService(Service base, int percentOff) {
        super("P-" + base.getCode());
        if (percentOff <= 0 || percentOff >= 100) {
            throw new IllegalArgumentException("percentOff must be between 1 and 99");
        }
        this.base = base;
        this.percentOff = percentOff;
    }

    @Override
    String getName() {
        return base.getName() + " -" + percentOff + "%";
    }

    @Override
    long price() {
        return base.price() - base.price() * percentOff / 100;
    }

    @Override
    int readyInHours() {
        return base.readyInHours();
    }
}
```

```
PROFITINA LAUNDRY  order ORD-0713  customer Budi Santoso
WF      Wash & Fold 3.4 kg                Rp24,500   48 h
SH      Shoe Cleaning 2 pairs             Rp50,000   72 h
P-SH    Shoe Cleaning 1 pair -20%         Rp20,000   72 h
EX-P-WF Express Wash & Fold 5.0 kg -20%   Rp42,000   12 h
Subtotal                                 Rp136,500
TOTAL                                    Rp136,500
Everything ready in 72 h
```

Kode 7.33 melakukan hal yang sama dalam Python dengan mengimpor kelas-kelas dari `laundry_pricing.py`, yang harus berada di folder yang sama.

Kode 7.33 `shoe_promo_demo.py`: perluasan yang sama dalam Python.

```
"""shoe_promo_demo.py: memperluas laundry_pricing.py tanpa menyuntingnya."""
from laundry_pricing import ExpressService, Order, Service, WashService


class ShoeCleaningService(Service):
    def __init__(self, pairs):
        super().__init__("SH")
        if pairs <= 0:
            raise ValueError("pairs must be positive")
        self._pairs = pairs

    def name(self):
        return f"Shoe Cleaning {self._pairs} pair" + ("" if self._pairs == 1 else "s")

    def price(self):
        return self._pairs * 25000

    def ready_in_hours(self):
        return 72


# Promosi membungkus layanan apa pun, persis seperti ExpressService.
class PromoService(Service):
    def __init__(self, base, percent_off):
        super().__init__("P-" + base.code)
        if not 0 < percent_off < 100:
            raise ValueError("percent_off must be between 1 and 99")
        self._base = base
        self._percent_off = percent_off

    def name(self):
        return f"{self._base.name()} -{self._percent_off}%"

    def price(self):
        return self._base.price() - self._base.price() * self._percent_off // 100

    def ready_in_hours(self):
        return self._base.ready_in_hours()
```

Kode 7.33 (Lanjutan) *shoe_promo_demo.py: perluasan yang sama dalam Python.*

```
def main():
    order = Order("ORD-0713", "Budi Santoso", False)
    order.add(WashService(3.4))
    order.add(ShoeCleaningService(2))
    order.add(PromoService(ShoeCleaningService(1), 20))
    order.add(ExpressService(PromoService(WashService(5.0), 20)))
    order.print_receipt()


if __name__ == "__main__":
    main()
```

```
PROFITINA LAUNDRY  order ORD-0713  customer Budi Santoso
WF      Wash & Fold 3.4 kg                Rp24,500   48 h
SH      Shoe Cleaning 2 pairs             Rp50,000   72 h
P-SH    Shoe Cleaning 1 pair -20%         Rp20,000   72 h
EX-P-WF Express Wash & Fold 5.0 kg -20%   Rp42,000   12 h
Subtotal                                 Rp136,500
TOTAL                                    Rp136,500
Everything ready in 72 h
```

Periksa: 3.4 kg dibulatkan ke atas menjadi 3.5 kg, $3.5 \times 7000 = 24{,}500$; dua pasang 50,000; sepasang dengan potongan 20% adalah $25{,}000 - 5{,}000 = 20{,}000$; 5.0 kg Wash & Fold adalah

35,000, potongan 20% menghasilkan 28,000, dan layanan kilat menghasilkan $\lfloor 3 \times 28{,}000/2 \rfloor = 42{,}000$, selesai dalam $\max(6, 48/4) = 12$ jam. Subtotal 136,500; tanpa potongan anggota. Karena $0.8 \times 1.5 = 1.5 \times 0.8$, promosi-di-dalam-kilat dan kilat-di-dalam-promosi berharga sama sampai pembulatan ke bawah pada setiap langkah, paling jauh berselisih satu rupiah; untuk pesanan ini keduanya menghasilkan Rp42,000. Kilat dari kilat sah dan menerapkan Persamaan (7.9) dua kali: untuk 4.2 kg Wash & Fold, $31{,}500 \to 47{,}250 \to 70{,}875$, selesai dalam $\max(6, 12/4) = 6$ jam. Bila usaha itu tidak menginginkannya, konstruktor `ExpressService` sebaiknya menolak `base` yang sudah berupa `ExpressService`.

Penyelesaian 7.10. Keluarannya:

```
['_Parent__secret', '_hint', '_Child__secret']
```

Di dalam `Parent.__init__`, `self.__secret` diubah namanya menjadi `self._Parent__secret`; di dalam `Child.__init__`, teks yang sama diubah menjadi `self._Child__secret`. Keduanya atribut yang berbeda, sehingga penugasan oleh anak menciptakan atribut baru dan nilai 1 milik induk tidak tersentuh. Untuk itulah persisnya `__name` ada: subkelas tidak dapat menimpa isi dalam induknya dengan memilih nama yang sama. `_hint` tidak diubah namanya dan dipakai bersama.

Penyelesaian 7.11. Pemanggil `WashService` dapat mengandalkan sifat-sifat seperti "harganya sama dengan $7000 \cdot \widehat{w}$" (Persamaan (7.8)) dan "`chargeableKg()` paling sedikit 3". Kode yang memperkirakan muatan mesin dengan menjumlahkan `chargeableKg()` atas pesanan hari itu akan menghitung setiap kantong bertarif tetap sebagai tak berbobot; kode yang mengaudit harga terhadap Persamaan (7.8) akan menandai setiap kantong bertarif tetap. Sifat ϕ berlaku untuk `WashService` dan gagal untuk `FlatRateWash`, sehingga menurut Persamaan (7.4) `FlatRateWash` bukan subtipe yang sah dari `WashService`. Rancangan yang lebih baik menjadikan `FlatRateWash` saudara, yaitu subkelas langsung `Service` dengan `price()` sendiri yang mengembalikan 50,000 dan `getName()` sendiri, tanpa menjanjikan apa pun tentang berat.

Penyelesaian 7.12. Sketsa penyelesaian dalam Python:

```
class Customer:
    def __init__(self, name, phone):
        self._phone = phone
        self.name = name            # melewati setter

    @property
    def phone(self):
        return self._phone          # tanpa setter: hanya-baca

    @property
    def name(self):
        return self._name

    @name.setter
    def name(self, value):
        value = value.strip()
        if not value:
            raise ValueError("name must not be empty")
        self._name = value

    def __eq__(self, other):
        if not isinstance(other, Customer):
            return NotImplemented
        return self._phone == other._phone

    def __hash__(self):
        return hash(self._phone)
```

Nomor telepon harus hanya-baca karena ia menentukan hash. Himpunan menyimpan sebuah objek di keranjang (*bucket*) yang dipilih oleh hash-nya *pada saat objek itu ditambahkan*. Bila nomor telepon dapat berubah sesudahnya, objek itu berada di keranjang yang salah: himpunan tidak lagi dapat menemukannya, dan bahkan dapat menerima pelanggan kedua yang "sama". Nama tidak ikut menentukan kesamaan maupun hash, sehingga aman untuk diubah.

Penyelesaian 7.13. **Kode 7.34** dan **Kode 7.35** adalah kedua programnya.

Kode 7.34 `ConstructorTrap.java`: konstruktor superkelas yang memanggil metode yang dapat di-override.

```java
public class ConstructorTrap {
    public static void main(String[] args) {
        new Towel("white");
    }
}

class Item {
    Item() {
        System.out.println("Item(): " + describe());
    }

    String describe() {
        return "an item";
    }
}

class Towel extends Item {
    private final String color;

    Towel(String color) {
        super();
        this.color = color;
        System.out.println("Towel(): " + describe());
    }

    @Override
    String describe() {
        return "a " + color + " towel";
    }
}
```

```
Item(): a null towel
Towel(): a white towel
```

Kode 7.35 `constructor_trap.py`: jebakan yang sama dalam Python.

```python
"""constructor_trap.py: konstruktor yang memanggil metode yang dapat di-override."""


class Item:
    def __init__(self):
        print("Item():", self.describe())

    def describe(self):
        return "an item"


class Towel(Item):
    def __init__(self, color):
        super().__init__()
        self.color = color
        print("Towel():", self.describe())

    def describe(self):
        return f"a {self.color} towel"


def main():
    try:
        Towel("white")
    except AttributeError as e:
        print("AttributeError:", e)


if __name__ == "__main__":
    main()
```

```
AttributeError: 'Towel' object has no attribute 'color'
```

Mengikuti **Gambar 7.2**: `new Towel("white")` mula-mula menjalankan `Item()`. Panggilannya ke `describe()` di-*dispatch* berdasarkan tipe dinamis objek, yang sudah berupa `Towel`, sehingga `Towel.describe()` yang berjalan — tetapi badan konstruktor `Towel` belum mengisi `color`, yang masih bernilai bawaan Java, `null`. Karena itulah tercetak `a null towel`. Baru setelah `Item()` kembali, `Towel()` mengisi warnanya, dan panggilannya sendiri mencetak `a white towel`. Python berperilaku sama, kecuali bahwa atribut yang belum diberi nilai sama sekali belum ada, sehingga panggilan dini itu gagal dengan `AttributeError`. Di kedua bahasa perbaikannya sama: konstruktor tidak semestinya memanggil metode yang dapat di-*override* oleh subkelas [6].

Daftar Pustaka

[1] O.-J. Dahl and K. Nygaard, “SIMULA: an ALGOL-based simulation language,” *Communications of the ACM*, vol. 9, no. 9, pp. 671–678, 1966.

[2] Python Software Foundation, *The Python Language Reference*, version 3.14, and *The Python Standard Library*, module `abc`. [Online]. Available: https://docs.python.org/3/

[3] J. Gosling, B. Joy, G. Steele, G. Bracha, A. Buckley, D. Smith, and G. Bierman, *The Java Language Specification, Java SE 21 Edition*. Oracle America, 2023. [Online]. Available: https://docs.oracle.com/javase/specs/jls/se21/html/

[4] OpenJDK, “JEP 513: Flexible constructor bodies,” final in JDK 25, 2025. [Online]. Available: https://openjdk.org/jeps/513

[5] T. Lindholm, F. Yellin, G. Bracha, A. Buckley, and D. Smith, *The Java Virtual Machine Specification, Java SE 21 Edition*. Oracle America, 2023. [Online]. Available: https://docs.oracle.com/javase/specs/jvms/se21/html/

[6] J. Bloch, *Effective Java*, 3rd ed. Boston, MA, USA: Addison-Wesley, 2018.

[7] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Reading, MA, USA: Addison-Wesley, 1994.

[8] A. Snyder, "Encapsulation and inheritance in object-oriented programming languages," in *Proc. OOPSLA '86*, Portland, OR, USA, 1986, pp. 38–45.

[9] B. H. Liskov and J. M. Wing, "A behavioral notion of subtyping," *ACM Transactions on Programming Languages and Systems*, vol. 16, no. 6, pp. 1811–1841, 1994.

[10] L. Ramalho, *Fluent Python*, 2nd ed. Sebastopol, CA, USA: O'Reilly Media, 2022.

[11] K. Barrett, B. Cassels, P. Haahr, D. A. Moon, K. Playford, and P. T. Withington, "A monotonic superclass linearization for Dylan," in *Proc. OOPSLA '96*, San Jose, CA, USA, 1996, pp. 69–82.

[12] M. Simionato, "The Python 2.3 method resolution order," Python Software Foundation, 2003. [Online]. Available: https://docs.python.org/3/howto/mro.html

[13] Oracle, "Controlling access to members of a class," *The Java Tutorials*. [Online]. Available: https://docs.oracle.com/javase/tutorial/java/javaOO/accesscontrol.html

[14] G. van Rossum, B. Warsaw, and A. Coghlan, "PEP 8 – Style guide for Python code," Python Enhancement Proposal 8, 2001. [Online]. Available: https://peps.python.org/pep-0008/

[15] D. L. Parnas, "On the criteria to be used in decomposing systems into modules," *Communications of the ACM*, vol. 15, no. 12, pp. 1053–1058, 1972.

[16] B. Meyer, *Object-Oriented Software Construction*, 2nd ed. Upper Saddle River, NJ, USA: Prentice Hall, 1997.

Bab 8

Evaluasi dan Pembahasan II: Struktur Rekursif, Kelas yang Tangguh, dan Hierarki Kelas

Bab ini tidak menambahkan fitur bahasa yang baru. Bab ini mengerjakan soal: dua puluh satu soal, masing-masing diselesaikan dari halaman kosong, dalam Java dan dalam Python. Lima belas soal berasal dari satu rangkaian latihan yang saling terhubung tentang dua struktur rekursif, yaitu pohon biner dan *list* imutabel. Enam soal lainnya baru, dan masing-masing membidik satu gagasan dari Bab 5 sampai Bab 7: berkas pesanan yang dibaca tanpa program terhenti karena baris yang rusak, larik yang tumbuh sendiri dan mendokumentasikan kontraknya sendiri, antarmuka dengan dua implementasi yang lolos dari uji yang sama, hierarki kelas yang pemanggilan metodenya Anda telusuri dengan tangan, dua kelas permainan yang kodenya berulang lalu di-*refactor* menjadi satu hierarki, serta sebuah kelas dengan empat *bug* enkapsulasi yang harus ditemukan dan diperbaiki. Setiap soal menempuh jalan yang sama. Kita menyatakan ulang soalnya dengan kata-kata sederhana, merancangnya dengan tabel atau gambar UML, menulis penyelesaian Java dan penyelesaian Python, menjalankan uji dan memperlihatkan apa yang benar-benar dicetaknya, menjelaskan kode baris demi baris, mempelajari penyelesaian keliru yang menggoda dan melihat persis bagaimana ia gagal, lalu menutup dengan varian *uji diri* beserta jawabannya. Di akhir bab, Anda akan memiliki sebuah metode untuk soal apa pun yang melibatkan beberapa kelas yang bekerja sama, dan Anda sudah memakainya dua puluh satu kali.

8.1 Titik Tolak Kita

Bab 5, 6, dan 7 telah mengisi kotak perkakas yang dipakai oleh setiap bab sesudahnya. Sebelum membukanya, mari kita daftarkan isinya, sebab menyebut perkakas yang tepat sudah separuh dari menyelesaikan soal.

Dari **Bab 5** kita memiliki *larik* (*array*: sejumlah sel yang tetap banyaknya, bertipe sama, bernomor mulai dari 0), `ArrayList` yang dapat tumbuh di Java dan `list` di Python, pembacaan masukan, *eksepsi* (mesin `try`/`catch`/`finally` di Java dan `try`/`except`/`finally` di Python, eksepsi *checked* dan *unchecked*, serta eksepsi yang kita definisikan sendiri), dan *berkas* (membuka, membaca baris demi baris, dan menutupnya dengan aman memakai `try`-with-resources atau `with`).

Dari **Bab 6** kita memiliki *dokumentasi* (komentar Javadoc dengan `@param`, `@return`, dan `@throws`, serta *docstring* Python), *pengujian* (JUnit 6 di Java dan `unittest` di Python: sebuah uji menyiapkan suatu keadaan, menjalankan kode, lalu menyatakan hasil yang seharusnya), dan *antarmuka* (*interface*: janji bernama berupa sekumpulan metode tanpa *field* sendiri, yang dapat diimplementasikan oleh kelas mana pun), bersama dengan *overloading* (beberapa metode dengan satu nama tetapi daftar parameter yang berbeda).

Dari **Bab 7** kita memiliki *pewarisan* (*inheritance*: subkelas memakai ulang dan memperluas superkelas), *kelas abstrak*, *overriding* (subkelas mengganti metode yang diwarisinya), *polimorfisme* (satu pemanggilan, `s.describe()`, menjalankan kode yang berbeda bergantung

pada objek di balik `s`), `super`, `final`, dan keempat *tingkat akses* Java (`private`, *package-private*, `protected`, `public`) beserta konvensi penamaan Python sebagai padanannya.

Bab ini memakai semuanya. Bab ini disusun dalam empat lapis.

1. **Sebuah metode** (Subbab 8.2). Delapan langkah yang membawa Anda dari satu paragraf kebutuhan menjadi sekumpulan kelas, uji, dan kode.
2. **Dua struktur kecil** (Subbab 8.3): kelas `Tree` yang bekerja melalui *delegasi*, yakni meminta subpohonnya mengerjakan sebagian dari setiap tugas, dan `List` imutabel yang dibangun dari dua operasi, `nil` dan `cons`. Lima belas soal memakainya.
3. **Dua puluh satu soal yang dikerjakan.** Soal 8.1 sampai 8.7 (Subbab 8.4) tentang pohon, Soal 8.8 sampai 8.15 (Subbab 8.5) tentang *list*, dan Soal 8.16 sampai 8.21 (Subbab 8.6 sampai 8.8) adalah soal-soal baru tentang eksepsi, berkas, larik, dokumentasi, pengujian, antarmuka, pewarisan, polimorfisme, dan kontrol akses.
4. **Subbab penutup seperti biasa**: perbandingan semua soal, tempat gagasan-gagasan ini bertemu produk nyata, kesalahan yang sering terjadi, ringkasan, soal latihan, dan penyelesaian lengkapnya.

Setiap soal disajikan dalam urutan yang sama, dan setiap bagiannya diberi label tebal agar mudah Anda temukan kembali: **Soal** (pernyataannya), **Dengan bahasa kita sendiri**, **Rancangan**, **Penyelesaian Java**, **Penyelesaian Python**, **Baris demi baris**, **Penyelesaian keliru yang menggoda**, dan **Uji diri**. Uji-ujinya tertanam di dalam program, sehingga keluaran yang menyertai setiap kode *adalah* laporan ujinya.

> **CARA MEMAKAI BAB INI.** Bacalah pernyataan sebuah soal, lalu tutup sisanya dengan selembar kertas dan cobalah sendiri selama sepuluh menit: gambarlah rancangannya, tulislah kasus dasarnya, cobalah satu contoh dengan tangan. Penyelesaian yang lebih dulu Anda pergumulkan akan Anda pahami dengan cara yang tidak pernah dicapai oleh penyelesaian yang hanya Anda baca. Sesudah itu, lanjutkan membaca dan bandingkan. Varian *uji diri* memberi tahu Anda apakah gagasannya benar-benar sudah sampai.

Sepatah kata tentang uji. JUnit 6 adalah perkakas pengujian baku untuk Java (Bab 6), tetapi ia pustaka terpisah yang harus diunduh. Agar setiap program Java di bab ini dapat dijalankan hanya dengan JDK, setiap program menguji dirinya sendiri dengan sebuah metode bantu kecil, `check`, yang membandingkan nilai yang diharapkan dengan nilai sebenarnya, mencetak `PASS` beserta sebuah nama bila keduanya sama, dan melempar `AssertionError` bila berbeda. Subbab 8.7 memperlihatkan wujud pemeriksaan yang sama sebagai uji JUnit 6 dan cara menjalankannya. Program Python langsung memakai `unittest`, karena modul itu sudah tersedia bersama Python [1].

8.2 Cara Menghadapi Soal Perancangan dengan Banyak Kelas

Kebanyakan soal dalam pemrograman berorientasi objek bukanlah “tulislah satu metode ini”. Soalnya berbunyi “inilah sebuah keadaan yang berisi beberapa jenis benda; buatlah programnya”. Sebuah binatu menerima pesanan dari pelanggan, setiap pesanan memakai satu layanan, setiap layanan memiliki aturan harga, dan seorang kurir mengantar pesanan yang sudah selesai. Dari mana Anda mulai? Subbab ini memberikan sebuah metode, delapan langkah yang memberi tahu apa yang harus ditulis di kertas, dan dalam urutan apa, sebelum Anda menulis sebaris kode pun. Pada soal kecil, langkah-langkah ini memakan sepuluh menit. Pada soal besar, langkah-langkah inilah yang membedakan sebuah rancangan dari setumpuk kelas.

> **DEFINISI.** *Tanggung jawab* (*responsibility*) adalah sesuatu yang diketahui sebuah kelas (sebuah *field*) atau dikerjakannya (sebuah metode). Kelas yang baik memiliki sedikit tanggung jawab yang saling berkaitan, dan hanya kelas itu sendiri yang mengubah *field*-nya.

Langkah 1 — Garis bawahi kata benda dan lingkari kata kerja. Bacalah kebutuhan sekali, pelan-pelan, dengan pena di tangan. Kata benda (*pesanan*, *pelanggan*, *layanan*, *berat*, *harga*) adalah calon kelas dan calon *field*; kata kerja (*menimbang*, *menghargai*, *mengantar*, *mengurai*) adalah calon metode. Jangan memutuskan apa pun dulu; kumpulkan saja.

Langkah 2 — Pilah kata bendanya. Kata benda yang memiliki data sendiri *dan* perilaku sendiri menjadi kelas (`Order`). Kata benda yang hanya berupa nilai menjadi *field* (`kg` adalah `double` di dalam `Order`). Kata benda yang menamai suatu *jenis* dari sesuatu (*layanan ekspres*, *layanan standar*) adalah petunjuk adanya antarmuka atau hierarki kelas.

Langkah 3 — Beri setiap kata kerja seorang pemilik. Untuk setiap metode, tanyakan "objek mana yang memiliki data yang diperlukan metode ini?" lalu letakkan metodenya di sana. Penghitungan harga memerlukan tarif, jadi ia milik aturan harga, bukan milik pelanggan. Satu pertanyaan ini, bila dijawab dengan jujur, mencegah sebagian besar rancangan yang buruk.

Langkah 4 — Pilih hubungannya. Empat jenis hubungan mencakup hampir semua hal: *has-a* (sebuah pesanan memiliki daftar barang: sebuah *field*), *uses* (sebuah pengurai memakai sebuah pesanan: parameter atau variabel lokal), *is-a* dengan kode bersama (layanan ekspres adalah layanan cuci: pewarisan, Bab 7), dan *can-do* tanpa kode bersama (apa pun yang dapat menghargai suatu berat: antarmuka, Bab 6). Utamakan *has-a* dan *can-do*; pakailah *is-a* hanya bila subkelasnya benar-benar kasus khusus dan benar-benar berbagi kode [2].

Langkah 5 — Tetapkan tingkat akses untuk segala sesuatu. *Field* bersifat `private` (di Python, dinamai dengan garis bawah di depan). Sebuah metode `public` hanya bila ada kelas luar yang memerlukannya; `protected` bila hanya subkelas yang memerlukannya; selain itu *package-private* atau `private`. Tanyakan untuk setiap *field*: "siapa yang boleh mengubahnya, dan aturan apa yang harus tetap berlaku sesudah perubahan itu?" Jawabannya memberi tahu pemeriksaan apa yang harus dilakukan konstruktor dan *setter*.

Langkah 6 — Gambarlah. Diagram kelas UML [3] adalah satu kotak per kelas dengan tiga ruang (nama, *field*, metode), satu tanda di depan setiap anggota (`-` *private*, `#` *protected*, `~` *package-private*, `+` *public*), segitiga berongga untuk "*extends*", garis putus-putus dengan segitiga berongga untuk "*implements*", dan panah putus-putus untuk "*uses*". Untuk soal kecil, sebuah tabel dengan informasi yang sama sama baiknya; bab ini memakai keduanya.

Langkah 7 — Tulis uji sebelum kode. Untuk setiap metode, tulislah dua atau tiga pemanggilan beserta hasil yang diharapkan, dan selalu sertakan *kasus tepi* (*edge case*): *list* kosong, daun, berkas yang tidak ada, berat nol, indeks yang sama dengan ukuran. Sebagian besar penyelesaian keliru di bab ini lolos pada contoh biasa dan gagal tepat pada satu kasus tepi.

Langkah 8 — Kerjakan potongan terkecil, jalankan, ulangi. Tulislah satu metode, jalankan ujinya, lalu lanjut ke metode berikutnya. Setelah uji lolos, baca kodenya sekali lagi dengan hanya memperhatikan perulangan kode dan tingkat akses, lalu rapikan (*refactoring*, Soal 8.20).

Tabel 8.1 meringkas metode ini dalam satu halaman. Anda akan melihatnya diterapkan secara eksplisit pada setiap soal di bab ini.

Tabel 8.1 Metode delapan langkah untuk soal perancangan dengan banyak kelas. Langkah 1 sampai 6 adalah perancangan, Langkah 7 adalah spesifikasi, dan Langkah 8 adalah implementasi.

Langkah	Yang ditulis di kertas	Pertanyaan yang dijawabnya
1	Kata benda dan kata kerja dari kebutuhan	Apa saja yang ada dalam keadaan ini?
2	Kelas, *field*, dan "jenis dari"	Kata benda mana yang layak menjadi kelas?
3	Seorang pemilik untuk setiap metode	Siapa yang memiliki data yang diperlukan?
4	*has-a*, *uses*, *is-a*, *can-do*	Bagaimana kelas-kelas itu terhubung?
5	Tingkat akses untuk setiap anggota	Siapa boleh mengubah apa, dengan aturan apa?
6	Diagram UML atau tabel rancangan	Muatkah seluruh rancangan dalam satu halaman?
7	Uji, beserta kasus tepinya	Bagaimana saya tahu bahwa ini bekerja?
8	Satu metode sekali jalan, lalu rapikan	Apakah sekarang bekerja? Apakah rapi?

Untuk metode rekursif, yang mengisi Subbab 8.4 dan 8.5, Langkah 3 dan 7 mengambil bentuk khusus yang layak dinyatakan tersendiri.

> **RESEP DELEGASI UNTUK REKURSI.** Untuk menulis metode rekursif pada pohon atau *list*, jawablah tiga pertanyaan secara tertulis. (1) *Kasus dasar*: apa jawaban untuk struktur terkecil, yaitu *list* kosong atau subpohon yang tidak ada? (2) *Delegasi*: apa yang saya minta dihitungkan oleh bagian-bagian saya yang lebih kecil (*tail*, subpohon kiri dan kanan)? Anggaplah mereka mengerjakannya dengan benar. (3) *Tugas saya sendiri*: bagaimana saya menggabungkan jawaban mereka dengan data saya sendiri (*head*, akar)? Bila ketiga jawaban itu benar, metodenya benar, berdasarkan induksi pada ukuran struktur.

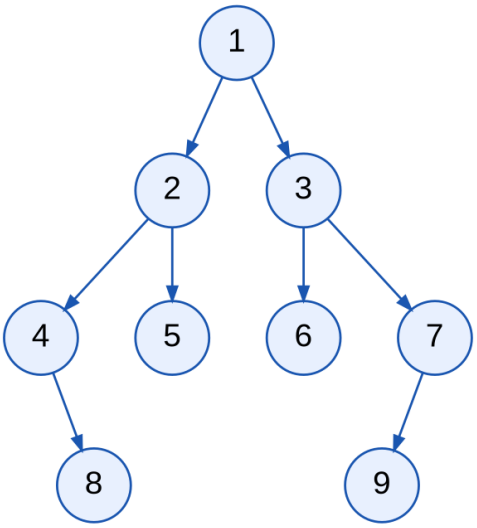
Resep ini persis penalaran induksi matematika [4]: kasus dasar adalah basis induksi, dan "anggaplah bagian-bagian yang lebih kecil mengerjakannya dengan benar" adalah hipotesis induksi. Tabel rancangan di Subbab 8.4 dan 8.5 memiliki satu kolom untuk setiap jawaban dari ketiga pertanyaan itu.

8.3 Dua Struktur dalam Bab Ini: Pohon dan *List* Cons

Lima belas soal di bawah ini memakai dua struktur data kecil. Keduanya *rekursif*: sebuah pohon memuat pohon-pohon yang lebih kecil, dan sebuah *list* memuat *list* yang lebih pendek. Keduanya juga latihan yang sangat baik untuk Bab 5 sampai 7, karena memaksa Anda memikirkan referensi (siapa menunjuk ke apa), kontrol akses (siapa boleh mengubah apa), eksepsi (apa yang terjadi bila Anda meminta elemen pertama dari *list* kosong), dan pengujian (bagaimana Anda tahu sebuah metode rekursif itu benar?).

8.3.1 Pohon biner yang bekerja melalui delegasi

Pohon biner (*binary tree*) adalah sesuatu yang tidak ada (dalam kode, `null` di Java dan `None` di Python) atau sebuah *simpul* (*node*) yang menyimpan satu nilai, disebut *akar*-nya (*root*), dan dua pohon biner, disebut *subpohon kiri* dan *subpohon kanan*. Simpul yang kedua subpohonnya tidak ada disebut *daun* (*leaf*). **Gambar 8.1** memperlihatkan pohon biner dengan sembilan simpul: akarnya 1, daunnya 5, 6, 8, dan 9, dan simpul 4 memiliki subpohon kanan tetapi tidak memiliki subpohon kiri.



Gambar 8.1 Pohon biner Soal 8.1. Simpul 4 hanya memiliki anak kanan (8) dan simpul 7 hanya memiliki anak kiri (9); setiap simpul dalam lainnya memiliki dua anak.

Kode 8.1 adalah kelas `Tree` yang menjadi titik awal ketujuh soal pohon. Kelas ini adalah kelas *bersarang* (*nested*): ia berada di dalam kelas publik `TreeStart` dan ditandai `static`, sehingga sebuah `Tree` tidak memerlukan objek `TreeStart` untuk bisa ada. Setiap program di Subbab 8.4 membawa salinannya sendiri dari kelas bersarang ini, hanya dengan metode yang diperlukan soalnya, sehingga setiap berkas dapat dikompilasi dan dijalankan secara mandiri.

Kode 8.1 `TreeStart.java`: kelas `Tree` awal dengan dua konstruktor, `mirror()` melalui delegasi, `toString()` yang ringkas, dan gambar menyamping.

```java
// TreeStart.java - kelas Tree awal: dua konstruktor,
// mirror() melalui delegasi, dan dua cara mencetak pohon.
public class TreeStart {

    /** Pohon biner yang simpulnya berlabel bilangan bulat. */
    static final class Tree {
        private int root;
        private Tree left, right;

        /** Cabang: sebuah akar dengan subpohon (yang mungkin kosong). */
        Tree(int root, Tree left, Tree right) {
            this.root = root;
            this.left = left;
            this.right = right;
        }

        /** Daun: kasus khusus sebuah cabang. */
        Tree(int root) {
            this(root, null, null);
        }

        /** Mencerminkan diri: subpohon saya mencerminkan dirinya, lalu saya menukarnya. */
        void mirror() {
            if (left != null) {
                left.mirror();          // delegasi ke subpohon kiri
            }
            if (right != null) {
                right.mirror();         // delegasi ke subpohon kanan
            }
            Tree originalLeft = left;   // tugas saya sendiri: menukar cabang
            left = right;
            right = originalLeft;
        }

        /** Bentuk ringkas: daun adalah bilangannya, cabang adalah root(left,right). */
        @Override
        public String toString() {
            if (left == null && right == null) {
                return Integer.toString(root);
            }
            String l = (left == null) ? "-" : left.toString();
            String r = (right == null) ? "-" : right.toString();
            return root + "(" + l + "," + r + ")";
        }
```

Kode 8.1 (Lanjutan) *TreeStart.java: kelas Tree awal dengan dua konstruktor, mirror() melalui delegasi, toString() yang ringkas, dan gambar menyamping.*

```
        /** Gambar menyamping: subpohon kanan di atas, 4 spasi per tingkat. */
        String toStringAdv() {
            return toStringFrom(0);
        }

        private String toStringFrom(int depth) {
            String sRight = (right == null) ? "" : right.toStringFrom(depth + 4);
            String sLeft = (left == null) ? "" : left.toStringFrom(depth + 4);
            return sRight + " ".repeat(depth) + root + "\n" + sLeft;
        }
    }

    public static void main(String[] args) {
        Tree t = new Tree(3,
                new Tree(4, new Tree(6), new Tree(7)),
                new Tree(5, new Tree(8), new Tree(9)));
        System.out.println("t          = " + t);
        System.out.print(t.toStringAdv());
        t.mirror();
        System.out.println("mirrored t = " + t);
        System.out.print(t.toStringAdv());
    }
}
```

```
t          = 3(4(6,7),5(8,9))
        9
    5
        8
3
        7
    4
        6
mirrored t = 3(5(9,8),4(7,6))
        6
    4
        7
3
        8
    5
        9
```

Ada tiga rincian yang layak dibaca pelan-pelan. Pertama, *field* `root`, `left`, dan `right` bersifat `private`: hanya kode di dalam `Tree` yang dapat membaca atau mengubahnya, itulah sebabnya setiap operasi pada pohon adalah metode milik `Tree`. Kedua, konstruktor daun `Tree(int root)` tidak mengulang ketiga penugasan; ia memanggil konstruktor cabang dengan `this(root, null, null)`, yaitu perangkaian konstruktor (*constructor chaining*) dari Bab 7. Ketiga, perhatikan `mirror()`. Metode ini tidak menjelajahi pohon dengan gelung. Ia meminta subpohon kirinya mencerminkan diri, meminta subpohon kanannya mencerminkan diri, lalu mengerjakan tugas kecilnya sendiri, yakni menukar kedua referensinya. Itulah *resep delegasi* dari Subbab 8.2: kasus dasarnya tersembunyi dalam pemeriksaan `if (left != null)` (subpohon yang tidak ada tidak perlu dicerminkan), delegasinya adalah kedua pemanggilan rekursif, dan tugas simpul itu sendiri adalah penukarannya.

Kedua metode pencetak adalah alat bantu untuk melihat hasil. `toString()` menulis daun sebagai bilangannya dan cabang sebagai `root(left,right)`, dengan `-` untuk subpohon yang tidak ada, sehingga pohon pada **Gambar 8.1** tercetak sebagai `1(2(4(-,8),5),3(6,7(9,-)))`. `toStringAdv()` menggambar pohon dalam posisi menyamping: miringkan kepala Anda ke kiri, maka subpohon kanan berada di atas. Setiap uji di Subbab 8.4 membandingkan bentuk

`toString()` dengan sebuah *string* yang diharapkan, karena dua pohon dengan *string* yang sama pasti memiliki bentuk dan nilai yang persis sama.

Kode 8.2 adalah kelas yang sama dalam Python.

Kode 8.2 `tree_start.py`: kelas `Tree` yang sama dalam Python. Garis bawah di depan menandai sebuah atribut sebagai *private* menurut konvensi.

```
# tree_start.py - kelas Tree awal: mirror() melalui delegasi,
# dan dua cara mencetak pohon.


class Tree:
    """Pohon biner yang simpulnya berlabel bilangan bulat."""

    def __init__(self, root, left=None, right=None):
        self._root = root          # garis bawah di depan: "private menurut konvensi"
        self._left = left
        self._right = right

    def mirror(self):
        """Mencerminkan diri: subpohon saya mencerminkan dirinya, lalu saya menukarnya."""
        if self._left is not None:
            self._left.mirror()    # delegasi ke subpohon kiri
        if self._right is not None:
            self._right.mirror()   # delegasi ke subpohon kanan
        self._left, self._right = self._right, self._left

    def __str__(self):
        """Bentuk ringkas: daun adalah bilangannya, cabang adalah root(left,right)."""
        if self._left is None and self._right is None:
            return str(self._root)
        l = "-" if self._left is None else str(self._left)
        r = "-" if self._right is None else str(self._right)
        return f"{self._root}({l},{r})"

    def to_string_adv(self, depth=0):
        """Gambar menyamping: subpohon kanan di atas, 4 spasi per tingkat."""
        s_right = "" if self._right is None else self._right.to_string_adv(depth + 4)
        s_left = "" if self._left is None else self._left.to_string_adv(depth + 4)
        return s_right + " " * depth + str(self._root) + "\n" + s_left


if __name__ == "__main__":
    t = Tree(3, Tree(4, Tree(6), Tree(7)), Tree(5, Tree(8), Tree(9)))
    print("t          =", t)
    print(t.to_string_adv(), end="")
    t.mirror()
    print("mirrored t =", t)
    print(t.to_string_adv(), end="")
```

```
t          = 3(4(6,7),5(8,9))
        9
    5
        8
3
        7
    4
        6
mirrored t = 3(5(9,8),4(7,6))
        6
    4
        7
3
        8
    5
        9
```

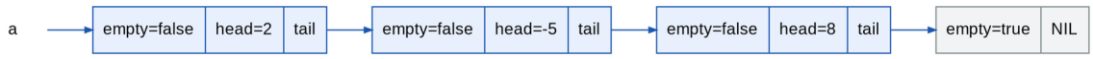
Python tidak memiliki kata kunci `private`. Garis bawah di depan pada `self._root` adalah sebuah *konvensi* yang berarti "ini rincian implementasi; jangan disentuh dari luar kelas". Tidak ada yang dapat menghalangi pemrogram yang bersikeras, tetapi setiap pemrogram Python membaca garis bawah itu sebagai pagar. Python juga tidak memiliki konstruktor yang di-*overload*; argumen bawaan `left=None, right=None` membuat satu `__init__` dapat melayani sebagai konstruktor cabang sekaligus konstruktor daun. Penukaran di `mirror()` adalah satu *penugasan tuple*, `self._left, self._right = self._right, self._left`, yang mengevaluasi seluruh ruas kanan sebelum menugaskan apa pun. Kedua program mencetak baris-baris yang persis sama.

8.3.2 *List* imutabel yang dibangun dari nil dan cons

Struktur kedua adalah *list* bilangan bulat yang didefinisikan seperti cara seorang matematikawan mendefinisikannya. Sebuah *list* bisa *kosong*, disebut `nil`, atau dibangun oleh `cons` dari sebuah elemen pertama, disebut *head*, dan sebuah *list* yang lebih pendek, disebut *tail*. Dengan demikian, *list* $[2, -5, 8]$ adalah

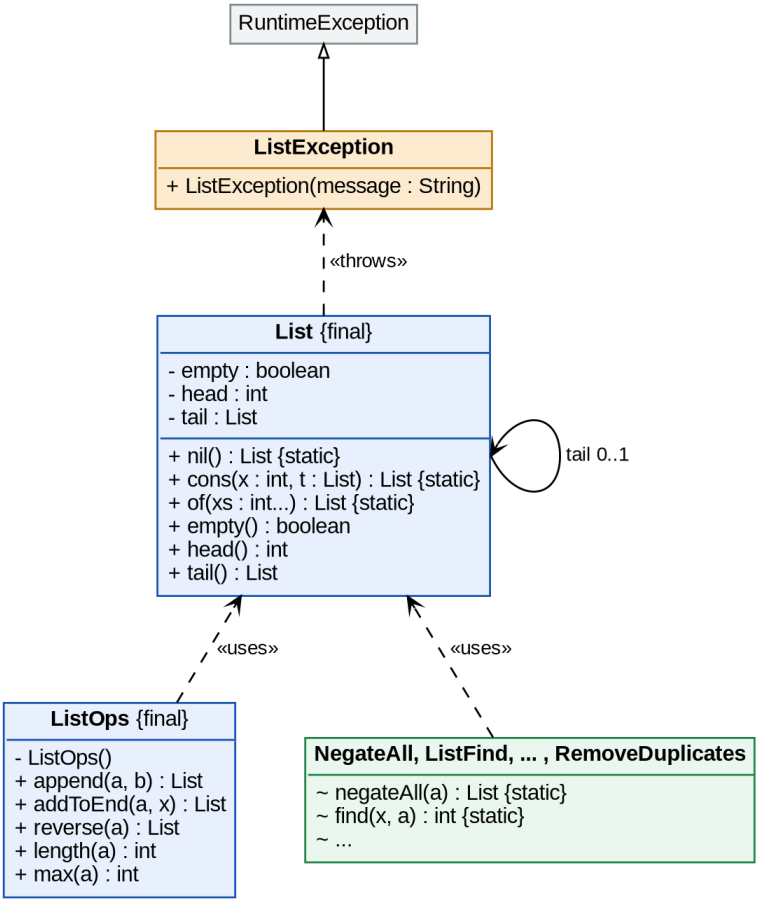
$$[2, -5, 8] = \text{cons}(2, \text{cons}(-5, \text{cons}(8, \text{nil}))) \tag{8.1}$$

dan **Gambar 8.2** menggambarkan objek-objek yang diciptakan oleh Persamaan (8.1): tiga sel tak kosong, masing-masing menyimpan sebuah *head* dan sebuah referensi ke *tail*-nya, serta satu *list* kosong bersama di ujungnya. Nama `cons` dan `nil` berasal dari keluarga bahasa Lisp, tempat definisi ini lahir [4].



Gambar 8.2 Objek-objek di balik `List a = cons(2, cons(-5, cons(8, nil())))`. Setiap sel memiliki *head* dan referensi *tail*; *tail* yang terakhir adalah satu-satunya *list* kosong yang dipakai bersama.

Versi Java dipecah menjadi tiga berkas, persis seperti yang dilakukan proyek sungguhan: `List` itu sendiri (**Kode 8.3**), eksepsi yang dilemparkannya (**Kode 8.4**), dan sebuah kelas utilitas berisi operasi bantu (**Kode 8.5**). **Gambar 8.3** memperlihatkan bagaimana ketiganya saling terkait.



Gambar 8.3 Diagram kelas UML untuk kelas-kelas *list*. `ListException` bersifat *unchecked* karena memperluas `RuntimeException`; setiap kelas soal di Subbab 8.5 hanya memakai `List`, tidak memperluasnya.

Kode 8.3 `List.java`: *list* bilangan bulat yang imutabel. Konstruktornya *private*, sehingga `nil()`, `cons()`, dan `of()` adalah satu-satunya cara membangun sebuah *list*.

```java
// List.java - list bilangan bulat yang imutabel, dibangun dari "nil" dan "cons".
public final class List {

    private final boolean empty;
    private final int head;
    private final List tail;

    private static final List NIL = new List();     // satu list kosong yang dipakai
bersama

    private List(int head, List tail) {              // list tak kosong
        this.empty = false;
        this.head = head;
        this.tail = tail;
    }

    private List() {                                 // list kosong
        this.empty = true;
        this.head = 0;
        this.tail = null;
    }

    /** List kosong, ditulis [] dalam komentar bab ini. */
    public static List nil() {
        return NIL;
    }

    /** List baru yang elemen pertamanya x dan sisanya t. */
    public static List cons(int x, List t) {
        return new List(x, t);
    }

    /** Kemudahan: List.of(2, -5, 8) sama dengan cons(2, cons(-5, cons(8, nil()))). */
    public static List of(int... xs) {
        List result = nil();
        for (int i = xs.length - 1; i >= 0; i--) {  // bangun dari belakang
            result = cons(xs[i], result);
        }
        return result;
    }

    public boolean empty() {
        return empty;
    }
```

Kode 8.3 (Lanjutan) *List.java: list bilangan bulat yang imutabel. Konstruktornya private, sehingga nil(), cons(), dan of() adalah satu-satunya cara membangun sebuah list.*

```
    public int head() {
        if (empty) {
            throw new ListException("head() of the empty list");
        }
        return head;
    }

    public List tail() {
        if (empty) {
            throw new ListException("tail() of the empty list");
        }
        return tail;
    }

    @Override
    public String toString() {
        StringBuilder sb = new StringBuilder("[");
        for (List p = this; !p.empty; p = p.tail) {
            sb.append(p.head);
            if (!p.tail.empty) {
                sb.append(", ");
            }
        }
        return sb.append("]").toString();
    }
}
```

Kode 8.4 `ListException.java`: eksepsi *unchecked* untuk operasi yang memerlukan *list* tak kosong.

```
// ListException.java - dilempar bila suatu operasi memerlukan list tak kosong.
public class ListException extends RuntimeException {

    private static final long serialVersionUID = 1L;

    public ListException(String message) {
        super(message);
    }
}
```

Kode 8.5 `ListOps.java`: operasi bantu yang ditulis secara rekursif. Konstruktor *private* mencegah siapa pun membuat objek `ListOps`.

```
// ListOps.java - operasi bantu rekursif pada List (sebuah kelas utilitas).
public final class ListOps {

    private ListOps() {                // tanpa objek: hanya metode static
    }

    /** Seluruh a, diikuti seluruh b. */
    public static List append(List a, List b) {
        if (a.empty()) {
            return b;
        }
        return List.cons(a.head(), append(a.tail(), b));
    }

    /** Salinan a dengan x ditambahkan di akhir. */
    public static List addToEnd(List a, int x) {
        return append(a, List.cons(x, List.nil()));
    }

    /** Elemen-elemen a dalam urutan terbalik. */
    public static List reverse(List a) {
        if (a.empty()) {
            return List.nil();
        }
        return addToEnd(reverse(a.tail()), a.head());
    }

    /** Banyaknya elemen a. */
    public static int length(List a) {
        return a.empty() ? 0 : 1 + length(a.tail());
    }

    /** Elemen terbesar; list kosong tidak memilikinya, jadi itu galat. */
    public static int max(List a) {
        if (a.empty()) {
            throw new ListException("max() of the empty list");
        }
        if (a.tail().empty()) {
            return a.head();
        }
        return Math.max(a.head(), max(a.tail()));
    }
}
```

Pilihan rancangan dalam ketiga berkas ini semuanya adalah keputusan dari Bab 5 sampai 7, dan layak disebut satu per satu.

- **Imutabilitas.** Ketiga *field* `List` bersifat `private final`. Sekali sebuah *list* ada, ia tidak pernah berubah; setiap operasi yang "mengubah" *list* sebenarnya membangun *list* baru. Itulah sebabnya `negateAll` di Soal 8.8 harus *mengembalikan* sebuah *list*, dan itulah sebabnya berbagi *tail* antar-*list* (seperti yang dilakukan `append` terhadap `b`) sepenuhnya aman.
- **Konstruktor *private*, *factory* publik.** Tidak ada kode di luar `List` yang dapat memanggil `new List(...)`. Kode luar harus memakai `nil()`, `cons()`, atau `of()`. *List* kosong diciptakan sekali saja, di *field* `static final` bernama `NIL`, lalu dipakai bersama; tidak ada alasan untuk memiliki dua *list* kosong.
- **Eksepsi untuk kontrak yang dilanggar.** `head()` dan `tail()` dari *list* kosong tidak memiliki jawaban yang masuk akal, sehingga keduanya melempar `ListException`. Karena memperluas `RuntimeException`, eksepsi ini *unchecked*: *compiler* tidak memaksa setiap pemanggil menulis `try`, dan itu cocok untuk galat yang tidak pernah dipicu oleh kode yang benar.
- **Kelas utilitas, bukan superkelas.** `ListOps` hanya memuat metode `static` dan memiliki konstruktor `private`. Kelas-kelas soal di Subbab 8.5 memanggil `List.cons(...)` dan

`SkipElement.skip(...)` dengan menyebut namanya. Memang menggoda untuk menulis `class NegateAll extends List` hanya agar bisa mengetik `cons(...)` tanpa awalan `List.`, tetapi itu pewarisan demi kenyamanan: ia menyatakan bahwa `NegateAll` *adalah* sebuah *list*, dan itu keliru. Bab 7 mencadangkan pewarisan untuk hubungan *is-a* yang sungguhan, dan `List` bersifat `final` sehingga pertanyaan itu bahkan tidak muncul.

Kode 8.6 mencoba kelas-kelas ini untuk pertama kalinya.

Kode 8.6 `ListStart.java`: perkenalan pertama dengan `List`, `ListOps`, dan `ListException`.

```
// ListStart.java - perkenalan pertama dengan List, ListOps dan ListException.
public class ListStart {

    public static void main(String[] args) {
        List a = List.cons(2, List.cons(-5, List.cons(8, List.nil())));
        List b = List.of(7, 1);
        System.out.println("a               = " + a);
        System.out.println("a.head()        = " + a.head());
        System.out.println("a.tail()        = " + a.tail());
        System.out.println("append(a, b)    = " + ListOps.append(a, b));
        System.out.println("addToEnd(a, 0)  = " + ListOps.addToEnd(a, 0));
        System.out.println("reverse(a)      = " + ListOps.reverse(a));
        System.out.println("length(a)       = " + ListOps.length(a));
        System.out.println("max(a)          = " + ListOps.max(a));
        System.out.println("a is unchanged  = " + a);
        try {
            ListOps.max(List.nil());
        } catch (ListException e) {
            System.out.println("caught ListException: " + e.getMessage());
        }
    }
}
```

Kompilasi keempat berkas bersama-sama lalu jalankan demonya: `javac List.java ListException.java ListOps.java ListStart.java`, kemudian `java ListStart`.

```
a               = [2, -5, 8]
a.head()        = 2
a.tail()        = [-5, 8]
append(a, b)    = [2, -5, 8, 7, 1]
addToEnd(a, 0)  = [2, -5, 8, 0]
reverse(a)      = [8, -5, 2]
length(a)       = 3
max(a)          = 8
a is unchanged  = [2, -5, 8]
caught ListException: max() of the empty list
```

Perhatikan baris `a is unchanged = [2, -5, 8]`: `append`, `addToEnd`, dan `reverse` semuanya mengembalikan *list* baru dan membiarkan `a` apa adanya. Perhatikan pula baris terakhir. `max` dari *list* kosong melempar `ListException`, dan blok `catch` di `main` menangkapnya lalu mencetak pesannya; tanpa `try`, program akan berhenti dengan *stack trace*.

Versi Python menyimpan kelas dan fungsi-fungsi bantunya dalam satu *modul*, `cons_list.py` (**Kode 8.7**), karena pemrogram Python mengelompokkan definisi yang berkaitan menurut berkas, bukan menurut kelas. **Kode 8.8** adalah perkenalan yang sama.

Kode 8.7 `cons_list.py`: kelas *list*, eksepsinya, dan operasi-operasi bantunya dalam satu modul Python.

```
# cons_list.py - list bilangan bulat imutabel yang dibangun dari nil() dan cons(),
# dengan operasi bantu ListOps sebagai fungsi biasa.


class ListException(Exception):
    """Dimunculkan bila suatu operasi memerlukan list tak kosong."""


class List:
    __slots__ = ("_empty", "_head", "_tail")        # atribut lain tidak diizinkan

    def __init__(self, empty, head=0, tail=None):
        self._empty = empty
        self._head = head
        self._tail = tail

    @staticmethod
    def nil():
        """List kosong, ditulis [] dalam komentar bab ini."""
        return _NIL

    @staticmethod
    def cons(x, t):
        """List baru yang elemen pertamanya x dan sisanya t."""
        return List(False, x, t)

    @staticmethod
    def of(*xs):
        """Kemudahan: List.of(2, -5, 8) sama dengan cons(2, cons(-5, cons(8, nil()))))."""
        result = List.nil()
        for x in reversed(xs):                      # bangun dari belakang
            result = List.cons(x, result)
        return result

    def empty(self):
        return self._empty

    def head(self):
        if self._empty:
            raise ListException("head() of the empty list")
        return self._head
```

Kode 8.7 (Lanjutan) *cons_list.py: kelas list, eksepsinya, dan operasi-operasi bantunya dalam satu modul Python.*

```
    def tail(self):
        if self._empty:
            raise ListException("tail() of the empty list")
        return self._tail

    def __str__(self):
        items = []
        p = self
        while not p._empty:
            items.append(str(p._head))
            p = p._tail
        return "[" + ", ".join(items) + "]"


_NIL = List(True)                                   # satu list kosong yang dipakai bersama


def append(a, b):
    """Seluruh a, diikuti seluruh b."""
    if a.empty():
        return b
    return List.cons(a.head(), append(a.tail(), b))


def add_to_end(a, x):
    """Salinan a dengan x ditambahkan di akhir."""
    return append(a, List.cons(x, List.nil()))


def reverse(a):
    """Elemen-elemen a dalam urutan terbalik."""
    if a.empty():
        return List.nil()
    return add_to_end(reverse(a.tail()), a.head())


def length(a):
    """Banyaknya elemen a."""
    return 0 if a.empty() else 1 + length(a.tail())
```

Kode 8.7 (Lanjutan) *cons_list.py: kelas list, eksepsinya, dan operasi-operasi bantunya dalam satu modul Python.*

```
def max_of(a):
    """Elemen terbesar; list kosong tidak memilikinya, jadi itu galat."""
    if a.empty():
        raise ListException("max() of the empty list")
    if a.tail().empty():
        return a.head()
    return max(a.head(), max_of(a.tail()))
```

Kode 8.8 `list_start.py`: perkenalan pertama dengan modul *list* Python.

```
# list_start.py - perkenalan pertama dengan List, fungsi-fungsi bantu dan ListException.
from cons_list import List, ListException, append, add_to_end, reverse, length, max_of

a = List.cons(2, List.cons(-5, List.cons(8, List.nil())))
b = List.of(7, 1)
print("a                =", a)
print("a.head()         =", a.head())
print("a.tail()         =", a.tail())
print("append(a, b)     =", append(a, b))
print("add_to_end(a, 0) =", add_to_end(a, 0))
print("reverse(a)       =", reverse(a))
print("length(a)        =", length(a))
print("max_of(a)        =", max_of(a))
print("a is unchanged   =", a)
try:
    max_of(List.nil())
except ListException as e:
    print("caught ListException:", e)
```

```
a                = [2, -5, 8]
a.head()         = 2
a.tail()         = [-5, 8]
append(a, b)     = [2, -5, 8, 7, 1]
add_to_end(a, 0) = [2, -5, 8, 0]
reverse(a)       = [8, -5, 2]
length(a)        = 3
max_of(a)        = 8
a is unchanged   = [2, -5, 8]
caught ListException: max() of the empty list
```

Ada dua rincian Python yang berbeda dari Java. `__slots__` menyatakan hanya tiga nama atribut yang boleh dimiliki sebuah objek, sehingga salah ketik seperti `a._haed = 3` memunculkan galat alih-alih diam-diam menciptakan atribut baru. Selain itu, fungsi bantu untuk nilai maksimum bernama `max_of`, karena `max` adalah fungsi bawaan Python yang tidak ingin kita tutupi. Di Python semua eksepsi *unchecked*, sehingga `ListException` cukup memperluas `Exception`.

Dengan kedua struktur di tangan, kita dapat mulai menyelesaikan soal.

8.4 Soal-Soal tentang Pohon Biner

Ketujuh soal dalam subbab ini semuanya memperluas kelas `Tree` dari Subbab 8.3.1. Soal 8.1 membangun sebuah pohon dengan tangan; Soal 8.2 sampai 8.5 masing-masing menambahkan metode rekursif yang mengunjungi setiap simpul; Soal 8.6 dan 8.7 menambahkan metode untuk *pohon pencarian biner* (*binary search tree*), yaitu pohon yang nilai-nilainya disusun sedemikian rupa sehingga pencarian cukup mengunjungi satu lintasan saja.

8.4.1 Soal 8.1 — Membangun pohon dengan tangan

> **SOAL.** Write Java code to create the tree of **Gambar 8.1** using `new Tree` statements. You will need nine such statements.
>
> *Dengan kata lain:* tulislah kode Java yang membangun pohon pada gambar itu dengan pernyataan `new Tree`; diperlukan sembilan pernyataan semacam itu.

Dengan bahasa kita sendiri. Kita memiliki dua konstruktor, `Tree(int root)` untuk daun dan `Tree(int root, Tree left, Tree right)` untuk cabang. Dengan memakai `new` tepat satu kali untuk setiap simpul, kita harus menghasilkan susunan objek yang bentuknya sama dengan gambar: 1 di puncak, 2 dan 3 di bawahnya, 4 dan 5 di bawah 2, 6 dan 7 di bawah 3, 8 tergantung di *kanan* 4, dan 9 tergantung di *kiri* 7.

Rancangan. Pengamatan kuncinya menyangkut *urutan*. Konstruktor cabang memerlukan kedua subpohonnya sebagai argumen, sehingga sebuah subpohon harus sudah ada sebelum induknya dapat dibangun. Karena itu pohon harus dibangun *dari bawah ke atas*: daun lebih dulu, lalu induknya, lalu akarnya. **Tabel 8.2** mendaftar kesembilan pernyataan dalam urutan yang sah, beserta tingkat setiap simpul yang dihitung dari akar pada tingkat 0.

Tabel 8.2 Kesembilan pemanggilan konstruktor pada Soal 8.1 dalam urutan dari bawah ke atas. Sebuah simpul baru dapat dibangun setelah kedua subpohonnya ada.

#	Simpul	Tingkat	Subpohon kiri	Subpohon kanan	Konstruktor
1–4	5, 6, 8, 9	2, 2, 3, 3	tidak ada	tidak ada	`Tree(int)`
5	4	2	tidak ada	8	`Tree(4, null, leaf8)`
6	7	2	9	tidak ada	`Tree(7, leaf9, null)`
7	2	1	4	5	`Tree(2, t4, leaf5)`
8	3	1	6	7	`Tree(3, leaf6, t7)`
9	1	0	2	3	`Tree(1, t2, t3)`

Penyelesaian Java. **Kode 8.9** membangun pohon di dalam `build()`, memeriksa bentuk dan ukurannya, dan juga membangun versi keliru yang menggoda yang dibahas di bawah.

Kode 8.9 `BuildTree.java`: kesembilan pernyataan `new Tree`, `size()` yang rekursif, dan pemeriksaan diri.

```java
// BuildTree.java - Soal 8.1: membangun pohon sembilan simpul dengan sembilan pernyataan "new Tree".
public class BuildTree {

    static final class Tree {
        private int root;
        private Tree left, right;

        Tree(int root, Tree left, Tree right) {
            this.root = root;
            this.left = left;
            this.right = right;
        }

        Tree(int root) {
            this(root, null, null);
        }

        /** Banyaknya simpul: 1 untuk saya, ditambah laporan subpohon-subpohon saya. */
        int size() {
            int l = (left == null) ? 0 : left.size();
            int r = (right == null) ? 0 : right.size();
            return 1 + l + r;
        }

        @Override
        public String toString() {
            if (left == null && right == null) {
                return Integer.toString(root);
            }
            String l = (left == null) ? "-" : left.toString();
            String r = (right == null) ? "-" : right.toString();
            return root + "(" + l + "," + r + ")";
        }
    }

    /** Membangun pohon dari bawah ke atas: anak harus ada sebelum induknya. */
    static Tree build() {
        Tree leaf5 = new Tree(5);              // 1
        Tree leaf6 = new Tree(6);              // 2
        Tree leaf8 = new Tree(8);              // 3
        Tree leaf9 = new Tree(9);              // 4
        Tree t4 = new Tree(4, null, leaf8);    // 5: 8 tergantung di KANAN 4
        Tree t7 = new Tree(7, leaf9, null);    // 6: 9 tergantung di KIRI 7
        Tree t2 = new Tree(2, t4, leaf5);      // 7
        Tree t3 = new Tree(3, leaf6, t7);      // 8
        return new Tree(1, t2, t3);            // 9
```

Kode 8.9 (Lanjutan) *BuildTree.java: kesembilan pernyataan new Tree, size() yang rekursif, dan pemeriksaan diri.*

```
    }

    /** Menggoda tetapi salah: setiap anak tunggal diletakkan di kiri. */
    static Tree buildWrong() {
        Tree t4 = new Tree(4, new Tree(8), null);
        Tree t7 = new Tree(7, new Tree(9), null);
        return new Tree(1, new Tree(2, t4, new Tree(5)), new Tree(3, new Tree(6), t7));
    }

    private static int passed = 0;

    private static void check(String name, Object expected, Object actual) {
        if (!java.util.Objects.equals(expected, actual)) {
            throw new AssertionError(name + ": expected " + expected + " but got " +
actual);
        }
        passed++;
        System.out.println("PASS " + name);
    }

    public static void main(String[] args) {
        Tree t = build();
        System.out.println("t       = " + t);
        System.out.println("wrong t = " + buildWrong());
        check("shape", "1(2(4(-,8),5),3(6,7(9,-)))", t.toString());
        check("size", 9, t.size());
        check("wrong shape differs", false, t.toString().equals(buildWrong().toString()));
        System.out.println("All " + passed + " checks passed.");
    }
}
```

```
t       = 1(2(4(-,8),5),3(6,7(9,-)))
wrong t = 1(2(4(8,-),5),3(6,7(9,-)))
PASS shape
PASS size
PASS wrong shape differs
All 3 checks passed.
```

Penyelesaian Python. **Kode 8.10** adalah program yang sama. Uji-ujinya adalah metode-metode dari sebuah kelas `unittest.TestCase` yang namanya diawali `test_`; `unittest.main(verbosity=2)` menemukan dan menjalankannya, lalu mencetak satu baris untuk setiap uji.

Kode 8.10 `build_tree.py`: pohon yang sama dalam Python, diuji dengan `unittest`.

```
# build_tree.py - Soal 8.1: membangun pohon sembilan simpul dengan sembilan panggilan
Tree(...).
import sys
import unittest


class Tree:
    def __init__(self, root, left=None, right=None):
        self._root = root
        self._left = left
        self._right = right

    def size(self):
        """Banyaknya simpul: 1 untuk saya, ditambah laporan subpohon-subpohon saya."""
        l = 0 if self._left is None else self._left.size()
        r = 0 if self._right is None else self._right.size()
        return 1 + l + r

    def __str__(self):
        if self._left is None and self._right is None:
            return str(self._root)
        l = "-" if self._left is None else str(self._left)
        r = "-" if self._right is None else str(self._right)
        return f"{self._root}({l},{r})"


def build():
    """Membangun pohon dari bawah ke atas: anak harus ada sebelum induknya."""
    leaf5 = Tree(5)                    # 1
    leaf6 = Tree(6)                    # 2
    leaf8 = Tree(8)                    # 3
    leaf9 = Tree(9)                    # 4
    t4 = Tree(4, None, leaf8)          # 5: 8 tergantung di KANAN 4
    t7 = Tree(7, leaf9, None)          # 6: 9 tergantung di KIRI 7
    t2 = Tree(2, t4, leaf5)            # 7
    t3 = Tree(3, leaf6, t7)            # 8
    return Tree(1, t2, t3)             # 9


def build_wrong():
    """Menggoda tetapi salah: setiap anak tunggal diletakkan di kiri."""
    t4 = Tree(4, Tree(8))
    t7 = Tree(7, Tree(9))
    return Tree(1, Tree(2, t4, Tree(5)), Tree(3, Tree(6), t7))
```

Kode 8.10 (Lanjutan) *build_tree.py: pohon yang sama dalam Python, diuji dengan unittest.*

```
class TestBuild(unittest.TestCase):
    def test_shape(self):
        self.assertEqual("1(2(4(-,8),5),3(6,7(9,-)))", str(build()))

    def test_size(self):
        self.assertEqual(9, build().size())

    def test_wrong_shape_differs(self):
        self.assertNotEqual(str(build()), str(build_wrong()))


if __name__ == "__main__":
    print("t       =", build())
    print("wrong t =", build_wrong())
    sys.stdout.flush()
    unittest.main(verbosity=2)
```

```
t       = 1(2(4(-,8),5),3(6,7(9,-)))
wrong t = 1(2(4(8,-),5),3(6,7(9,-)))
test_shape (__main__.TestBuild.test_shape) ... ok
test_size (__main__.TestBuild.test_size) ... ok
test_wrong_shape_differs (__main__.TestBuild.test_wrong_shape_differs) ... ok

----------------------------------------------------------------------
Ran 3 tests in 0.000s

OK
```

Baris demi baris. Pernyataan 1 sampai 4 menciptakan keempat daun; belum ada yang menunjuk ke sana. Pernyataan 5, `new Tree(4, null, leaf8)`, perlu dibaca dua kali: argumen *kedua* adalah subpohon kiri, dan nilainya `null`, karena dalam gambar 8 tergantung di sisi kanan 4. Pernyataan 6 adalah bayangan cerminnya: 9 adalah anak *kiri* 7, jadi `leaf9` menjadi argumen kedua dan `null` argumen ketiga. Pernyataan 7 dan 8 membangun kedua simpul tengah dari objek-objek yang kini sudah ada, dan pernyataan 9 membangun akar lalu mengembalikannya. `size()` adalah metode delegasi kita yang pertama: sebuah simpul melaporkan 1 untuk dirinya ditambah apa pun yang dilaporkan subpohonnya, dan subpohon yang tidak ada menyumbang 0. Pemeriksaan bahwa `size` sama dengan 9 memastikan tidak ada simpul yang terlupa dan tidak ada yang dibangun dua kali. (Baris uji Python berbunyi `test_shape (__main__.TestBuild.test_shape)` pada Python 3.11 ke atas; Python 3.10 mencetak bentuk yang lebih pendek, `test_shape (__main__.TestBuild)`.)

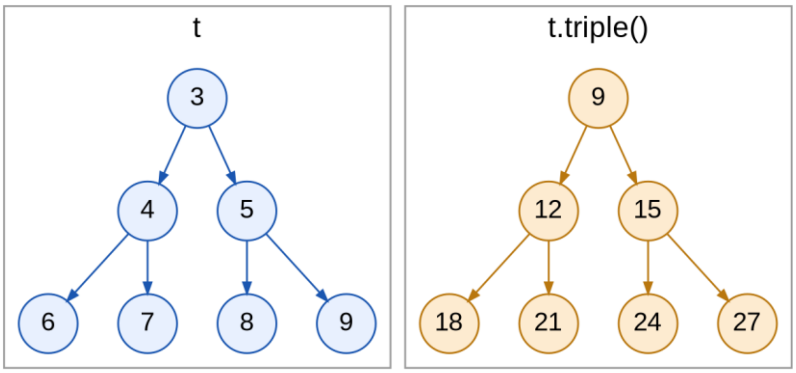
Penyelesaian keliru yang menggoda. Dalam gambar tangan, anak tunggal sering tampak tergantung "di bawah" induknya, dan pemanggilan konstruktor yang paling mudah adalah `new Tree(4, new Tree(8), null)`: anaknya diletakkan di posisi pertama. Program memperlihatkan hasilnya: `wrong t = 1(2(4(8,-),5),3(6,7(9,-)))`. Simpul 8 kini menjadi anak *kiri*. Pohonnya tetap memiliki sembilan simpul, sehingga uji yang hanya memeriksa ukuran akan lolos. Hanya uji terhadap bentuk yang persis yang dapat menangkapnya, itulah sebabnya pemeriksaan `shape` membandingkan *string* secara utuh. Pelajarannya berlaku umum: pada pohon, urutan kiri–kanan adalah bagian dari data.

> **UJI DIRI.** Tulislah bentuk `toString()` dari pohon dengan akar 10, yang subpohon kirinya daun 20 dan subpohon kanannya 30 dengan hanya anak kiri 40. Berapa pernyataan `new` yang diperlukan, dan dalam urutan apa?
>
> *Jawaban.* `10(20,30(40,-))`. Empat pernyataan: daun 20 dan 40 lebih dulu (urutannya bebas), lalu `new Tree(30, leaf40, null)`, kemudian `new Tree(10, leaf20, t30)`.

8.4.2 Soal 8.2 — Melipattigakan setiap simpul

> **SOAL.** Include a recursive method `public void triple()` in `Tree` that multiplies every node of the tree by three. For example, the tree with root 3, children 4 and 5, and grandchildren 6, 7, 8, 9 becomes the tree with root 9, children 12 and 15, and grandchildren 18, 21, 24, 27. Hint: ask your left and right subtrees to apply the procedure to themselves, then multiply your root by three.
>
> *Dengan kata lain:* tambahkan ke `Tree` metode rekursif `public void triple()` yang mengalikan setiap simpul dengan tiga. Pohon berakar 3 dengan anak 4 dan 5 serta cucu 6, 7, 8, 9 menjadi pohon berakar 9 dengan anak 12 dan 15 serta cucu 18, 21, 24, 27. Petunjuk: mintalah subpohon kiri dan kanan menerapkan prosedur itu pada dirinya, lalu kalikan akar Anda dengan tiga.

Dengan bahasa kita sendiri. `triple()` mengubah pohon di tempat; ia tidak mengembalikan apa-apa (`void`). Setelah pemanggilan, setiap nilai menjadi tiga kali nilai semula, dan bentuk pohonnya tidak berubah. **Gambar 8.4** memperlihatkan contohnya.



Gambar 8.4 Pohon contoh sebelum dan sesudah `t.triple()`: bentuknya sama dan setiap nilai dikalikan tiga.

Rancangan. Ketiga pertanyaan resep delegasi memiliki jawaban yang pendek (**Tabel 8.3**).

Tabel 8.3 Resep delegasi yang diterapkan pada `triple()`.

Pertanyaan	Jawaban untuk `triple()`
Kasus dasar	Subpohon yang tidak ada: tidak melakukan apa-apa (dijaga oleh `if (left != null)`)
Delegasi	`left.triple()` dan `right.triple()`
Tugas saya sendiri	`root = 3 * root`
Nilai kembalian	tidak ada; pohon diubah di tempat

Penyelesaian Java. **Kode 8.11** memuat metodenya, sebuah varian yang keliru, dan pemeriksaannya.

Kode 8.11 `TripleTree.java`: melipattigakan setiap simpul melalui delegasi, dengan varian keliru yang berakhir *crash*.

```java
// TripleTree.java - Soal 8.2: mengalikan setiap simpul pohon dengan tiga.
public class TripleTree {

    static final class Tree {
        private int root;
        private Tree left, right;

        Tree(int root, Tree left, Tree right) {
            this.root = root;
            this.left = left;
            this.right = right;
        }

        Tree(int root) {
            this(root, null, null);
        }

        /** Melipattigakan setiap simpul: subpohon dahulu (melalui delegasi), lalu akar saya. */
        void triple() {
            if (left != null) {
                left.triple();
            }
            if (right != null) {
                right.triple();
            }
            root = 3 * root;
        }

        /** Menggoda tetapi salah: lupa bahwa subpohon bisa saja tidak ada. */
        void tripleWrong() {
            root = 3 * root;
            left.tripleWrong();
            right.tripleWrong();
        }

        @Override
        public String toString() {
            if (left == null && right == null) {
                return Integer.toString(root);
            }
            String l = (left == null) ? "-" : left.toString();
            String r = (right == null) ? "-" : right.toString();
            return root + "(" + l + "," + r + ")";
        }
    }
```

Kode 8.11 (Lanjutan) *TripleTree.java: melipattigakan setiap simpul melalui delegasi, dengan varian keliru yang berakhir crash.*

```
    static Tree sample() {
        return new Tree(3,
                new Tree(4, new Tree(6), new Tree(7)),
                new Tree(5, new Tree(8), new Tree(9)));
    }

    private static int passed = 0;

    private static void check(String name, Object expected, Object actual) {
        if (!java.util.Objects.equals(expected, actual)) {
            throw new AssertionError(name + ": expected " + expected + " but got " +
actual);
        }
        passed++;
        System.out.println("PASS " + name);
    }

    public static void main(String[] args) {
        Tree t = sample();
        System.out.println("before: " + t);
        t.triple();
        System.out.println("after : " + t);
        check("triple sample", "9(12(18,21),15(24,27))", t.toString());

        Tree lopsided = new Tree(1, new Tree(2), null);   // tanpa subpohon kanan
        lopsided.triple();
        check("triple lopsided", "3(6,-)", lopsided.toString());
        Tree leaf = new Tree(-4);
        leaf.triple();
        check("triple leaf", "-12", leaf.toString());

        Tree w = sample();
        try {
            w.tripleWrong();
        } catch (NullPointerException e) {
            System.out.println("tripleWrong crashed: NullPointerException");
            System.out.println("half-done tree   : " + w);
        }
        System.out.println("All " + passed + " checks passed.");
    }
}
```

```
before: 3(4(6,7),5(8,9))
after : 9(12(18,21),15(24,27))
PASS triple sample
PASS triple lopsided
PASS triple leaf
tripleWrong crashed: NullPointerException
half-done tree   : 9(12(18,7),5(8,9))
All 3 checks passed.
```

Penyelesaian Python. **Kode 8.12** mengikuti rencana yang sama.

Kode 8.12 `triple_tree.py`: melipattigakan setiap simpul dalam Python.

```python
# triple_tree.py - Soal 8.2: mengalikan setiap simpul pohon dengan tiga.
import sys
import unittest


class Tree:
    def __init__(self, root, left=None, right=None):
        self._root = root
        self._left = left
        self._right = right

    def triple(self):
        """Melipattigakan setiap simpul: subpohon dahulu (melalui delegasi), lalu akar
saya."""
        if self._left is not None:
            self._left.triple()
        if self._right is not None:
            self._right.triple()
        self._root = 3 * self._root

    def triple_wrong(self):
        """Menggoda tetapi salah: lupa bahwa subpohon bisa saja tidak ada."""
        self._root = 3 * self._root
        self._left.triple_wrong()
        self._right.triple_wrong()

    def __str__(self):
        if self._left is None and self._right is None:
            return str(self._root)
        l = "-" if self._left is None else str(self._left)
        r = "-" if self._right is None else str(self._right)
        return f"{self._root}({l},{r})"


def sample():
    return Tree(3, Tree(4, Tree(6), Tree(7)), Tree(5, Tree(8), Tree(9)))


class TestTriple(unittest.TestCase):
    def test_sample(self):
        t = sample()
        t.triple()
        self.assertEqual("9(12(18,21),15(24,27))", str(t))
```

Kode 8.12 (Lanjutan) *triple_tree.py: melipattigakan setiap simpul dalam Python.*

```
    def test_lopsided(self):
        t = Tree(1, Tree(2))
        t.triple()
        self.assertEqual("3(6,-)", str(t))

    def test_wrong_crashes(self):
        with self.assertRaises(AttributeError):
            sample().triple_wrong()


if __name__ == "__main__":
    t = sample()
    print("before:", t)
    t.triple()
    print("after :", t)
    w = sample()
    try:
        w.triple_wrong()
    except AttributeError as e:
        print("triple_wrong crashed:", e)
        print("half-done tree      :", w)
    sys.stdout.flush()
    unittest.main(verbosity=2)
```

```
before: 3(4(6,7),5(8,9))
after : 9(12(18,21),15(24,27))
triple_wrong crashed: 'NoneType' object has no attribute 'triple_wrong'
half-done tree      : 9(12(18,7),5(8,9))
test_lopsided (__main__.TestTriple.test_lopsided) ... ok
test_sample (__main__.TestTriple.test_sample) ... ok
test_wrong_crashes (__main__.TestTriple.test_wrong_crashes) ... ok

----------------------------------------------------------------------
Ran 3 tests in 0.000s

OK
```

Baris demi baris. Metode ini persis berbentuk `mirror()` pada **Kode 8.1**: dua delegasi yang dijaga, lalu tugas simpul itu sendiri. Urutan ketiga langkah itu tidak penting di sini, karena melipattigakan sebuah simpul tidak bergantung pada anak-anaknya; petunjuknya berkata "subpohon lebih dulu", dan kita mengikutinya. Uji `triple lopsided` membangun pohon yang memiliki subpohon kiri tetapi tidak memiliki subpohon kanan, yaitu kasus yang membongkar pemeriksaan `null` yang terlupa; `triple leaf` memeriksa pohon terkecil yang mungkin beserta sebuah nilai negatif.

Penyelesaian keliru yang menggoda. `tripleWrong()` membuang kedua pemeriksaan `if`, dengan alasan bahwa "setiap simpul memiliki dua anak". Kenyataannya tidak: daun tidak punya anak. Hasil eksekusinya memperlihatkan apa yang terjadi. Akar menjadi 9, anak kiri menjadi 12, daun 6 menjadi 18, lalu daun itu memanggil `left.tripleWrong()` pada referensi `null`. Java melempar `NullPointerException`; Python memunculkan `AttributeError: 'NoneType' object has no attribute 'triple_wrong'`. Yang lebih buruk daripada *crash*-nya adalah apa yang ditinggalkannya: `9(12(18,7),5(8,9))`, pohon yang hanya sebagian dilipattigakan. Eksepsi di tengah metode yang mengubah sebuah objek dapat meninggalkan objek itu dalam keadaan yang tidak akan pernah dihasilkan program yang benar; ini salah satu alasan Bab 5 menganjurkan agar masukan diperiksa *sebelum* mengubah apa pun.

> **UJI DIRI.** Tulislah `void addToAll(int k)` yang menambahkan `k` ke setiap simpul. Menjadi apakah `1(2,-)` setelah `addToAll(10)`?

Jawaban. Bentuknya sama, dengan `root = root + k` sebagai tugas simpul itu sendiri: `11(12,-)`. Penjaga `null` tetap persis seperti pada `triple()`.

8.4.3 Soal 8.3 — Mencetak penelusuran *depth-first*

SOAL. Include a recursive method `public void printDepthFirst()` in `Tree` that prints the depth-first traversal of the tree, choosing left branches first. For the tree of **Gambar 8.4** (left) it should print `3 4 6 7 5 8 9`. Hint: print your root, then ask your left subtree to print its traversal, and finally ask your right subtree.

Dengan kata lain: tambahkan ke `Tree` metode rekursif `public void printDepthFirst()` yang mencetak penelusuran *depth-first* (mendalam lebih dulu) dengan cabang kiri didahulukan. Untuk pohon contoh (sebelum dilipattigakan), keluarannya `3 4 6 7 5 8 9`. Petunjuk: cetak akar Anda, lalu minta subpohon kiri mencetak penelusurannya, dan terakhir minta subpohon kanan.

Dengan bahasa kita sendiri. Sebuah *penelusuran* (*traversal*) mengunjungi setiap simpul tepat satu kali. *Mendalam lebih dulu, kiri lebih dulu, akar sebelum anak* disebut juga *pre-order*: sebuah simpul dicetak sebelum apa pun di bawahnya, dan seluruh subpohon kirinya dicetak sebelum bagian mana pun dari subpohon kanannya.

Rancangan. **Tabel 8.4** menelusuri pemanggilan pada pohon contoh. Setiap baris adalah satu pemanggilan; indentasi kolom pertama menunjukkan kedalaman rekursi.

Tabel 8.4 Ketujuh pemanggilan `printDepthFirst()` pada pohon contoh, dalam urutan terjadinya. Membaca kolom tengah dari atas ke bawah menghasilkan keluarannya.

Pemanggilan	Mencetak	Lalu memanggil
simpul 3	3	simpul 4, lalu simpul 5
simpul 4	4	simpul 6, lalu simpul 7
simpul 6	6	tidak ada (daun)
simpul 7	7	tidak ada (daun)
simpul 5	5	simpul 8, lalu simpul 9
simpul 8	8	tidak ada (daun)
simpul 9	9	tidak ada (daun)

Metode yang *mencetak* lebih sulit diuji daripada metode yang *mengembalikan nilai*, karena hasilnya pergi ke konsol alih-alih kembali ke pemanggil. Bab 6 memperlihatkan obatnya: alihkan konsol sementara ke sebuah penyangga (*buffer*), jalankan metodenya, pulihkan konsol, lalu bandingkan isi penyangga dengan teks yang diharapkan.

Penyelesaian Java. **Kode 8.13** melakukan persis itu di dalam metode bantu `captured`.

Kode 8.13 `DepthFirstPrint.java`: pencetakan *pre-order*, dengan metode bantu yang menangkap keluaran konsol untuk pengujian.

```java
// DepthFirstPrint.java - Soal 8.3: mencetak penelusuran depth-first (pre-order).
import java.io.ByteArrayOutputStream;
import java.io.PrintStream;

public class DepthFirstPrint {

    static final class Tree {
        private int root;
        private Tree left, right;

        Tree(int root, Tree left, Tree right) {
            this.root = root;
            this.left = left;
            this.right = right;
        }

        Tree(int root) {
            this(root, null, null);
        }

        /** Akar dahulu, lalu seluruh subpohon kiri, lalu seluruh subpohon kanan. */
        void printDepthFirst() {
            System.out.print(root + " ");
            if (left != null) {
                left.printDepthFirst();
            }
            if (right != null) {
                right.printDepthFirst();
            }
        }

        /** Menggoda tetapi salah: akar dicetak di antara subpohon-subpohonnya. */
        void printDepthFirstWrong() {
            if (left != null) {
                left.printDepthFirstWrong();
            }
            System.out.print(root + " ");
            if (right != null) {
                right.printDepthFirstWrong();
            }
        }
    }
```

Kode 8.13 (Lanjutan) *DepthFirstPrint.java: pencetakan pre-order, dengan metode bantu yang menangkap keluaran konsol untuk pengujian.*

```
    /** Menjalankan suatu aksi dan mengembalikan semua yang dicetaknya (untuk pengujian).
*/
    static String captured(Runnable action) {
        PrintStream original = System.out;
        ByteArrayOutputStream buffer = new ByteArrayOutputStream();
        System.setOut(new PrintStream(buffer));
        try {
            action.run();
        } finally {
            System.setOut(original);        // selalu pulihkan konsol yang asli
        }
        return buffer.toString();
    }

    private static int passed = 0;

    private static void check(String name, Object expected, Object actual) {
        if (!java.util.Objects.equals(expected, actual)) {
            throw new AssertionError(name + ": expected " + expected + " but got " +
actual);
        }
        passed++;
        System.out.println("PASS " + name);
    }

    public static void main(String[] args) {
        Tree t = new Tree(3,
                new Tree(4, new Tree(6), new Tree(7)),
                new Tree(5, new Tree(8), new Tree(9)));
        t.printDepthFirst();
        System.out.println();
        t.printDepthFirstWrong();
        System.out.println(" <- wrong order");
        check("sample", "3 4 6 7 5 8 9 ", captured(t::printDepthFirst));
        Tree chain = new Tree(1, null, new Tree(2, new Tree(3), null));
        check("chain", "1 2 3 ", captured(chain::printDepthFirst));
        System.out.println("All " + passed + " checks passed.");
    }
}
```

```
3 4 6 7 5 8 9
6 4 7 3 8 5 9  <- wrong order
PASS sample
PASS chain
All 2 checks passed.
```

Penyelesaian Python. Pada **Kode 8.14**, `contextlib.redirect_stdout` melakukan pengalihan itu dan membatalkannya secara otomatis di akhir blok `with`.

Kode 8.14 `depth_first_print.py`: pencetakan *pre-order* dalam Python, diuji dengan menangkap keluaran standar.

```python
# depth_first_print.py - Soal 8.3: mencetak penelusuran depth-first (pre-order).
import io
import sys
import unittest
from contextlib import redirect_stdout


class Tree:
    def __init__(self, root, left=None, right=None):
        self._root = root
        self._left = left
        self._right = right

    def print_depth_first(self):
        """Akar dahulu, lalu seluruh subpohon kiri, lalu seluruh subpohon kanan."""
        print(self._root, end=" ")
        if self._left is not None:
            self._left.print_depth_first()
        if self._right is not None:
            self._right.print_depth_first()

    def print_depth_first_wrong(self):
        """Menggoda tetapi salah: akar dicetak di antara subpohon-subpohonnya."""
        if self._left is not None:
            self._left.print_depth_first_wrong()
        print(self._root, end=" ")
        if self._right is not None:
            self._right.print_depth_first_wrong()


def captured(action):
    """Menjalankan suatu aksi dan mengembalikan semua yang dicetaknya (untuk pengujian)."""
    buffer = io.StringIO()
    with redirect_stdout(buffer):       # dipulihkan otomatis sesudahnya
        action()
    return buffer.getvalue()


def sample():
    return Tree(3, Tree(4, Tree(6), Tree(7)), Tree(5, Tree(8), Tree(9)))


class TestDepthFirst(unittest.TestCase):
    def test_sample(self):
        self.assertEqual("3 4 6 7 5 8 9 ", captured(sample().print_depth_first))
```

Kode 8.14 (Lanjutan) *depth_first_print.py: pencetakan pre-order dalam Python, diuji dengan menangkap keluaran standar.*

```python
    def test_chain(self):
        chain = Tree(1, None, Tree(2, Tree(3)))
        self.assertEqual("1 2 3 ", captured(chain.print_depth_first))


if __name__ == "__main__":
    sample().print_depth_first()
    print()
    sample().print_depth_first_wrong()
    print(" <- wrong order")
    sys.stdout.flush()
    unittest.main(verbosity=2)
```

```
3 4 6 7 5 8 9
6 4 7 3 8 5 9  <- wrong order
test_chain (__main__.TestDepthFirst.test_chain) ... ok
test_sample (__main__.TestDepthFirst.test_sample) ... ok

----------------------------------------------------------------------
Ran 2 tests in 0.000s

OK
```

Baris demi baris. `printDepthFirst()` mencetak `root + " "` lebih dulu, sehingga setiap bilangan diikuti satu spasi, termasuk bilangan terakhir; karena itu *string* yang diharapkan dalam pemeriksaan berakhir dengan spasi, `"3 4 6 7 5 8 9 "`. Di dalam `captured`, `System.setOut` menukar konsol dengan sebuah `PrintStream` yang menulis ke `ByteArrayOutputStream`. Blok `finally` memulihkan konsol asli *bahkan bila aksi itu melempar eksepsi*; tanpanya, satu uji yang gagal akan membuat semua pesan sesudahnya tidak terlihat. Tipe parameter `Runnable` adalah antarmuka dari pustaka Java dengan satu metode `run()`, dan *method reference* `t::printDepthFirst` adalah cara ringkas menulis sebuah objek yang mengimplementasikannya dengan memanggil `t.printDepthFirst()`. Uji kedua memakai sebuah "rantai", yaitu pohon yang setiap simpulnya memiliki satu anak, bentuk yang menangkap kode yang menganggap selalu ada dua anak.

Penyelesaian keliru yang menggoda. Memindahkan `print` ke antara kedua pemanggilan rekursif tampak tidak berbahaya, tetapi mengubah penelusurannya. `printDepthFirstWrong()` mencetak `6 4 7 3 8 5 9`: setiap subpohon kiri, lalu simpulnya, lalu subpohon kanan. Itulah penelusuran *in-order*, yang berguna dengan caranya sendiri (Soal 8.7 memakainya), tetapi bukan yang diminta. Posisi tugas simpul itu sendiri relatif terhadap delegasinya adalah satu-satunya perbedaan antara *pre-order*, *in-order*, dan *post-order*.

> **UJI DIRI.** Apa yang dicetak penelusuran *post-order* (subpohon kiri, subpohon kanan, lalu simpulnya) untuk pohon contoh, dan untuk pohon pada **Gambar 8.1**?
>
> *Jawaban.* Pohon contoh: `6 7 4 8 9 5 3`. **Gambar 8.1**: `8 4 5 2 6 9 7 3 1`. Pada *post-order*, akar selalu dicetak paling akhir.

8.4.4 Soal 8.4 — Salinan segar sebuah pohon

> **SOAL.** Include a recursive method `public Tree createFreshCopy()` in `Tree` that generates a fresh copy of a tree. After `Tree u = t.createFreshCopy(); t.mirror();`, the tree `u` must be exactly as `t` was before mirroring. Use one `new` statement per node; do not use Java's `clone` facility. Hint: ask your subtrees for fresh copies of themselves, then use them to build a fresh copy of yourself with `new Tree`.
>
> *Dengan kata lain:* tambahkan ke `Tree` metode rekursif `public Tree createFreshCopy()` yang menghasilkan salinan segar sebuah pohon. Setelah `Tree u = t.createFreshCopy(); t.mirror();`, pohon `u` harus persis seperti `t` sebelum dicerminkan. Pakailah satu pernyataan `new` untuk setiap simpul, tanpa fasilitas `clone` Java. Petunjuk: mintalah salinan segar dari subpohon-subpohon Anda, lalu pakailah untuk membangun salinan segar diri Anda dengan `new Tree`.

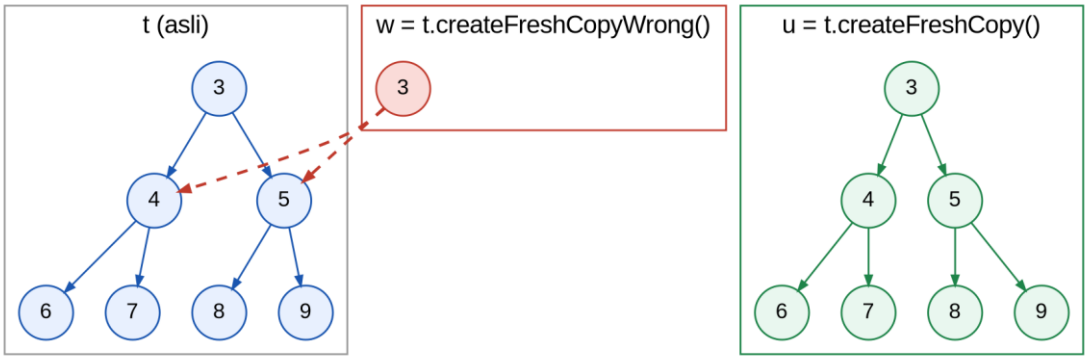
Dengan bahasa kita sendiri. Kita memerlukan *salinan dalam* (*deep copy*): pohon kedua dengan bentuk dan nilai yang sama, tersusun dari objek-objek simpul yang seluruhnya baru, sehingga mengubah satu pohon tidak dapat memengaruhi pohon yang lain. Uji yang diberikan dalam pernyataan soal tepat merupakan uji kemandirian: cerminkan pohon aslinya, dan salinannya tidak boleh ikut bergerak.

Rancangan. Salinan segar sebuah simpul adalah simpul baru yang menyimpan akar yang sama dan *salinan segar* dari kedua subpohonnya. Salinan segar dari subpohon yang tidak ada adalah subpohon yang tidak ada. **Tabel 8.5** menyatakannya dalam istilah resep.

Tabel 8.5 Resep delegasi untuk salinan dalam. Berbeda dengan `triple()`, hasil dari pemanggilan yang didelegasikan justru menjadi inti persoalannya.

Pertanyaan	Jawaban untuk `createFreshCopy()`
Kasus dasar	Subpohon yang tidak ada disalin menjadi `null`
Delegasi	`left.createFreshCopy()` dan `right.createFreshCopy()`, dan *simpan hasilnya*
Tugas saya sendiri	`return new Tree(root, leftCopy, rightCopy)`
Nilai kembalian	pohon yang baru; `this` tidak berubah

Gambar 8.5 memperlihatkan perbedaan antara salinan dalam dan salinan dangkal yang menggoda, yang dibahas di bawah, dalam bentuk objek dan referensi.



Gambar 8.5 Kiri: `t` yang asli. Kanan: salinan segar `u`, tujuh objek baru. Tengah: salinan keliru `w`, satu objek baru yang referensinya menunjuk kembali ke dalam `t`.

Penyelesaian Java. **Kode 8.15** juga memuat metode `sharesNodeWith` yang menjawab "apakah kedua pohon ini memiliki *objek* simpul yang sama?", dengan memakai `==`, yang di Java membandingkan referensi, bukan nilai.

Kode 8.15 `FreshCopy.java`: salinan dalam, salinan dangkal yang keliru, dan uji untuk objek simpul yang dipakai bersama.

```java
// FreshCopy.java - Soal 8.4: salinan dalam ("segar") sebuah pohon, satu "new" per simpul.
public class FreshCopy {

    static final class Tree {
        private int root;
        private Tree left, right;

        Tree(int root, Tree left, Tree right) {
            this.root = root;
            this.left = left;
            this.right = right;
        }

        Tree(int root) {
            this(root, null, null);
        }

        /** Subpohon menyalin dirinya dan menyerahkan salinannya; saya membungkusnya dalam simpul baru. */
        Tree createFreshCopy() {
            Tree leftCopy = (left == null) ? null : left.createFreshCopy();
            Tree rightCopy = (right == null) ? null : right.createFreshCopy();
            return new Tree(root, leftCopy, rightCopy);
        }

        /** Menggoda tetapi salah: hasil rekursif dibuang. */
        Tree createFreshCopyWrong() {
            if (left != null) {
                left.createFreshCopyWrong();      // hasilnya diabaikan!
            }
            if (right != null) {
                right.createFreshCopyWrong();     // hasilnya diabaikan!
            }
            return new Tree(root, left, right);   // berbagi subpohon-subpohon saya
        }

        void mirror() {
            if (left != null) {
                left.mirror();
            }
            if (right != null) {
                right.mirror();
            }
            Tree originalLeft = left;
            left = right;
            right = originalLeft;
        }
```

Kode 8.15 (Lanjutan) *FreshCopy.java: salinan dalam, salinan dangkal yang keliru, dan uji untuk objek simpul yang dipakai bersama.*

```
        /** True bila pohon ini dan pohon lain berbagi sedikitnya satu objek simpul. */
        boolean sharesNodeWith(Tree other) {
            return other != null
                    && (contains(other)
                        || sharesNodeWith(other.left)
                        || sharesNodeWith(other.right));
        }

        /** True bila node (objek yang persis sama) muncul di suatu tempat dalam pohon ini.
*/
        private boolean contains(Tree node) {
            return this == node
                    || (left != null && left.contains(node))
                    || (right != null && right.contains(node));
        }

        @Override
        public String toString() {
            if (left == null && right == null) {
                return Integer.toString(root);
            }
            String l = (left == null) ? "-" : left.toString();
            String r = (right == null) ? "-" : right.toString();
            return root + "(" + l + "," + r + ")";
        }
    }

    static Tree sample() {
        return new Tree(3,
                new Tree(4, new Tree(6), new Tree(7)),
                new Tree(5, new Tree(8), new Tree(9)));
    }

    private static int passed = 0;

    private static void check(String name, Object expected, Object actual) {
        if (!java.util.Objects.equals(expected, actual)) {
            throw new AssertionError(name + ": expected " + expected + " but got " +
actual);
        }
        passed++;
        System.out.println("PASS " + name);
    }
```

Kode 8.15 (Lanjutan) *FreshCopy.java: salinan dalam, salinan dangkal yang keliru, dan uji untuk objek simpul yang dipakai bersama.*

```
    public static void main(String[] args) {
        Tree t = sample();
        Tree u = t.createFreshCopy();
        t.mirror();
        System.out.println("t after mirror  = " + t);
        System.out.println("u (fresh copy)  = " + u);
        check("copy unchanged", "3(4(6,7),5(8,9))", u.toString());
        check("no shared nodes", false, t.sharesNodeWith(u));

        Tree t2 = sample();
        Tree w = t2.createFreshCopyWrong();
        t2.mirror();
        System.out.println("w (wrong copy)  = " + w);
        check("wrong copy shares nodes", true, t2.sharesNodeWith(w));
        System.out.println("All " + passed + " checks passed.");
    }
}
```

```
t after mirror  = 3(5(9,8),4(7,6))
u (fresh copy)  = 3(4(6,7),5(8,9))
PASS copy unchanged
PASS no shared nodes
w (wrong copy)  = 3(4(7,6),5(9,8))
PASS wrong copy shares nodes
All 3 checks passed.
```

Penyelesaian Python. Pada **Kode 8.16**, uji berbagi simpul mengumpulkan identitas `id(n)` dari semua simpul satu pohon ke dalam sebuah himpunan (*set*), lalu menanyakan apakah ada simpul pohon lain yang memiliki salah satu identitas itu.

Kode 8.16 `fresh_copy.py`: salinan dalam di Python, dengan uji berbagi simpul yang sama.

```python
# fresh_copy.py - Soal 8.4: salinan dalam ("segar") sebuah pohon, satu Tree(...) per
simpul.
import sys
import unittest


class Tree:
    def __init__(self, root, left=None, right=None):
        self._root = root
        self._left = left
        self._right = right

    def create_fresh_copy(self):
        """Subpohon menyalin dirinya dan menyerahkan salinannya; saya membungkusnya dalam
simpul baru."""
        left_copy = None if self._left is None else self._left.create_fresh_copy()
        right_copy = None if self._right is None else self._right.create_fresh_copy()
        return Tree(self._root, left_copy, right_copy)

    def create_fresh_copy_wrong(self):
        """Menggoda tetapi salah: hasil rekursif dibuang."""
        if self._left is not None:
            self._left.create_fresh_copy_wrong()        # hasilnya diabaikan!
        if self._right is not None:
            self._right.create_fresh_copy_wrong()       # hasilnya diabaikan!
        return Tree(self._root, self._left, self._right)  # berbagi subpohon-subpohon saya

    def mirror(self):
        if self._left is not None:
            self._left.mirror()
        if self._right is not None:
            self._right.mirror()
        self._left, self._right = self._right, self._left

    def nodes(self):
        """Setiap objek simpul pohon ini (dipakai untuk mendeteksi keberbagian)."""
        result = [self]
        if self._left is not None:
            result += self._left.nodes()
        if self._right is not None:
            result += self._right.nodes()
        return result

    def shares_node_with(self, other):
        mine = {id(n) for n in self.nodes()}
        return any(id(n) in mine for n in other.nodes())
```

Kode 8.16 (Lanjutan) *fresh_copy.py: salinan dalam di Python, dengan uji berbagi simpul yang sama.*

```
    def __str__(self):
        if self._left is None and self._right is None:
            return str(self._root)
        l = "-" if self._left is None else str(self._left)
        r = "-" if self._right is None else str(self._right)
        return f"{self._root}({l},{r})"


def sample():
    return Tree(3, Tree(4, Tree(6), Tree(7)), Tree(5, Tree(8), Tree(9)))


class TestFreshCopy(unittest.TestCase):
    def test_copy_survives_mirror(self):
        t = sample()
        u = t.create_fresh_copy()
        t.mirror()
        self.assertEqual("3(4(6,7),5(8,9))", str(u))
        self.assertFalse(t.shares_node_with(u))

    def test_wrong_copy_shares(self):
        t = sample()
        self.assertTrue(t.shares_node_with(t.create_fresh_copy_wrong()))


if __name__ == "__main__":
    t = sample()
    u = t.create_fresh_copy()
    w = t.create_fresh_copy_wrong()
    t.mirror()
    print("t after mirror  =", t)
    print("u (fresh copy)  =", u)
    print("w (wrong copy)  =", w)
    sys.stdout.flush()
    unittest.main(verbosity=2)
```

```
t after mirror  = 3(5(9,8),4(7,6))
u (fresh copy)  = 3(4(6,7),5(8,9))
w (wrong copy)  = 3(4(7,6),5(9,8))
test_copy_survives_mirror (__main__.TestFreshCopy.test_copy_survives_mirror) ... ok
test_wrong_copy_shares (__main__.TestFreshCopy.test_wrong_copy_shares) ... ok

----------------------------------------------------------------------
Ran 2 tests in 0.000s

OK
```

Baris demi baris. `createFreshCopy()` menyimpan kedua hasil delegasi dalam variabel lokal, `leftCopy` dan `rightCopy`, lalu meneruskannya ke konstruktor. Ekspresi kondisional `(left == null) ? null : left.createFreshCopy()` adalah kasus dasar sekaligus delegasi dalam satu baris. Banyaknya pernyataan `new` yang dieksekusi sama dengan banyaknya simpul, tujuh untuk pohon contoh, sesuai tuntutan soal. Pemeriksaannya memastikan kedua belahan spesifikasi: `u` tetap mencetak bentuk aslinya setelah `t.mirror()`, dan `t` serta `u` tidak berbagi satu objek simpul pun.

Penyelesaian keliru yang menggoda. `createFreshCopyWrong()` melakukan pemanggilan rekursif, yang tampak sesuai petunjuk, tetapi membuang hasilnya, lalu membangun `new Tree(root, left, right)` dari subpohon *asli*. Hanya akarnya yang baru. Hasil eksekusinya memperlihatkan kerusakannya: setelah `t2.mirror()`, salinan keliru itu mencetak `3(4(7,6),5(9,8))`. Akarnya masih memiliki 4 di kiri, karena objek akar yang baru tidak ikut

dicerminkan, tetapi semua yang di bawahnya sudah tercermin, karena itu adalah objek-objek yang sama dengan yang ada di `t2`. Inilah *salinan dangkal* (*shallow copy*), dan keluarannya yang setengah benar membuatnya menjebak: sekilas tampak "4 masih di kiri, berhasil". Uji berbagi simpul berkata lain. `copy.copy` di Python membuat salinan dangkal dan `copy.deepcopy` membuat salinan dalam; soal ini melarang pustaka agar Anda melihat sendiri apa arti "dalam".

> **UJI DIRI.** Tulislah `Tree mirroredCopy()` yang mengembalikan salinan segar yang *tercermin* dan membiarkan `this` tidak tersentuh. Apa yang dikembalikannya untuk pohon contoh?
>
> *Jawaban.* Tukar argumennya: `return new Tree(root, rightCopy, leftCopy)`, dengan salinan-salinan itu sendiri berupa `mirroredCopy()` dari subpohonnya. Untuk pohon contoh, hasilnya `3(5(9,8),4(7,6))`, dan `t` masih mencetak `3(4(6,7),5(8,9))`.

8.4.5 Soal 8.5 — Menyimpan penelusuran ke dalam larik

> **SOAL.** Include a recursive method `public int saveDepthFirst(int a[], int whereToStart)` in `Tree` that stores the depth-first traversal of the tree into the array `a`, beginning at position `whereToStart`, and returns the last position used. For the tree with root 4, children 5 and 6, and grandchildren 7, 8, 9, 10, the fragment `int a[] = new int[13]; int end = t.saveDepthFirst(a, 4);` must fill the array as `? ? ? ? 4 5 7 8 6 9 10 ? ?` and set `end` to 10.
>
> *Dengan kata lain:* tambahkan ke `Tree` metode rekursif `public int saveDepthFirst(int a[], int whereToStart)` yang menyimpan penelusuran *depth-first* ke larik `a` mulai dari posisi `whereToStart` dan mengembalikan posisi terakhir yang dipakai. Untuk pohon berakar 4 dengan anak 5 dan 6 serta cucu 7, 8, 9, 10, penggalan di atas harus mengisi larik menjadi `? ? ? ? 4 5 7 8 6 9 10 ? ?` dan membuat `end` bernilai 10.

Dengan bahasa kita sendiri. Ini Soal 8.3 lagi, hanya saja alih-alih mencetak setiap nilai, kita menuliskannya ke sel bebas berikutnya dari sebuah larik (Bab 5), dan metode harus memberi tahu pemanggilnya di mana ia berhenti, agar pemanggil tahu dari mana harus melanjutkan.

Rancangan. Kesulitannya ada pada pembukuan. Setiap pemanggilan menulis akarnya di `whereToStart`; subpohon kiri harus mulai satu sel sesudahnya; subpohon kanan harus mulai satu sel sesudah *sel terakhir yang dipakai subpohon kiri*, dan tidak ada yang mengetahui bilangan itu sebelumnya. Karena itulah metode mengembalikannya. **Tabel 8.6** menelusuri pemanggilannya. Variabel `last` selalu menyimpan posisi terakhir yang sudah ditulis.

Tabel 8.6 Pemanggilan-pemanggilan `saveDepthFirst` pada pohon contoh yang dimulai dari posisi 4. Setiap pemanggilan mengembalikan posisi terakhir yang ditulisnya; pemanggil menambahkan satu untuk menemukan sel bebas berikutnya.

Pemanggilan	Menulis	Pemanggilan kiri mulai di	Kiri mengembalikan	Pemanggilan kanan mulai di	Mengembalikan
simpul 4, mulai 4	`a[4]=4`	5	7	8	10
simpul 5, mulai 5	`a[5]=5`	6	6	7	7
daun 7, mulai 6	`a[6]=7`	—	—	—	6
daun 8, mulai 7	`a[7]=8`	—	—	—	7
simpul 6, mulai 8	`a[8]=6`	9	9	10	10
daun 9, mulai 9	`a[9]=9`	—	—	—	9
daun 10, mulai 10	`a[10]=10`	—	—	—	10

Konvensinya penting, jadi mari kita tetapkan dalam satu kalimat: *metode mengembalikan posisi terakhir yang ditulis, dan setiap pemanggil memulai potongan berikutnya pada posisi itu ditambah satu.* Bila sebuah pohon memiliki n simpul dan penelusuran dimulai di posisi s, posisi terakhir yang ditulis adalah

$$end = s + n - 1, \tag{8.2}$$

yang untuk pohon contoh menghasilkan $4 + 7 - 1 = 10$, sesuai tuntutan. Persamaan (8.2) juga merupakan uji yang berguna: apa pun bentuk pohonnya, hanya banyaknya simpul dan posisi awal yang menentukan nilai kembaliannya.

Penyelesaian Java. **Kode 8.17** mengimplementasikan konvensi itu, bersama sebuah varian yang keliru.

Kode 8.17 `SaveDepthFirst.java`: menulis penelusuran *pre-order* ke dalam larik dan mengembalikan posisi terakhir yang dipakai.

```java
// SaveDepthFirst.java - Soal 8.5: menyimpan penelusuran depth-first ke dalam larik.
import java.util.Arrays;

public class SaveDepthFirst {

    static final class Tree {
        private int root;
        private Tree left, right;

        Tree(int root, Tree left, Tree right) {
            this.root = root;
            this.left = left;
            this.right = right;
        }

        Tree(int root) {
            this(root, null, null);
        }

        /**
         * Menulis penelusuran depth-first saya ke a, mulai dari whereToStart.
         * Mengembalikan posisi TERAKHIR yang ditulis.
         */
        int saveDepthFirst(int[] a, int whereToStart) {
            a[whereToStart] = root;
            int last = whereToStart;                    // sejauh ini, hanya akar saya
            if (left != null) {
                last = left.saveDepthFirst(a, last + 1);
            }
            if (right != null) {
                last = right.saveDepthFirst(a, last + 1);
            }
            return last;
        }

        /** Menggoda tetapi salah: subpohon kanan mulai DI sel terakhir, bukan sesudahnya.
*/
        int saveDepthFirstWrong(int[] a, int whereToStart) {
            a[whereToStart] = root;
            int last = whereToStart;
            if (left != null) {
                last = left.saveDepthFirstWrong(a, last + 1);
            }
            if (right != null) {
                last = right.saveDepthFirstWrong(a, last);
            }
            return last;
```

Kode 8.17 (Lanjutan) *SaveDepthFirst.java: menulis penelusuran pre-order ke dalam larik dan mengembalikan posisi terakhir yang dipakai.*

```
        }
    }

    private static int passed = 0;

    private static void check(String name, Object expected, Object actual) {
        if (!java.util.Objects.equals(expected, actual)) {
            throw new AssertionError(name + ": expected " + expected + " but got " +
actual);
        }
        passed++;
        System.out.println("PASS " + name);
    }

    public static void main(String[] args) {
        Tree t = new Tree(4,
                new Tree(5, new Tree(7), new Tree(8)),
                new Tree(6, new Tree(9), new Tree(10)));
        int[] a = new int[13];                  // Java mengisi int[] baru dengan 0
        int end = t.saveDepthFirst(a, 4);
        System.out.println("a   = " + Arrays.toString(a));
        System.out.println("end = " + end);
        check("array", "[0, 0, 0, 0, 4, 5, 7, 8, 6, 9, 10, 0, 0]", Arrays.toString(a));
        check("end", 10, end);

        int[] b = new int[3];
        check("single leaf", 0, new Tree(42).saveDepthFirst(b, 0));

        int[] c = new int[13];
        int wrongEnd = t.saveDepthFirstWrong(c, 4);
        System.out.println("wrong a = " + Arrays.toString(c) + ", end = " + wrongEnd);
        System.out.println("All " + passed + " checks passed.");
    }
}
```

```
a   = [0, 0, 0, 0, 4, 5, 7, 8, 6, 9, 10, 0, 0]
end = 10
PASS array
PASS end
PASS single leaf
wrong a = [0, 0, 0, 0, 4, 5, 6, 10, 0, 0, 0, 0, 0], end = 7
All 3 checks passed.
```

Penyelesaian Python. **Kode 8.18** memakai sebuah *list* berisi tiga belas nol sebagai pengganti larik Java.

Kode 8.18 `save_depth_first.py`: penelusuran yang sama disimpan ke dalam *list* Python.

```
# save_depth_first.py - Soal 8.5: menyimpan penelusuran depth-first ke dalam larik.
import sys
import unittest


class Tree:
    def __init__(self, root, left=None, right=None):
        self._root = root
        self._left = left
        self._right = right

    def save_depth_first(self, a, where_to_start):
        """Menulis penelusuran depth-first saya ke a mulai dari where_to_start.
        Mengembalikan posisi TERAKHIR yang ditulis."""
        a[where_to_start] = self._root
        last = where_to_start                       # sejauh ini, hanya akar saya
        if self._left is not None:
            last = self._left.save_depth_first(a, last + 1)
        if self._right is not None:
            last = self._right.save_depth_first(a, last + 1)
        return last

    def save_depth_first_wrong(self, a, where_to_start):
        """Menggoda tetapi salah: subpohon kanan mulai DI sel terakhir."""
        a[where_to_start] = self._root
        last = where_to_start
        if self._left is not None:
            last = self._left.save_depth_first_wrong(a, last + 1)
        if self._right is not None:
            last = self._right.save_depth_first_wrong(a, last)
        return last


def sample():
    return Tree(4, Tree(5, Tree(7), Tree(8)), Tree(6, Tree(9), Tree(10)))


class TestSave(unittest.TestCase):
    def test_sample(self):
        a = [0] * 13
        end = sample().save_depth_first(a, 4)
        self.assertEqual([0, 0, 0, 0, 4, 5, 7, 8, 6, 9, 10, 0, 0], a)
        self.assertEqual(10, end)
```

Kode 8.18 (Lanjutan) *save_depth_first.py: penelusuran yang sama disimpan ke dalam list Python.*

```
    def test_single_leaf(self):
        self.assertEqual(0, Tree(42).save_depth_first([0] * 3, 0))


if __name__ == "__main__":
    a = [0] * 13                    # tiga belas nol, seperti new int[13] di Java
    end = sample().save_depth_first(a, 4)
    print("a   =", a)
    print("end =", end)
    c = [0] * 13
    wrong_end = sample().save_depth_first_wrong(c, 4)
    print(f"wrong a = {c}, end = {wrong_end}")
    sys.stdout.flush()
    unittest.main(verbosity=2)
```

```
a   = [0, 0, 0, 0, 4, 5, 7, 8, 6, 9, 10, 0, 0]
end = 10
wrong a = [0, 0, 0, 0, 4, 5, 6, 10, 0, 0, 0, 0, 0], end = 7
test_sample (__main__.TestSave.test_sample) ... ok
test_single_leaf (__main__.TestSave.test_single_leaf) ... ok

----------------------------------------------------------------------
Ran 2 tests in 0.000s

OK
```

Baris demi baris. Baris pertama metode menulis akarnya. Variabel lokal `last` diawali sama dengan `whereToStart`, karena sejauh ini baru akar yang ditulis. Bila ada subpohon kiri, ia mulai di `last + 1` dan jawabannya menggantikan `last`; hal yang sama terjadi untuk subpohon kanan. Metode mengembalikan `last`. Perhatikan isi sel-sel yang tidak terpakai. Pernyataan soal menuliskannya sebagai `?`, tetapi Java tidak membiarkan memori tak terdefinisi: `new int[13]` mengisi larik baru dengan nol (Bab 5), itulah sebabnya keluaran memperlihatkan `0` di posisi 0 sampai 3, 11, dan 12. Program Python meniru hal ini dengan `[0] * 13`. Pemeriksaan `single leaf` memastikan kasus terkecil dari Persamaan (8.2): satu simpul yang mulai di 0 berakhir di 0.

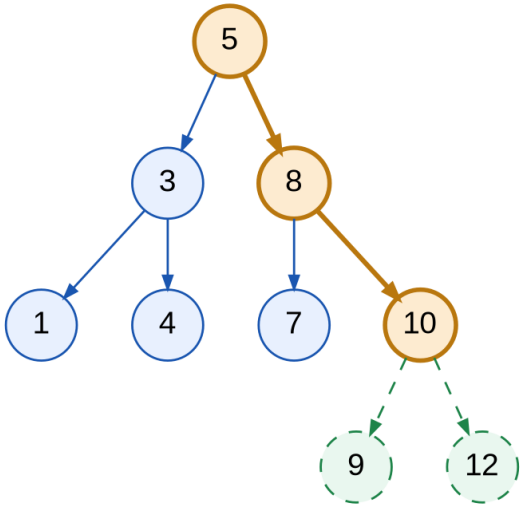
Penyelesaian keliru yang menggoda. Petunjuk dalam pernyataan asli berbicara tentang "posisi tersedia berikutnya", dan pemrogram yang mengikutinya setengah-setengah menulis pemanggilan kanan sebagai `right.saveDepthFirst(a, last)`, memulai subpohon kanan *di* sel terakhir yang ditulis, bukan sesudahnya. Keluaran memperlihatkan hasilnya: `[0, 0, 0, 0, 4, 5, 6, 10, 0, ...]` dengan `end = 7`. Daun 8 menimpa daun 7 di `a[6]`, simpul 6 menimpa daun 8, dan daun 10 menimpa daun 9; tiga nilai lenyap tanpa satu pun pesan galat. Mencampur dua konvensi ("kembalikan yang terakhir dipakai" dan "kembalikan yang bebas berikutnya") adalah sumber klasik galat selisih satu (*off-by-one*) ini. Pilih satu, tuliskan dalam komentar dokumentasi, dan buat setiap pemanggilan mematuhinya.

> **UJI DIRI.** Tulis ulang metode itu dengan konvensi yang lain: ia mengembalikan posisi *bebas berikutnya*. Apa yang dikembalikannya untuk pohon contoh yang dimulai di 4, dan seperti apa pemanggilan rekursifnya?
>
> *Jawaban.* Ia mengembalikan 11 ($s + n$). Simpul menulis `a[next] = root`, menetapkan `next = next + 1`, lalu `next = left.save(a, next)` dan `next = right.save(a, next)`, tanpa `+ 1` di mana pun dalam pemanggilan, dan mengembalikan `next`.

8.4.6 Soal 8.6 — Mencari nilai di pohon pencarian biner

> **SOAL.** Recall that in a *binary search tree* (BST) all nodes in the left branch are smaller than the root, all nodes in the right branch are greater, and both branches are themselves BSTs. Include a recursive method `public Tree find(int n)` in `Tree` that, assuming the tree is a BST, returns a reference to the subtree whose root equals `n`, or `null` if there is none. It must be efficient: it should follow only the single path down from the root, not search the whole tree. Hint: check your root; if the value is not there, decide whether to ask left or right, but not both.
>
> *Dengan kata lain:* pada *pohon pencarian biner* (BST), semua simpul di cabang kiri lebih kecil daripada akar, semua simpul di cabang kanan lebih besar, dan kedua cabang itu sendiri juga BST. Tambahkan ke `Tree` metode rekursif `public Tree find(int n)` yang, dengan anggapan pohonnya BST, mengembalikan referensi ke subpohon yang akarnya sama dengan `n`, atau `null` bila tidak ada. Metodenya harus efisien: hanya menyusuri satu lintasan dari akar ke bawah, bukan menggeledah seluruh pohon. Petunjuk: periksa akar Anda; bila nilainya tidak di sana, putuskan apakah bertanya ke kiri atau ke kanan, tetapi jangan keduanya.

Dengan bahasa kita sendiri. Aturan urutan berarti satu perbandingan di sebuah simpul sudah memberi tahu kita separuh mana yang tidak mungkin memuat `n`. Kita lalu melanjutkan hanya di separuh yang lain. **Gambar 8.6** memperlihatkan pohon uji beserta lintasan yang ditempuh `find(8)`: akar 5, lalu 8, ketemu, setelah dua kunjungan.



Gambar 8.6 Pohon pencarian biner yang dipakai pada Soal 8.6 dan 8.7. Lintasan tebal ditempuh oleh `find(8)` dan `find(10)`; daun putus-putus 9 dan 12 ditambahkan oleh `insert(9)` dan `insert(12)`.

Rancangan. **Tabel 8.7** mendaftar keempat keadaan yang mungkin di sebuah simpul.

Tabel 8.7 Kasus-kasus `find(n)` pada BST. Hanya satu dari kedua subpohon yang pernah ditanya.

Keadaan di sebuah simpul	Tindakan
`n == root`	ketemu: `return this`
`n < root`, subpohon kiri ada	`return left.find(n)`
`n > root`, subpohon kanan ada	`return right.find(n)`
subpohon yang diperlukan tidak ada	tidak ada di pohon: `return null`

Setiap pemanggilan berhenti atau turun satu tingkat, sehingga banyaknya simpul yang dikunjungi paling banyak sama dengan *tinggi* h pohon, yaitu banyaknya simpul pada lintasan terpanjang dari akar ke sebuah daun. Pohon seimbang dengan n simpul memiliki tinggi sekitar $\log_2 n$, sehingga

$$\text{visits(find)} \le h, \qquad h \approx \log_2 n \text{ untuk pohon seimbang,} \tag{8.3}$$

yang untuk sejuta simpul berarti sekitar 20 kunjungan, bukan sejuta.

Penyelesaian Java. **Kode 8.19** menghitung kunjungan dalam sebuah *field* `static` agar keluarannya dapat memperlihatkan lintasan tunggal itu.

Kode 8.19 `BstFind.java`: mencari di BST melalui satu lintasan, dengan penghitung kunjungan dan varian keliru yang berakhir *crash*.

```java
// BstFind.java - Soal 8.6: mencari nilai di pohon pencarian biner melalui SATU lintasan.
public class BstFind {

    static final class Tree {
        private int root;
        private Tree left, right;
        static int visits = 0;                // banyaknya simpul yang dilihat find()

        Tree(int root, Tree left, Tree right) {
            this.root = root;
            this.left = left;
            this.right = right;
        }

        Tree(int root) {
            this(root, null, null);
        }

        /** Mengembalikan subpohon yang akarnya n, atau null bila n tidak ada di BST ini.
*/
        Tree find(int n) {
            visits++;
            if (n == root) {
                return this;
            } else if (n < root) {
                return (left == null) ? null : left.find(n);
            } else {
                return (right == null) ? null : right.find(n);
            }
        }

        /** Menggoda tetapi salah: melangkah ke subpohon yang tidak ada. */
        Tree findWrong(int n) {
            if (n == root) {
                return this;
            } else if (n < root) {
                return left.findWrong(n);
            } else {
                return right.findWrong(n);
            }
        }
```

Kode 8.19 (Lanjutan) *BstFind.java: mencari di BST melalui satu lintasan, dengan penghitung kunjungan dan varian keliru yang berakhir crash.*

```
        @Override
        public String toString() {
            if (left == null && right == null) {
                return Integer.toString(root);
            }
            String l = (left == null) ? "-" : left.toString();
            String r = (right == null) ? "-" : right.toString();
            return root + "(" + l + "," + r + ")";
        }
    }

    private static int passed = 0;

    private static void check(String name, Object expected, Object actual) {
        if (!java.util.Objects.equals(expected, actual)) {
            throw new AssertionError(name + ": expected " + expected + " but got " +
actual);
        }
        passed++;
        System.out.println("PASS " + name);
    }

    public static void main(String[] args) {
        Tree t = new Tree(5,
                new Tree(3, new Tree(1), new Tree(4)),
                new Tree(8, new Tree(7), new Tree(10)));
        for (int n : new int[] {8, 4, 5, 6}) {
            Tree.visits = 0;
            Tree found = t.find(n);
            System.out.println("find(" + n + ") = " + found + ", nodes visited: " +
Tree.visits);
        }
        check("find 8", "8(7,10)", String.valueOf(t.find(8)));
        check("find 1", "1", String.valueOf(t.find(1)));
        check("find 6 is null", null, t.find(6));
        check("find 11 is null", null, t.find(11));
        try {
            t.findWrong(6);
        } catch (NullPointerException e) {
            System.out.println("findWrong(6) crashed: NullPointerException");
        }
        System.out.println("All " + passed + " checks passed.");
    }
}
```

```
find(8) = 8(7,10), nodes visited: 2
find(4) = 4, nodes visited: 3
find(5) = 5(3(1,4),8(7,10)), nodes visited: 1
find(6) = null, nodes visited: 3
PASS find 8
PASS find 1
PASS find 6 is null
PASS find 11 is null
findWrong(6) crashed: NullPointerException
All 4 checks passed.
```

Penyelesaian Python. **Kode 8.20** memakai *atribut kelas* `Tree.visits`, yang dipakai bersama oleh semua simpul, sebagai penghitung.

Kode 8.20 `bst_find.py`: pencarian yang sama dalam Python.

```python
# bst_find.py - Soal 8.6: mencari nilai di pohon pencarian biner melalui SATU lintasan.
import sys
import unittest


class Tree:
    visits = 0                      # banyaknya simpul yang dilihat find()

    def __init__(self, root, left=None, right=None):
        self._root = root
        self._left = left
        self._right = right

    def find(self, n):
        """Mengembalikan subpohon yang akarnya n, atau None bila n tidak ada di BST ini."""
        Tree.visits += 1
        if n == self._root:
            return self
        elif n < self._root:
            return None if self._left is None else self._left.find(n)
        else:
            return None if self._right is None else self._right.find(n)

    def find_wrong(self, n):
        """Menggoda tetapi salah: melangkah ke subpohon yang tidak ada."""
        if n == self._root:
            return self
        elif n < self._root:
            return self._left.find_wrong(n)
        else:
            return self._right.find_wrong(n)

    def __str__(self):
        if self._left is None and self._right is None:
            return str(self._root)
        l = "-" if self._left is None else str(self._left)
        r = "-" if self._right is None else str(self._right)
        return f"{self._root}({l},{r})"


def sample():
    return Tree(5, Tree(3, Tree(1), Tree(4)), Tree(8, Tree(7), Tree(10)))
```

Kode 8.20 (Lanjutan) *bst_find.py: pencarian yang sama dalam Python.*

```
class TestFind(unittest.TestCase):
    def test_present(self):
        self.assertEqual("8(7,10)", str(sample().find(8)))
        self.assertEqual("1", str(sample().find(1)))

    def test_absent(self):
        self.assertIsNone(sample().find(6))
        self.assertIsNone(sample().find(11))


if __name__ == "__main__":
    t = sample()
    for n in [8, 4, 5, 6]:
        Tree.visits = 0
        found = t.find(n)
        print(f"find({n}) = {found}, nodes visited: {Tree.visits}")
    try:
        t.find_wrong(6)
    except AttributeError as e:
        print("find_wrong(6) crashed:", e)
    sys.stdout.flush()
    unittest.main(verbosity=2)
```

```
find(8) = 8(7,10), nodes visited: 2
find(4) = 4, nodes visited: 3
find(5) = 5(3(1,4),8(7,10)), nodes visited: 1
find(6) = None, nodes visited: 3
find_wrong(6) crashed: 'NoneType' object has no attribute 'find_wrong'
test_absent (__main__.TestFind.test_absent) ... ok
test_present (__main__.TestFind.test_present) ... ok

----------------------------------------------------------------------
Ran 2 tests in 0.000s

OK
```

Baris demi baris. Keluarannya menegaskan Persamaan (8.3): `find(5)` berhenti di akar setelah satu kunjungan, `find(8)` memerlukan dua, `find(4)` tiga (5, 3, 4), dan `find(6)` yang tidak berhasil juga tiga (5, 8, 7, dan 7 tidak memiliki subpohon kiri). Tinggi pohon ini 3, dan tidak ada pencarian yang mengunjungi lebih banyak simpul. Metode ini mengembalikan *subpohon*, bukan sekadar jawaban ya atau tidak, sehingga pemanggil dapat melanjutkan pekerjaan dari sana; mencetaknya menghasilkan `8(7,10)`. Penghitung `visits` bersifat `static` karena ia milik kelas, bukan milik satu simpul tertentu; ia alat bantu pengujian, dan kelas untuk produksi tidak akan menyimpannya.

Penyelesaian keliru yang menggoda. `findWrong` menghilangkan pemeriksaan untuk subpohon yang tidak ada. Untuk nilai yang ada di pohon, metode ini bekerja, sehingga mudah dipercaya. Untuk `find(6)` ia menempuh 5, 8, 7, lalu memanggil `left.findWrong(6)` pada subpohon kiri 7 yang tidak ada, dan *crash* dengan `NullPointerException` (Python: `AttributeError`). Kesalahan menggoda kedua, yang tidak diperlihatkan, menggeledah *kedua* subpohon "supaya aman": jawabannya selalu benar tetapi ia mengunjungi setiap simpul, persis hal yang dilarang soal. Uji yang menghitung kunjungan, seperti uji kita, adalah satu-satunya jenis uji yang dapat menangkapnya.

> **UJI DIRI.** Pada pohon **Gambar 8.6** (tanpa 9 dan 12), simpul mana saja yang dikunjungi `find(11)`, dan apa yang dikembalikannya? Bagaimana dengan `find(1)`?
>
> *Jawaban.* `find(11)`: 5, 8, 10, lalu 10 tidak memiliki subpohon kanan, sehingga ia mengembalikan `null` setelah 3 kunjungan. `find(1)`: 5, 3, 1, ketemu setelah 3 kunjungan; ia mengembalikan daun `1`.

8.4.7 Soal 8.7 — Menyisipkan nilai ke pohon pencarian biner

> **SOAL.** Using the BST of Problem 8.6, include a recursive method `public Tree insert(int n)` in `Tree` that inserts a new node in the correct position, which will be a leaf. Return the new tree if you succeed, and the current tree if the element is already there and cannot be inserted. Hint: modify your solution to Problem 8.6. If you find the element, it cannot be inserted. If not, you reach a node whose needed subtree is missing, and the new element becomes that child.
>
> *Dengan kata lain:* dengan BST dari Soal 8.6, tambahkan ke `Tree` metode rekursif `public Tree insert(int n)` yang menyisipkan simpul baru di posisi yang benar, yang pasti berupa daun. Kembalikan pohon yang baru bila berhasil, dan pohon yang sekarang bila elemennya sudah ada sehingga tidak dapat disisipkan. Petunjuk: ubahlah penyelesaian Soal 8.6. Bila elemennya ditemukan, ia tidak dapat disisipkan. Bila tidak, Anda tiba di simpul yang subpohon yang diperlukannya tidak ada, dan elemen baru menjadi anak di sana.

Dengan bahasa kita sendiri. Turunlah persis seperti `find`. Bila kita bertemu `n`, berhenti dan jangan mengubah apa pun. Bila subpohon yang kita perlukan tidak ada, buatlah daunnya *dan pasangkan* di sana. Aturan urutan terjaga dengan sendirinya, karena daun baru itu terletak persis di tempat `find` akan mencarinya.

Rancangan. **Tabel 8.8** membandingkan `find` dan `insert` kasus demi kasus; hanya baris terakhir yang berbeda.

Tabel 8.8 `insert` adalah `find` dengan kasus terakhir yang berbeda: di tempat pencarian akan menyerah, daun baru dipasangkan.

Keadaan	`find(n)`	`insert(n)`
`n == root`	kembalikan `this`	tidak ada yang dikerjakan; kembalikan `this`
`n < root`, kiri ada	`left.find(n)`	`left.insert(n)`
`n > root`, kanan ada	`right.find(n)`	`right.insert(n)`
subpohon yang diperlukan tidak ada	kembalikan `null`	`left = new Tree(n)` atau `right = new Tree(n)`

Penyelesaian Java. **Kode 8.21** juga menambahkan pencetak *in-order* (kiri, simpul, kanan), yang untuk BST yang benar harus mendaftar nilai-nilainya dalam urutan menaik.

Kode 8.21 `BstInsert.java`: menyisipkan ke BST dengan memasang daun baru, disertai pemeriksaan *in-order*.

```java
// BstInsert.java - Soal 8.7: menyisipkan nilai ke pohon pencarian biner sebagai daun baru.
public class BstInsert {

    static final class Tree {
        private int root;
        private Tree left, right;

        Tree(int root, Tree left, Tree right) {
            this.root = root;
            this.left = left;
            this.right = right;
        }

        Tree(int root) {
            this(root, null, null);
        }

        /**
         * Menyisipkan n sebagai daun baru di tempat yang benar dan mengembalikan pohon
ini.
         * Bila n sudah ada, tidak ada yang berubah dan pohon ini dikembalikan.
         */
        Tree insert(int n) {
            if (n < root) {
                if (left == null) {
                    left = new Tree(n);          // pasang daun baru di sini
                } else {
                    left.insert(n);
                }
            } else if (n > root) {
                if (right == null) {
                    right = new Tree(n);
                } else {
                    right.insert(n);
                }
            }
            return this;                         // n == root: sudah ada
        }
```

Kode 8.21 (Lanjutan) *BstInsert.java: menyisipkan ke BST dengan memasang daun baru, disertai pemeriksaan in-order.*

```
        /** Menggoda tetapi salah: membuat daun tetapi tidak pernah memasangnya. */
        Tree insertWrong(int n) {
            if (n < root) {
                return (left == null) ? new Tree(n) : left.insertWrong(n);
            } else if (n > root) {
                return (right == null) ? new Tree(n) : right.insertWrong(n);
            }
            return this;
        }

        /** In-order: kiri, akar, kanan. Untuk BST, ini urutan menaik. */
        String inOrder() {
            String l = (left == null) ? "" : left.inOrder();
            String r = (right == null) ? "" : right.inOrder();
            return l + root + " " + r;
        }

        @Override
        public String toString() {
            if (left == null && right == null) {
                return Integer.toString(root);
            }
            String l = (left == null) ? "-" : left.toString();
            String r = (right == null) ? "-" : right.toString();
            return root + "(" + l + "," + r + ")";
        }
    }

    static Tree sample() {
        return new Tree(5,
                new Tree(3, new Tree(1), new Tree(4)),
                new Tree(8, new Tree(7), new Tree(10)));
    }

    private static int passed = 0;

    private static void check(String name, Object expected, Object actual) {
        if (!java.util.Objects.equals(expected, actual)) {
            throw new AssertionError(name + ": expected " + expected + " but got " +
actual);
        }
        passed++;
        System.out.println("PASS " + name);
    }
```

Kode 8.21 (Lanjutan) *BstInsert.java: menyisipkan ke BST dengan memasang daun baru, disertai pemeriksaan in-order.*

```
    public static void main(String[] args) {
        Tree t = sample();
        t = t.insert(9);
        System.out.println("after insert(9) : " + t);
        t = t.insert(12);
        System.out.println("after insert(12): " + t);
        Tree before = t;
        t = t.insert(8);
        System.out.println("after insert(8) : " + t);
        check("shape", "5(3(1,4),8(7,10(9,12)))", t.toString());
        check("in-order is sorted", "1 3 4 5 7 8 9 10 12 ", t.inOrder());
        check("duplicate returns same object", true, before == t);

        Tree w = sample();
        w = w.insertWrong(9);
        System.out.println("wrong insert(9) : " + w);
        System.out.println("All " + passed + " checks passed.");
    }
}
```

```
after insert(9) : 5(3(1,4),8(7,10(9,-)))
after insert(12): 5(3(1,4),8(7,10(9,12)))
after insert(8) : 5(3(1,4),8(7,10(9,12)))
PASS shape
PASS in-order is sorted
PASS duplicate returns same object
wrong insert(9) : 9
All 3 checks passed.
```

Penyelesaian Python. **Kode 8.22** sama baris demi baris; `assertIs` memeriksa identitas objek, padanan Python untuk `==` Java pada referensi.

Kode 8.22 `bst_insert.py`: penyisipan BST dalam Python.

```python
# bst_insert.py - Soal 8.7: menyisipkan nilai ke pohon pencarian biner sebagai daun baru.
import sys
import unittest


class Tree:
    def __init__(self, root, left=None, right=None):
        self._root = root
        self._left = left
        self._right = right

    def insert(self, n):
        """Menyisipkan n sebagai daun baru di tempat yang benar dan mengembalikan pohon
ini.
        Bila n sudah ada, tidak ada yang berubah dan pohon ini dikembalikan."""
        if n < self._root:
            if self._left is None:
                self._left = Tree(n)             # pasang daun baru di sini
            else:
                self._left.insert(n)
        elif n > self._root:
            if self._right is None:
                self._right = Tree(n)
            else:
                self._right.insert(n)
        return self                              # n == root: sudah ada

    def insert_wrong(self, n):
        """Menggoda tetapi salah: membuat daun tetapi tidak pernah memasangnya."""
        if n < self._root:
            return Tree(n) if self._left is None else self._left.insert_wrong(n)
        elif n > self._root:
            return Tree(n) if self._right is None else self._right.insert_wrong(n)
        return self

    def in_order(self):
        """In-order: kiri, akar, kanan. Untuk BST, ini urutan menaik."""
        l = "" if self._left is None else self._left.in_order()
        r = "" if self._right is None else self._right.in_order()
        return l + f"{self._root} " + r
```

Kode 8.22 (Lanjutan) *bst_insert.py: penyisipan BST dalam Python.*

```
    def __str__(self):
        if self._left is None and self._right is None:
            return str(self._root)
        l = "-" if self._left is None else str(self._left)
        r = "-" if self._right is None else str(self._right)
        return f"{self._root}({l},{r})"


def sample():
    return Tree(5, Tree(3, Tree(1), Tree(4)), Tree(8, Tree(7), Tree(10)))


class TestInsert(unittest.TestCase):
    def test_inserts(self):
        t = sample().insert(9).insert(12)
        self.assertEqual("5(3(1,4),8(7,10(9,12)))", str(t))
        self.assertEqual("1 3 4 5 7 8 9 10 12 ", t.in_order())

    def test_duplicate_returns_same_object(self):
        t = sample()
        self.assertIs(t, t.insert(8))
        self.assertEqual("5(3(1,4),8(7,10))", str(t))


if __name__ == "__main__":
    t = sample()
    t = t.insert(9)
    print("after insert(9) :", t)
    t = t.insert(12)
    print("after insert(12):", t)
    t = t.insert(8)
    print("after insert(8) :", t)
    w = sample().insert_wrong(9)
    print("wrong insert(9) :", w)
    sys.stdout.flush()
    unittest.main(verbosity=2)
```

```
after insert(9) : 5(3(1,4),8(7,10(9,-)))
after insert(12): 5(3(1,4),8(7,10(9,12)))
after insert(8) : 5(3(1,4),8(7,10(9,12)))
wrong insert(9) : 9
test_duplicate_returns_same_object (__main__.TestInsert.test_duplicate_returns_same_object)
... ok
test_inserts (__main__.TestInsert.test_inserts) ... ok

----------------------------------------------------------------------
Ran 2 tests in 0.000s

OK
```

Baris demi baris. `insert(9)` menempuh 5, 8, 10; 9 lebih kecil daripada 10 dan 10 tidak memiliki subpohon kiri, sehingga `left = new Tree(9)` dijalankan di dalam simpul 10. `insert(12)` menempuh lintasan yang sama dan memasang 12 di kanan 10. `insert(8)` berhenti di simpul 8 dan tidak mengubah apa pun. Dalam setiap kasus metode mengembalikan `this`, akar seluruh pohon, sehingga pola pemanggilan `t = t.insert(9)` dari pernyataan soal berjalan. Tiga pemeriksaan mencakup spesifikasinya: bentuk yang persis, daftar *in-order* `1 3 4 5 7 8 9 10 12` (yang membuktikan aturan BST masih berlaku di mana-mana), dan kenyataan bahwa menyisipkan duplikat mengembalikan objek yang persis sama (`before == t`).

Penyelesaian keliru yang menggoda. `insertWrong` menciptakan daun dengan `new Tree(n)` dan *mengembalikannya*, tetapi tidak pernah menyimpannya di `left` atau `right`. Simpul baru itu tidak

terpasang ke mana pun. Lebih buruk lagi, karena metode mengembalikan daun baru alih-alih akar, pernyataan `w = w.insertWrong(9)` mengganti seluruh pohon milik pemanggil dengan satu simpul 9, seperti diperlihatkan baris keluaran `wrong insert(9) : 9`. Menciptakan objek tidak sama dengan menyambungkannya; objek yang tidak dirujuk oleh *field* mana pun hanyalah sampah, yang akan diambil kembali oleh *garbage collector* Java (atau Python).

Frasa "kembalikan pohon yang baru" dalam soal membuka kemungkinan rancangan benar yang kedua: `insert` tidak mengubah simpul lama mana pun, melainkan membangun simpul-simpul baru di sepanjang lintasan pencarian dan berbagi subpohon yang tidak tersentuh dengan pohon lama. Versi *persisten* itu adalah Latihan 8.11.

> **UJI DIRI.** Mulai dari pohon **Gambar 8.6** tanpa 9 dan 12, sisipkan 6 lalu 2. Ke mana masing-masing pergi, dan apa yang dicetak `toString()` sesudahnya?
>
> *Jawaban.* 6: lintasan 5, 8, 7; 6 < 7 dan 7 tidak memiliki anak kiri, sehingga 6 menjadi anak kiri 7. 2: lintasan 5, 3, 1; 2 > 1, sehingga 2 menjadi anak kanan 1. Hasilnya: `5(3(1(-,2),4),8(7(6,-),10))`.

8.5 Soal-Soal tentang *List* Rekursif

Kedelapan soal dalam subbab ini adalah metode `static` yang menerima *list* buatan `List` (**Kode 8.3**) dan, bila jawabannya berupa *list*, mengembalikan *list* yang *baru*; tidak ada *list* yang pernah diubah. Setiap program Java dikompilasi bersama `List.java` dan `ListException.java` (`javac *.java` di folder bab ini mengerjakan semuanya sekaligus), dan setiap program Python mengimpor `cons_list`. Soal meminta agar hanya operasi-operasi `List` dan `ListOps` yang dipakai, sehingga tidak ada gelung dan tidak ada *list* dari pustaka Java atau Python: setiap soal diselesaikan dengan `empty()`, `head()`, `tail()`, `nil()`, `cons()`, dan rekursi.

Setiap metode *list* di bawah ini memiliki kerangka yang sama, yaitu resep delegasi dari Subbab 8.2 yang dikhususkan untuk *list*:

$$f(a) = \begin{cases} \text{jawaban untuk list kosong} & \text{jika } a = \text{nil}, \\ \text{gabung}(\text{head}(a),\ f(\text{tail}(a))) & \text{selainnya}. \end{cases} \tag{8.4}$$

Menyelesaikan sebuah soal berarti mengisi kedua bagian kosong pada Persamaan (8.4). Karena setiap pemanggilan membuang satu elemen, *list* dengan n elemen menimbulkan $n+1$ pemanggilan, dan waktu eksekusinya sebanding dengan n selama "gabung" memakan waktu konstan.

8.5.1 Soal 8.8 — Menegasikan sebuah *list*

> **SOAL.** Include a recursive method `static List negateAll(List a)` that returns a new list with all the elements of `a` with their sign negated: positive integers become negative and negative integers become positive. Example: `[2, -5, 8, 0]` gives `[-2, 5, -8, 0]`.
>
> *Dengan kata lain:* tulislah metode rekursif `static List negateAll(List a)` yang mengembalikan *list* baru berisi semua elemen `a` dengan tanda yang dibalik: bilangan positif menjadi negatif dan bilangan negatif menjadi positif. Contoh: `[2, -5, 8, 0]` menghasilkan `[-2, 5, -8, 0]`.

Dengan bahasa kita sendiri. Bangunlah *list* dengan panjang yang sama yang elemen ke-i-nya adalah minus elemen ke-i dari `a`. *List* aslinya harus tetap seperti semula, dan hal itu memang sudah dijamin oleh `List` yang imutabel.

Rancangan. **Tabel 8.9** mengisi bagian kosong pada Persamaan (8.4).

Tabel 8.9 Rancangan `negateAll`: sebuah *map* atas *list*, satu elemen keluaran untuk setiap elemen masukan.

Bagian	Jawaban untuk `negateAll`
List kosong	`nil()`
Gabung	`cons(-head, negateAll(tail))`
Kasus tepi yang diuji	*list* kosong; menerapkannya dua kali menghasilkan *list* semula

Kode 8.23 `NegateAll.java`: menegasikan setiap elemen secara rekursif.

```
// NegateAll.java - Soal 8.8: list baru dengan setiap tanda dibalik.
public class NegateAll {

    /** [2, -5, 8, 0] menjadi [-2, 5, -8, 0]; list a sendiri tidak tersentuh. */
    static List negateAll(List a) {
        if (a.empty()) {
            return List.nil();                          // kasus dasar
        }
        return List.cons(-a.head(), negateAll(a.tail()));  // kasus rekursif
    }

    /** Menggoda tetapi salah: menegasikan head dan melupakan sisanya. */
    static List negateAllWrong(List a) {
        if (a.empty()) {
            return List.nil();
        }
        return List.cons(-a.head(), a.tail());
    }

    private static int passed = 0;

    private static void check(String name, Object expected, Object actual) {
        if (!java.util.Objects.equals(expected, actual)) {
            throw new AssertionError(name + ": expected " + expected + " but got " +
actual);
        }
        passed++;
        System.out.println("PASS " + name);
    }

    public static void main(String[] args) {
        List a = List.of(2, -5, 8, 0);
        System.out.println("a                 = " + a);
        System.out.println("negateAll(a)      = " + negateAll(a));
        System.out.println("negateAllWrong(a) = " + negateAllWrong(a));
        check("sample", "[-2, 5, -8, 0]", negateAll(a).toString());
        check("a untouched", "[2, -5, 8, 0]", a.toString());
        check("empty", "[]", negateAll(List.nil()).toString());
        check("twice is identity", a.toString(), negateAll(negateAll(a)).toString());
        System.out.println("negate MIN_VALUE  = " + negateAll(List.of(Integer.MIN_VALUE)));
        System.out.println("All " + passed + " checks passed.");
    }
}
```

```
a                = [2, -5, 8, 0]
negateAll(a)     = [-2, 5, -8, 0]
negateAllWrong(a) = [-2, -5, 8, 0]
PASS sample
PASS a untouched
PASS empty
PASS twice is identity
negate MIN_VALUE  = [-2147483648]
All 4 checks passed.
```

Kode 8.24 `negate_all.py`: fungsi yang sama dalam Python.

```python
# negate_all.py - Soal 8.8: list baru dengan setiap tanda dibalik.
import sys
import unittest
from cons_list import List


def negate_all(a):
    """[2, -5, 8, 0] menjadi [-2, 5, -8, 0]; list a sendiri tidak tersentuh."""
    if a.empty():
        return List.nil()                              # kasus dasar
    return List.cons(-a.head(), negate_all(a.tail()))   # kasus rekursif


def negate_all_wrong(a):
    """Menggoda tetapi salah: menegasikan head dan melupakan sisanya."""
    if a.empty():
        return List.nil()
    return List.cons(-a.head(), a.tail())


class TestNegateAll(unittest.TestCase):
    def test_sample(self):
        a = List.of(2, -5, 8, 0)
        self.assertEqual("[-2, 5, -8, 0]", str(negate_all(a)))
        self.assertEqual("[2, -5, 8, 0]", str(a))

    def test_empty(self):
        self.assertEqual("[]", str(negate_all(List.nil())))

    def test_twice_is_identity(self):
        a = List.of(2, -5, 8, 0)
        self.assertEqual(str(a), str(negate_all(negate_all(a))))


if __name__ == "__main__":
    a = List.of(2, -5, 8, 0)
    print("a                  =", a)
    print("negate_all(a)      =", negate_all(a))
    print("negate_all_wrong(a) =", negate_all_wrong(a))
    print("negate -2147483648  =", negate_all(List.of(-2147483648)))
    sys.stdout.flush()
    unittest.main(verbosity=2)
```

```
a                   = [2, -5, 8, 0]
negate_all(a)       = [-2, 5, -8, 0]
negate_all_wrong(a) = [-2, -5, 8, 0]
negate -2147483648  = [2147483648]
test_empty (__main__.TestNegateAll.test_empty) ... ok
test_sample (__main__.TestNegateAll.test_sample) ... ok
test_twice_is_identity (__main__.TestNegateAll.test_twice_is_identity) ... ok

----------------------------------------------------------------------
Ran 3 tests in 0.000s

OK
```

Baris demi baris. **Kode 8.23** dan **Kode 8.24** masing-masing hanya tiga baris setelah komentarnya dibuang: kasus dasar, dan satu `cons` yang *head*-nya adalah *head* yang dinegasikan dan yang *tail*-nya adalah jawaban rekursif untuk *tail*. Pemeriksaan `twice is identity` adalah *uji sifat* (*property test*): untuk *list* apa pun, menegasikan dua kali harus menghasilkan *list* semula, dan menguji hukum semacam itu, di samping contoh-contoh tunggal, adalah kebiasaan yang baik. Baris demonstrasi terakhir menyimpan kejutan bagi pemrogram Java. `int` terkecil, $-2^{31} = -2147483648$, tidak memiliki pasangan positif dalam 32 bit (`int` terbesar adalah $2^{31} - 1$), sehingga `-a.head()` mengalami *overflow* (luapan) dan mengembalikan -2147483648 tanpa berubah. Bilangan bulat Python tidak berukuran tetap, sehingga Python mencetak `[2147483648]`. Soal tidak menyebut kasus ini, tetapi pemrogram yang cermat memperhatikannya dan mendokumentasikannya.

Penyelesaian keliru yang menggoda. `negateAllWrong` menegasikan *head* lalu memasang *tail* yang asli, `cons(-a.head(), a.tail())`, lupa melakukan rekursi. Hasilnya `[-2, -5, 8, 0]`: hanya elemen pertama yang berubah. Uji dengan *list* satu elemen akan lolos; uji pada contoh empat elemen gagal.

> **UJI DIRI.** Tulislah `static List squares(List a)`. Apa yang dikembalikannya untuk `[3, -1, 0]`?
>
> *Jawaban.* Bentuknya sama, dengan `cons(a.head() * a.head(), squares(a.tail()))`; hasilnya `[9, 1, 0]`.

8.5.2 Soal 8.9 — Menemukan posisi sebuah elemen

> **SOAL.** Include a recursive method `static int find(int x, List a)` that returns the position of the first occurrence of `x` in `a`, counting positions from 0. If `x` does not occur, return `-1`. Examples: `x = 3`, `a = [7, 5, 3, 8]` gives 2; `x = 2` gives `-1`.
>
> *Dengan kata lain:* tulislah metode rekursif `static int find(int x, List a)` yang mengembalikan posisi kemunculan pertama `x` di dalam `a`, dengan posisi dihitung mulai dari 0. Bila `x` tidak muncul, kembalikan `-1`. Contoh: `x = 3`, `a = [7, 5, 3, 8]` menghasilkan 2; `x = 2` menghasilkan `-1`.

Dengan bahasa kita sendiri. Bila *head*-nya `x`, jawabannya 0. Selain itu, jawabannya satu lebih besar daripada posisi `x` di *tail*, *asalkan* `x` memang ada di *tail*.

Rancangan. Titik halusnya adalah bahwa jawaban rekursif dapat berupa posisi sungguhan atau sinyal `-1`, dan keduanya harus diperlakukan secara berbeda (**Tabel 8.10**).

Tabel 8.10 Rancangan `find`: posisi yang ditemukan di *tail* digeser satu; sinyal "tidak ditemukan" tidak.

Keadaan	Jawaban
`a` kosong	`-1`
`head(a) == x`	`0`
`find(x, tail)` mengembalikan `-1`	`-1` (teruskan sinyal itu tanpa diubah)
`find(x, tail)` mengembalikan $p \geq 0$	$p + 1$

Kode 8.25 (Java) dan **Kode 8.26** (Python) memperlakukan sinyal itu persis seperti kata tabel; uji-ujinya tertanam di setiap program.

Kode 8.25 `ListFind.java`: posisi kemunculan pertama, atau `-1`.

```java
// ListFind.java - Soal 8.9: posisi kemunculan pertama x, atau -1.
public class ListFind {

    /** Posisi dihitung 0, 1, 2, ...; -1 berarti "x tidak muncul di a". */
    static int find(int x, List a) {
        if (a.empty()) {
            return -1;                          // tak ada lagi yang dicari
        }
        if (a.head() == x) {
            return 0;                           // ditemukan tepat di sini
        }
        int inTail = find(x, a.tail());         // tanyakan pada sisa list
        return (inTail == -1) ? -1 : inTail + 1;
    }

    /** Menggoda tetapi salah: menambahkan 1 bahkan pada jawaban "tidak ditemukan". */
    static int findWrong(int x, List a) {
        if (a.empty()) {
            return -1;
        }
        if (a.head() == x) {
            return 0;
        }
        return 1 + findWrong(x, a.tail());
    }

    private static int passed = 0;

    private static void check(String name, Object expected, Object actual) {
        if (!java.util.Objects.equals(expected, actual)) {
            throw new AssertionError(name + ": expected " + expected + " but got " +
actual);
        }
        passed++;
        System.out.println("PASS " + name);
    }
```

Kode 8.25 (Lanjutan) *ListFind.java: posisi kemunculan pertama, atau -1.*

```
    public static void main(String[] args) {
        List a = List.of(7, 5, 3, 8);
        System.out.println("a = " + a);
        System.out.println("find(3, a)      = " + find(3, a));
        System.out.println("find(2, a)      = " + find(2, a));
        System.out.println("findWrong(2, a) = " + findWrong(2, a));
        check("present", 2, find(3, a));
        check("absent", -1, find(2, a));
        check("first element", 0, find(7, a));
        check("last element", 3, find(8, a));
        check("first of duplicates", 1, find(5, List.of(9, 5, 5)));
        check("empty list", -1, find(1, List.nil()));
        System.out.println("All " + passed + " checks passed.");
    }
}
```

```
a = [7, 5, 3, 8]
find(3, a)      = 2
find(2, a)      = -1
findWrong(2, a) = 3
PASS present
PASS absent
PASS first element
PASS last element
PASS first of duplicates
PASS empty list
All 6 checks passed.
```

Kode 8.26 `list_find.py`: pencarian yang sama dalam Python.

```
# list_find.py - Soal 8.9: posisi kemunculan pertama x, atau -1.
import sys
import unittest
from cons_list import List


def find(x, a):
    """Posisi dihitung 0, 1, 2, ...; -1 berarti "x tidak muncul di a"."""
    if a.empty():
        return -1                       # tak ada lagi yang dicari
    if a.head() == x:
        return 0                        # ditemukan tepat di sini
    in_tail = find(x, a.tail())         # tanyakan pada sisa list
    return -1 if in_tail == -1 else in_tail + 1


def find_wrong(x, a):
    """Menggoda tetapi salah: menambahkan 1 bahkan pada jawaban "tidak ditemukan"."""
    if a.empty():
        return -1
    if a.head() == x:
        return 0
    return 1 + find_wrong(x, a.tail())


class TestFind(unittest.TestCase):
    def test_present_and_absent(self):
        a = List.of(7, 5, 3, 8)
        self.assertEqual(2, find(3, a))
        self.assertEqual(-1, find(2, a))

    def test_ends(self):
        a = List.of(7, 5, 3, 8)
        self.assertEqual(0, find(7, a))
        self.assertEqual(3, find(8, a))

    def test_first_of_duplicates_and_empty(self):
        self.assertEqual(1, find(5, List.of(9, 5, 5)))
        self.assertEqual(-1, find(1, List.nil()))
```

Kode 8.26 (Lanjutan) *list_find.py: pencarian yang sama dalam Python.*

```
if __name__ == "__main__":
    a = List.of(7, 5, 3, 8)
    print("a =", a)
    print("find(3, a)       =", find(3, a))
    print("find(2, a)       =", find(2, a))
    print("find_wrong(2, a) =", find_wrong(2, a))
    sys.stdout.flush()
    unittest.main(verbosity=2)
```

```
a = [7, 5, 3, 8]
find(3, a)       = 2
find(2, a)       = -1
find_wrong(2, a) = 3
test_ends (__main__.TestFind.test_ends) ... ok
test_first_of_duplicates_and_empty (__main__.TestFind.test_first_of_duplicates_and_empty)
... ok
test_present_and_absent (__main__.TestFind.test_present_and_absent) ... ok

----------------------------------------------------------------------
Ran 3 tests in 0.000s

OK
```

Baris demi baris. Untuk `find(3, [7, 5, 3, 8])` pemanggilannya turun melalui `[7,5,3,8]`, `[5,3,8]`, `[3,8]`; pemanggilan ketiga menemukan 3 di *head*-nya dan mengembalikan 0; pemanggilan kedua mengembalikan $0 + 1 = 1$; pemanggilan pertama mengembalikan $1 + 1 = 2$. Karena metode berhenti pada kecocokan pertama, `find(5, [9, 5, 5])` mengembalikan 1, bukan 2, seperti yang ditegaskan pemeriksaan `first of duplicates`.

Penyelesaian keliru yang menggoda. `findWrong` mengembalikan `1 + findWrong(x, a.tail())` tanpa melihat apa yang kembali. Bila `x` tidak ada, `-1` dari *list* kosong bertambah satu untuk setiap elemen: untuk empat elemen, $-1 + 4 = 3$. Keluaran `findWrong(2, a) = 3` adalah jenis jawaban salah yang paling buruk, karena 3 adalah posisi yang sah di *list* itu (isinya 8). Pemanggil akan dengan senang hati memakainya. Nilai "sinyal" khusus seperti `-1` harus diteruskan tanpa diubah; `Optional` di Java dan `None` di Python ada, sebagian, untuk mempersulit kesalahan itu.

> **UJI DIRI.** Tulislah `static int findLast(int x, List a)`, posisi kemunculan *terakhir*. Apa yang dikembalikannya untuk `x = 5`, `a = [9, 5, 5]`?
>
> *Jawaban.* Tanyakan *tail* lebih dulu: bila `findLast(x, tail)` bernilai $p \geq 0$, kembalikan $p + 1$; selain itu kembalikan 0 bila *head*-nya `x`, atau `-1` bila bukan. Hasilnya 2.

8.5.3 Soal 8.10 — Apakah semua elemen positif?

> **SOAL.** Include a recursive method `static boolean allPositive(List a)` that returns whether all the elements of `a` are positive, where *positive* here means ≥ 0.
>
> *Dengan kata lain:* tulislah metode rekursif `static boolean allPositive(List a)` yang menjawab apakah semua elemen `a` positif, dengan *positif* di sini berarti tidak kurang dari nol.

Dengan bahasa kita sendiri. Perhatikan definisi yang tidak biasa ini: nol terhitung "positif" dalam soal ini. (Dalam matematika, 0 bukan positif dan bukan negatif, dan nama metode yang lebih jelas adalah `allNonNegative`; tetapi kita mengikuti soal dan mendokumentasikan pilihan itu.) Pertanyaannya adalah "adakah elemen yang di bawah nol?", dan jawaban yang dicari adalah *tidak*.

Rancangan. **Tabel 8.11** memperlihatkan rancangannya. *List* kosong adalah kasus yang menarik: tidak ada elemen yang melanggar aturan, sehingga jawabannya `true`. Ahli logika menyebutnya *kebenaran hampa* (*vacuous truth*), dan itu juga satu-satunya pilihan yang membuat rekursinya berjalan, karena `allPositive([x])` harus sama dengan `x >= 0 && allPositive([])`.

Tabel 8.11 Rancangan `allPositive`: operasi AND logis atas semua elemen, dengan `true` sebagai nilai awal yang netral.

Bagian	Jawaban untuk `allPositive`
List kosong	`true` (tidak ada contoh penyangkal)
Gabung	`head >= 0 && allPositive(tail)`
Kasus tepi yang diuji	`[]`, `[0]`, elemen negatif di posisi *terakhir*

Kode 8.27 (Java) dan **Kode 8.28** (Python) mengikuti tabel itu; uji-ujinya tertanam di setiap program.

Kode 8.27 `AllPositive.java`: memeriksa bahwa tidak ada elemen yang di bawah nol.

```java
// AllPositive.java - Soal 8.10: apakah semua elemen positif (di sini: >= 0)?
public class AllPositive {

    /** Bernilai true bila setiap elemen >= 0; true untuk list kosong. */
    static boolean allPositive(List a) {
        if (a.empty()) {
            return true;                      // tak ada elemen yang dapat melanggar aturan
        }
        return a.head() >= 0 && allPositive(a.tail());
    }

    /** Menggoda tetapi salah: berhenti satu elemen terlalu cepat. */
    static boolean allPositiveWrong(List a) {
        if (a.empty()) {
            return true;
        } else if (a.tail().empty()) {
            return true;                      // head terakhir tidak pernah diperiksa!
        } else if (a.head() >= 0) {
            return allPositiveWrong(a.tail());
        }
        return false;
    }

    private static int passed = 0;

    private static void check(String name, Object expected, Object actual) {
        if (!java.util.Objects.equals(expected, actual)) {
            throw new AssertionError(name + ": expected " + expected + " but got " +
actual);
        }
        passed++;
        System.out.println("PASS " + name);
    }

    public static void main(String[] args) {
        System.out.println("allPositive([7, 5, 3, 8])  = " + allPositive(List.of(7, 5, 3,
8)));
        System.out.println("allPositive([2, -5, 8, 0]) = " + allPositive(List.of(2, -5, 8,
0)));
        System.out.println("allPositiveWrong([4, -1])  = " + allPositiveWrong(List.of(4, -
1)));
        check("all non-negative", true, allPositive(List.of(7, 5, 3, 8)));
        check("one negative", false, allPositive(List.of(2, -5, 8, 0)));
        check("zero counts", true, allPositive(List.of(0)));
        check("negative last", false, allPositive(List.of(4, -1)));
        check("empty", true, allPositive(List.nil()));
        System.out.println("All " + passed + " checks passed.");
    }
}
```

```
allPositive([7, 5, 3, 8])  = true
allPositive([2, -5, 8, 0]) = false
allPositiveWrong([4, -1])  = true
PASS all non-negative
PASS one negative
PASS zero counts
PASS negative last
PASS empty
All 5 checks passed.
```

Kode 8.28 `all_positive.py`: pemeriksaan yang sama dalam Python.

```python
# all_positive.py - Soal 8.10: apakah semua elemen positif (di sini: >= 0)?
import sys
import unittest
from cons_list import List


def all_positive(a):
    """True bila setiap elemen >= 0; True untuk list kosong."""
    if a.empty():
        return True                         # tak ada elemen yang dapat melanggar aturan
    return a.head() >= 0 and all_positive(a.tail())


def all_positive_wrong(a):
    """Menggoda tetapi salah: berhenti satu elemen terlalu cepat."""
    if a.empty():
        return True
    elif a.tail().empty():
        return True                         # head terakhir tidak pernah diperiksa!
    elif a.head() >= 0:
        return all_positive_wrong(a.tail())
    return False


class TestAllPositive(unittest.TestCase):
    def test_cases(self):
        self.assertTrue(all_positive(List.of(7, 5, 3, 8)))
        self.assertFalse(all_positive(List.of(2, -5, 8, 0)))
        self.assertTrue(all_positive(List.of(0)))
        self.assertFalse(all_positive(List.of(4, -1)))
        self.assertTrue(all_positive(List.nil()))


if __name__ == "__main__":
    print("all_positive([7, 5, 3, 8])  =", all_positive(List.of(7, 5, 3, 8)))
    print("all_positive([2, -5, 8, 0]) =", all_positive(List.of(2, -5, 8, 0)))
    print("all_positive_wrong([4, -1]) =", all_positive_wrong(List.of(4, -1)))
    sys.stdout.flush()
    unittest.main(verbosity=2)
```

```
all_positive([7, 5, 3, 8])  = True
all_positive([2, -5, 8, 0]) = False
all_positive_wrong([4, -1]) = True
test_cases (__main__.TestAllPositive.test_cases) ... ok

----------------------------------------------------------------------
Ran 1 test in 0.000s

OK
```

Baris demi baris. `a.head() >= 0 && allPositive(a.tail())` memakai *evaluasi hubung-pendek* (*short-circuit evaluation*): bila *head*-nya negatif, `&&` (di Python `and`) sudah tahu bahwa jawabannya salah dan tidak pernah melakukan pemanggilan rekursif, sehingga metode berhenti pada elemen negatif pertama.

Penyelesaian keliru yang menggoda. `allPositiveWrong` menambahkan kasus dasar kedua, "*list* satu elemen sudah pasti beres", dan mengembalikan `true` untuknya *tanpa melihat elemennya*. Setiap *list* yang satu-satunya elemen negatifnya berada di posisi terakhir dinilai keliru: `allPositiveWrong([4, -1])` mencetak `true`. Kasus dasar tambahan itu dimaksudkan sebagai jalan pintas, dan diam-diam ia melewatkan sebuah pemeriksaan. Uji `negative last` ada justru untuk menangkap hal ini. Sebagai aturan, metode *list* rekursif memerlukan satu kasus dasar, yaitu *list* kosong; kasus dasar kedua adalah tanda bahaya.

> **UJI DIRI.** Tulislah `static boolean anyNegative(List a)`. Apa yang harus dikembalikannya untuk *list* kosong, dan apa hubungannya dengan `allPositive`?
>
> *Jawaban.* `false` untuk *list* kosong, dan `a.head() < 0 || anyNegative(a.tail())` selainnya. Untuk setiap *list* berlaku `anyNegative(a) == !allPositive(a)`.

8.5.4 Soal 8.11 — Mempertahankan hanya elemen positif

> **SOAL.** Include a recursive method `static List positives(List a)` that returns a new list containing all the positive elements of `a` (again, ≥ 0), in the same relative order as in `a`. Example: `[2, 3, -5, 8, -2]` gives `[2, 3, 8]`.
>
> *Dengan kata lain:* tulislah metode rekursif `static List positives(List a)` yang mengembalikan *list* baru berisi semua elemen positif `a` (sekali lagi, tidak kurang dari nol), dalam urutan relatif yang sama seperti di `a`. Contoh: `[2, 3, -5, 8, -2]` menghasilkan `[2, 3, 8]`.

Dengan bahasa kita sendiri. Ini sebuah *filter* (penyaring): setiap elemen dipertahankan atau dibuang, dan elemen yang dipertahankan tetap dalam urutannya.

Rancangan. Saring *tail* lebih dulu; lalu letakkan *head* di depan hasilnya bila ia lolos uji, atau biarkan hasilnya apa adanya bila tidak lolos (**Tabel 8.12**).

Tabel 8.12 Rancangan `positives`: sebuah *filter* menjaga urutan karena setiap *head* yang dipertahankan diletakkan di depan *tail* yang sudah disaring.

Bagian	Jawaban untuk `positives`
List kosong	`nil()`
Gabung, *head* ≥ 0	`cons(head, positives(tail))`
Gabung, *head* < 0	`positives(tail)`

Kode 8.29 (Java) dan **Kode 8.30** (Python) mengikuti tabel itu; uji-ujinya tertanam di setiap program.

Kode 8.29 `Positives.java`: menyaring elemen yang paling sedikit nol.

```java
// Positives.java - Soal 8.11: pertahankan elemen >= 0, dalam urutan aslinya.
public class Positives {

    static List positives(List a) {
        if (a.empty()) {
            return List.nil();
        }
        List rest = positives(a.tail());      // saring tail lebih dahulu
        return (a.head() >= 0) ? List.cons(a.head(), rest) : rest;
    }

    /** Menggoda tetapi salah: menyerah pada elemen negatif pertama. */
    static List positivesWrong(List a) {
        if (a.empty() || a.head() < 0) {
            return List.nil();
        }
        return List.cons(a.head(), positivesWrong(a.tail()));
    }

    private static int passed = 0;

    private static void check(String name, Object expected, Object actual) {
        if (!java.util.Objects.equals(expected, actual)) {
            throw new AssertionError(name + ": expected " + expected + " but got " +
actual);
        }
        passed++;
        System.out.println("PASS " + name);
    }

    public static void main(String[] args) {
        List a = List.of(2, 3, -5, 8, -2);
        System.out.println("positives(" + a + ")      = " + positives(a));
        System.out.println("positivesWrong(" + a + ") = " + positivesWrong(a));
        check("sample", "[2, 3, 8]", positives(a).toString());
        check("zero kept", "[0, 4]", positives(List.of(-1, 0, 4)).toString());
        check("none", "[]", positives(List.of(-3, -4)).toString());
        check("empty", "[]", positives(List.nil()).toString());
        System.out.println("All " + passed + " checks passed.");
    }
}
```

```
positives([2, 3, -5, 8, -2])      = [2, 3, 8]
positivesWrong([2, 3, -5, 8, -2]) = [2, 3]
PASS sample
PASS zero kept
PASS none
PASS empty
All 4 checks passed.
```

Kode 8.30 `positives.py`: *filter* yang sama dalam Python.

```python
# positives.py - Soal 8.11: pertahankan elemen >= 0, dalam urutan aslinya.
import sys
import unittest
from cons_list import List


def positives(a):
    if a.empty():
        return List.nil()
    rest = positives(a.tail())            # saring tail lebih dahulu
    return List.cons(a.head(), rest) if a.head() >= 0 else rest


def positives_wrong(a):
    """Menggoda tetapi salah: menyerah pada elemen negatif pertama."""
    if a.empty() or a.head() < 0:
        return List.nil()
    return List.cons(a.head(), positives_wrong(a.tail()))


class TestPositives(unittest.TestCase):
    def test_cases(self):
        self.assertEqual("[2, 3, 8]", str(positives(List.of(2, 3, -5, 8, -2))))
        self.assertEqual("[0, 4]", str(positives(List.of(-1, 0, 4))))
        self.assertEqual("[]", str(positives(List.of(-3, -4))))
        self.assertEqual("[]", str(positives(List.nil())))


if __name__ == "__main__":
    a = List.of(2, 3, -5, 8, -2)
    print(f"positives({a})       = {positives(a)}")
    print(f"positives_wrong({a}) = {positives_wrong(a)}")
    sys.stdout.flush()
    unittest.main(verbosity=2)
```

```
positives([2, 3, -5, 8, -2])       = [2, 3, 8]
positives_wrong([2, 3, -5, 8, -2]) = [2, 3]
test_cases (__main__.TestPositives.test_cases) ... ok

----------------------------------------------------------------------
Ran 1 test in 0.000s

OK
```

Baris demi baris. Variabel lokal `rest` menyimpan *tail* yang sudah disaring; ekspresi kondisional kemudian memasang *head* di depannya dengan `cons` atau mengembalikannya tanpa diubah. Urutan terjaga tanpa usaha apa pun: *head* dari `a` datang sebelum setiap elemen *tail*-nya, dan `cons` meletakkannya sebelum setiap elemen *tail* yang sudah disaring.

Penyelesaian keliru yang menggoda. `positivesWrong` memperlakukan elemen negatif sebagai akhir *list*: `if (a.empty() || a.head() < 0) return nil()`. Hasilnya `[2, 3]`, dan 8 yang datang sesudah -5 hilang. Inilah versi *list* dari memakai `break` padahal yang dimaksud `continue`. Uji dengan elemen negatif *di tengah* menangkapnya.

> **UJI DIRI.** Tulislah `static List evens(List a)` yang mempertahankan elemen genap. Apa yang dikembalikannya untuk `[1, 2, 3, 4, -6]`?
>
> *Jawaban.* Ganti ujinya dengan `a.head() % 2 == 0`: `[2, 4, -6]`. (Di Java maupun Python, `-6 % 2` bernilai 0.)

8.5.5 Soal 8.12 — Apakah *list* terurut?

> **SOAL.** Include a recursive method `static boolean sorted(List a)` that returns whether `a` is sorted in increasing order. There can be duplicate copies of elements, but sortedness requires that all duplicate copies appear together.
>
> *Dengan kata lain:* tulislah metode rekursif `static boolean sorted(List a)` yang menjawab apakah `a` terurut menaik. Salinan ganda elemen boleh ada, tetapi keterurutan menuntut agar semua salinan ganda muncul berdampingan.

Dengan bahasa kita sendiri. "Menaik dengan duplikat yang diperbolehkan" berarti *tidak turun* (*non-decreasing*): setiap elemen lebih kecil daripada atau sama dengan elemen berikutnya. Bila hal itu berlaku untuk setiap pasangan bersebelahan, elemen-elemen yang sama dengan sendirinya berdampingan, karena apa pun yang terletak di antara dua salinan v harus sekaligus $\geq v$ dan $\leq v$, yaitu sama dengan v.

Rancangan. Di sini langkah gabung memerlukan *dua* elemen, yaitu *head* dan *head* dari *tail*, sehingga ada dua kasus dasar: *list* kosong dan *list* satu elemen, keduanya terurut (**Tabel 8.13**). Inilah satu-satunya keadaan ketika kasus dasar kedua memang benar, karena langkah gabungnya sungguh-sungguh melihat dua elemen.

Tabel 8.13 Rancangan `sorted`: setiap pemanggilan membandingkan satu pasangan bersebelahan dan mendelegasikan sisanya.

Bagian	Jawaban untuk `sorted`
0 atau 1 elemen	`true`
Gabung	`head <= head(tail) && sorted(tail)`
Kasus tepi yang diuji	duplikat berdampingan `[2,5,5,8]`; duplikat terpisah `[5,7,5]`

Kode 8.31 (Java) dan **Kode 8.32** (Python) mengikuti tabel itu; uji-ujinya tertanam di setiap program.

Kode 8.31 `SortedCheck.java`: memeriksa urutan tidak turun pasangan demi pasangan.

```java
// SortedCheck.java - Soal 8.12: apakah List berurutan menaik (tidak menurun)?
public class SortedCheck {

    static boolean sorted(List a) {
        if (a.empty() || a.tail().empty()) {
            return true;                        // 0 atau 1 elemen: tak ada yang dibandingkan
        }
        return a.head() <= a.tail().head() && sorted(a.tail());
    }

    /** Menggoda tetapi salah: < yang ketat menolak tetangga yang sama. */
    static boolean sortedWrong(List a) {
        if (a.empty() || a.tail().empty()) {
            return true;
        }
        return a.head() < a.tail().head() && sortedWrong(a.tail());
    }

    private static int passed = 0;

    private static void check(String name, Object expected, Object actual) {
        if (!java.util.Objects.equals(expected, actual)) {
            throw new AssertionError(name + ": expected " + expected + " but got " +
actual);
        }
        passed++;
        System.out.println("PASS " + name);
    }

    public static void main(String[] args) {
        List dup = List.of(2, 5, 5, 8);
        System.out.println("sorted([2, 3, -5, 8, -2]) = " + sorted(List.of(2, 3, -5, 8, -
2)));
        System.out.println("sorted([2, 3, 8])         = " + sorted(List.of(2, 3, 8)));
        System.out.println("sorted(" + dup + ")      = " + sorted(dup));
        System.out.println("sortedWrong(" + dup + ") = " + sortedWrong(dup));
        check("unsorted", false, sorted(List.of(2, 3, -5, 8, -2)));
        check("sorted", true, sorted(List.of(2, 3, 8)));
        check("duplicates together", true, sorted(dup));
        check("duplicates apart", false, sorted(List.of(5, 7, 5)));
        check("empty", true, sorted(List.nil()));
        check("single", true, sorted(List.of(42)));
        System.out.println("All " + passed + " checks passed.");
    }
}
```

```
sorted([2, 3, -5, 8, -2]) = false
sorted([2, 3, 8])         = true
sorted([2, 5, 5, 8])      = true
sortedWrong([2, 5, 5, 8]) = false
PASS unsorted
PASS sorted
PASS duplicates together
PASS duplicates apart
PASS empty
PASS single
All 6 checks passed.
```

Kode 8.32 `sorted_check.py`: pemeriksaan yang sama dalam Python, dinamai `is_sorted` agar tidak menutupi fungsi bawaan `sorted`.

```python
# sorted_check.py - Soal 8.12: apakah List berurutan menaik (tidak menurun)?
import sys
import unittest
from cons_list import List


def is_sorted(a):
    if a.empty() or a.tail().empty():
        return True                          # 0 atau 1 elemen: tak ada yang dibandingkan
    return a.head() <= a.tail().head() and is_sorted(a.tail())


def is_sorted_wrong(a):
    """Menggoda tetapi salah: < yang ketat menolak tetangga yang sama."""
    if a.empty() or a.tail().empty():
        return True
    return a.head() < a.tail().head() and is_sorted_wrong(a.tail())


class TestSorted(unittest.TestCase):
    def test_cases(self):
        self.assertFalse(is_sorted(List.of(2, 3, -5, 8, -2)))
        self.assertTrue(is_sorted(List.of(2, 3, 8)))
        self.assertTrue(is_sorted(List.of(2, 5, 5, 8)))
        self.assertFalse(is_sorted(List.of(5, 7, 5)))
        self.assertTrue(is_sorted(List.nil()))
        self.assertTrue(is_sorted(List.of(42)))


if __name__ == "__main__":
    dup = List.of(2, 5, 5, 8)
    print("is_sorted([2, 3, -5, 8, -2]) =", is_sorted(List.of(2, 3, -5, 8, -2)))
    print("is_sorted([2, 3, 8])         =", is_sorted(List.of(2, 3, 8)))
    print(f"is_sorted({dup})      = {is_sorted(dup)}")
    print(f"is_sorted_wrong({dup}) = {is_sorted_wrong(dup)}")
    sys.stdout.flush()
    unittest.main(verbosity=2)
```

```
is_sorted([2, 3, -5, 8, -2]) = False
is_sorted([2, 3, 8])         = True
is_sorted([2, 5, 5, 8])      = True
is_sorted_wrong([2, 5, 5, 8]) = False
test_cases (__main__.TestSorted.test_cases) ... ok

----------------------------------------------------------------------
Ran 1 test in 0.000s

OK
```

Baris demi baris. `if` yang pertama mencakup kedua kasus dasar dalam satu syarat; `||` menjamin bahwa `a.tail()` hanya dievaluasi bila `a` tidak kosong, sehingga ia tidak mungkin melempar `ListException`. Baris rekursifnya membandingkan pasangan pertama dan menanyakan pasangan-pasangan sisanya kepada *tail*. *List* dengan n elemen memiliki $n-1$ pasangan bersebelahan, dan setiap pasangan dibandingkan tepat satu kali.

Penyelesaian keliru yang menggoda. `sortedWrong` memakai `<` alih-alih `<=`. Ia menyatakan `[2, 5, 5, 8]` tidak terurut, bertentangan dengan soal yang secara tegas memperbolehkan duplikat. Memilih antara `<` dan `<=` adalah *bug* satu karakter yang paling umum dalam pemrograman; uji dengan tetangga yang sama menuntaskannya.

> **UJI DIRI.** Mana di antara ini yang diterima `sorted`: `[]`, `[3, 3, 3]`, `[1, 2, 2, 1]`, `[-4, 0, 7]`?
>
> *Jawaban.* `[]` benar, `[3, 3, 3]` benar, `[1, 2, 2, 1]` salah (pasangan terakhir 2, 1 gagal), `[-4, 0, 7]` benar.

8.5.6 Soal 8.13 — Menggabungkan dua *list* terurut

> **SOAL.** Include a recursive method `static List merge(List a, List b)` that, given two sorted lists, returns a new sorted list containing all the elements of `a` and all the elements of `b`. All duplicates, within `a`, within `b`, or across them, are kept. Examples: `[2, 5, 5, 8]` and `[5, 7, 8, 9]` give `[2, 5, 5, 5, 7, 8, 8, 9]`; `[2, 5, 5, 8]` and `[9]` give `[2, 5, 5, 8, 9]`.
>
> *Dengan kata lain:* tulislah metode rekursif `static List merge(List a, List b)` yang, dari dua *list* terurut, mengembalikan *list* terurut baru berisi semua elemen `a` dan semua elemen `b`. Semua duplikat, di dalam `a`, di dalam `b`, maupun di antara keduanya, dipertahankan. Contohnya seperti di atas.

Dengan bahasa kita sendiri. Elemen terkecil dari hasil adalah yang lebih kecil di antara kedua *head*. Ambil itu, lalu gabungkan apa yang tersisa. Bila satu *list* habis, sisa *list* yang lain adalah sisa jawabannya. Inilah jantung *Merge Sort* [5].

Rancangan. Kini ada dua *list*, sehingga ada dua kasus dasar, satu untuk masing-masing *list* yang habis (**Tabel 8.14**).

Tabel 8.14 Rancangan `merge`: setiap pemanggilan memindahkan *head* yang lebih kecil ke hasil, sehingga setiap pemanggilan membuang satu elemen dari salah satu *list*.

Keadaan	Jawaban
`a` kosong	`b` (seluruhnya)
`b` kosong	`a` (seluruhnya)
`head(a) <= head(b)`	`cons(head(a), merge(tail(a), b))`
selainnya	`cons(head(b), merge(a, tail(b)))`

Setiap pemanggilan membuang tepat satu elemen dari `a` atau dari `b`, sehingga banyaknya pemanggilan yang bekerja adalah

$$\text{calls}(\text{merge}(a, b)) \leq |a| + |b|, \tag{8.5}$$

dengan $|a|$ adalah panjang a. Persamaan (8.5) menjelaskan mengapa penggabungan itu cepat: biayanya linear terhadap panjang total.

Kode 8.33 (Java) dan **Kode 8.34** (Python) mengimplementasikan keempat kasus itu; uji-ujinya tertanam di setiap program.

Kode 8.33 `MergeLists.java`: menggabungkan dua *list* terurut dengan kedua kasus dasarnya.

```
// MergeLists.java - Soal 8.13: menggabungkan dua list terurut, duplikat dipertahankan.
public class MergeLists {

    static List merge(List a, List b) {
        if (a.empty()) {
            return b;                          // a sudah habis: sisanya adalah b
        }
        if (b.empty()) {
            return a;                          // b sudah habis: sisanya adalah a
        }
        if (a.head() <= b.head()) {
            return List.cons(a.head(), merge(a.tail(), b));
        }
        return List.cons(b.head(), merge(a, b.tail()));
    }

    /** Menggoda tetapi salah: a yang kosong dianggap "tak ada yang tersisa sama sekali".
*/
    static List mergeWrong(List a, List b) {
        if (a.empty()) {
            return List.nil();
        }
        if (b.empty()) {
            return a;
        }
        if (a.head() <= b.head()) {
            return List.cons(a.head(), mergeWrong(a.tail(), b));
        }
        return List.cons(b.head(), mergeWrong(a, b.tail()));
    }

    private static int passed = 0;

    private static void check(String name, Object expected, Object actual) {
        if (!java.util.Objects.equals(expected, actual)) {
            throw new AssertionError(name + ": expected " + expected + " but got " +
actual);
        }
        passed++;
        System.out.println("PASS " + name);
    }
```

Kode 8.33 (Lanjutan) *MergeLists.java: menggabungkan dua list terurut dengan kedua kasus dasarnya.*

```
    public static void main(String[] args) {
        List a = List.of(2, 5, 5, 8);
        List b = List.of(5, 7, 8, 9);
        List c = List.of(9);
        System.out.println("merge(" + a + ", " + b + ") = " + merge(a, b));
        System.out.println("merge(" + a + ", " + c + ") = " + merge(a, c));
        System.out.println("mergeWrong(" + a + ", " + c + ") = " + mergeWrong(a, c));
        check("sample 1", "[2, 5, 5, 5, 7, 8, 8, 9]", merge(a, b).toString());
        check("sample 2", "[2, 5, 5, 8, 9]", merge(a, c).toString());
        check("empty left", "[9]", merge(List.nil(), c).toString());
        check("empty right", a.toString(), merge(a, List.nil()).toString());
        check("both empty", "[]", merge(List.nil(), List.nil()).toString());
        System.out.println("All " + passed + " checks passed.");
    }
}
```

```
merge([2, 5, 5, 8], [5, 7, 8, 9]) = [2, 5, 5, 5, 7, 8, 8, 9]
merge([2, 5, 5, 8], [9]) = [2, 5, 5, 8, 9]
mergeWrong([2, 5, 5, 8], [9]) = [2, 5, 5, 8]
PASS sample 1
PASS sample 2
PASS empty left
PASS empty right
PASS both empty
All 5 checks passed.
```

Kode 8.34 `merge_lists.py`: penggabungan yang sama dalam Python.

```python
# merge_lists.py - Soal 8.13: menggabungkan dua list terurut, duplikat dipertahankan.
import sys
import unittest
from cons_list import List


def merge(a, b):
    if a.empty():
        return b                          # a sudah habis: sisanya adalah b
    if b.empty():
        return a                          # b sudah habis: sisanya adalah a
    if a.head() <= b.head():
        return List.cons(a.head(), merge(a.tail(), b))
    return List.cons(b.head(), merge(a, b.tail()))


def merge_wrong(a, b):
    """Menggoda tetapi salah: a yang kosong dianggap "tak ada yang tersisa sama sekali"."""
    if a.empty():
        return List.nil()
    if b.empty():
        return a
    if a.head() <= b.head():
        return List.cons(a.head(), merge_wrong(a.tail(), b))
    return List.cons(b.head(), merge_wrong(a, b.tail()))


class TestMerge(unittest.TestCase):
    def test_samples(self):
        a = List.of(2, 5, 5, 8)
        self.assertEqual("[2, 5, 5, 5, 7, 8, 8, 9]", str(merge(a, List.of(5, 7, 8, 9))))
        self.assertEqual("[2, 5, 5, 8, 9]", str(merge(a, List.of(9))))

    def test_empty_sides(self):
        self.assertEqual("[9]", str(merge(List.nil(), List.of(9))))
        self.assertEqual("[1, 2]", str(merge(List.of(1, 2), List.nil())))
        self.assertEqual("[]", str(merge(List.nil(), List.nil())))


if __name__ == "__main__":
    a, b, c = List.of(2, 5, 5, 8), List.of(5, 7, 8, 9), List.of(9)
    print(f"merge({a}, {b}) = {merge(a, b)}")
    print(f"merge({a}, {c}) = {merge(a, c)}")
    print(f"merge_wrong({a}, {c}) = {merge_wrong(a, c)}")
    sys.stdout.flush()
    unittest.main(verbosity=2)
```

```
merge([2, 5, 5, 8], [5, 7, 8, 9]) = [2, 5, 5, 5, 7, 8, 8, 9]
merge([2, 5, 5, 8], [9]) = [2, 5, 5, 8, 9]
merge_wrong([2, 5, 5, 8], [9]) = [2, 5, 5, 8]
test_empty_sides (__main__.TestMerge.test_empty_sides) ... ok
test_samples (__main__.TestMerge.test_samples) ... ok

----------------------------------------------------------------------
Ran 2 tests in 0.000s

OK
```

Baris demi baris. Telusuri contoh kedua: *head* 2 dan 9, ambil 2; *head* 5 dan 9, ambil 5; 5 lagi; lalu 8; kini `a` kosong, sehingga jawaban untuk sisanya adalah `b` itu sendiri, `[9]`, yang dikembalikan utuh dan *dipakai bersama*, bukan disalin. Berbagi seperti itu aman karena *list* bersifat imutabel (Subbab 8.3.2). `<=` pada kasus ketiga mengambil elemen dari `a` lebih dulu bila kedua *head* sama; kedua pilihan menghasilkan deretan bilangan yang sama, tetapi `<=` menjaga elemen `a` yang sama nilainya tetap di depan elemen `b`, sebuah sifat yang disebut *stabilitas* (*stability*), yang penting bila elemen membawa data lebih dari sekadar bilangan.

Penyelesaian keliru yang menggoda. `mergeWrong` mengembalikan `nil()` bila `a` kosong, seolah-olah habisnya satu *list* berarti seluruh pekerjaan sudah selesai. Keluaran `mergeWrong([2, 5, 5, 8], [9]) = [2, 5, 5, 8]` memperlihatkan 9 yang hilang. Keempat pemeriksaan bernama `empty left`, `empty right`, `both empty`, dan `sample 2` ada karena rekursi atas dua *list* memiliki empat "bentuk kekosongan", dan masing-masing memerlukan uji.

> **UJI DIRI.** Berapa pemanggilan yang dilakukan `merge([1, 3], [2, 4, 6, 8])` sebelum mencapai kasus dasar, dan apa hasilnya?
>
> *Jawaban. Head* yang diambil adalah 1, 2, 3, 4; setelah pemanggilan keempat `a` kosong, dan `[6, 8]` dikembalikan utuh. Empat pemanggilan yang bekerja (kurang dari $|a| + |b| = 6$), hasilnya `[1, 2, 3, 4, 6, 8]`.

8.5.7 Soal 8.14 — Melewati setiap salinan sebuah elemen

> **SOAL.** Include a recursive method `static List skip(int x, List a)` that returns a copy of `a` with every occurrence of `x` skipped. Example: `x = 5`, `a = [2, 5, 5, 5, 7, 8, 8, 9]` gives `[2, 7, 8, 8, 9]`. (This method is needed in Problem 8.15.)
>
> *Dengan kata lain:* tulislah metode rekursif `static List skip(int x, List a)` yang mengembalikan salinan `a` dengan setiap kemunculan `x` dilewati. Contoh: `x = 5`, `a = [2, 5, 5, 5, 7, 8, 8, 9]` menghasilkan `[2, 7, 8, 8, 9]`. (Metode ini diperlukan pada Soal 8.15.)

Dengan bahasa kita sendiri. `skip` adalah sebuah *filter*, seperti `positives`, dengan uji "elemennya bukan `x`". Kita menyelesaikannya sebelum penghapusan duplikat karena Soal 8.15 memakainya.

Rancangan. **Tabel 8.15** adalah **Tabel 8.12** dengan uji yang berbeda.

Tabel 8.15 Rancangan `skip`: sebuah *filter* dengan uji "tidak sama dengan `x`".

Bagian	Jawaban untuk `skip`
List kosong	`nil()`
Gabung, *head* `== x`	`skip(x, tail)`
Gabung, *head* `!= x`	`cons(head, skip(x, tail))`
Kasus tepi yang diuji	`x` di akhir; `x` tidak ada; *list* yang hanya berisi `x`

Kode 8.35 (Java) dan **Kode 8.36** (Python) mengikuti tabel itu; uji-ujinya tertanam di setiap program.

Kode 8.35 `SkipElement.java`: membuang setiap salinan satu nilai.

```java
// SkipElement.java - Soal 8.14: salinan list tanpa satu pun salinan x.
public class SkipElement {

    static List skip(int x, List a) {
        if (a.empty()) {
            return List.nil();
        }
        List rest = skip(x, a.tail());
        return (a.head() == x) ? rest : List.cons(a.head(), rest);
    }

    /** Menggoda tetapi salah: list satu elemen disalin tanpa dilihat. */
    static List skipWrong(int x, List a) {
        if (a.empty()) {
            return List.nil();
        }
        if (a.tail().empty()) {
            return List.cons(a.head(), List.nil());
        }
        List rest = skipWrong(x, a.tail());
        return (a.head() == x) ? rest : List.cons(a.head(), rest);
    }

    private static int passed = 0;

    private static void check(String name, Object expected, Object actual) {
        if (!java.util.Objects.equals(expected, actual)) {
            throw new AssertionError(name + ": expected " + expected + " but got " +
actual);
        }
        passed++;
        System.out.println("PASS " + name);
    }

    public static void main(String[] args) {
        List a = List.of(2, 5, 5, 5, 7, 8, 8, 9);
        System.out.println("skip(5, a)      = " + skip(5, a));
        System.out.println("skip(9, a)      = " + skip(9, a));
        System.out.println("skipWrong(9, a) = " + skipWrong(9, a));
        check("sample", "[2, 7, 8, 8, 9]", skip(5, a).toString());
        check("last element", "[2, 5, 5, 5, 7, 8, 8]", skip(9, a).toString());
        check("absent", a.toString(), skip(4, a).toString());
        check("only x", "[]", skip(3, List.of(3, 3)).toString());
        System.out.println("All " + passed + " checks passed.");
    }
}
```

```
skip(5, a)      = [2, 7, 8, 8, 9]
skip(9, a)      = [2, 5, 5, 5, 7, 8, 8]
skipWrong(9, a) = [2, 5, 5, 5, 7, 8, 8, 9]
PASS sample
PASS last element
PASS absent
PASS only x
All 4 checks passed.
```

Kode 8.36 `skip_element.py`: *filter* yang sama dalam Python.

```python
# skip_element.py - Soal 8.14: salinan list tanpa satu pun salinan x.
import sys
import unittest
from cons_list import List


def skip(x, a):
    if a.empty():
        return List.nil()
    rest = skip(x, a.tail())
    return rest if a.head() == x else List.cons(a.head(), rest)


def skip_wrong(x, a):
    """Menggoda tetapi salah: list satu elemen disalin tanpa dilihat."""
    if a.empty():
        return List.nil()
    if a.tail().empty():
        return List.cons(a.head(), List.nil())
    rest = skip_wrong(x, a.tail())
    return rest if a.head() == x else List.cons(a.head(), rest)


class TestSkip(unittest.TestCase):
    def test_cases(self):
        a = List.of(2, 5, 5, 5, 7, 8, 8, 9)
        self.assertEqual("[2, 7, 8, 8, 9]", str(skip(5, a)))
        self.assertEqual("[2, 5, 5, 5, 7, 8, 8]", str(skip(9, a)))
        self.assertEqual(str(a), str(skip(4, a)))
        self.assertEqual("[]", str(skip(3, List.of(3, 3))))


if __name__ == "__main__":
    a = List.of(2, 5, 5, 5, 7, 8, 8, 9)
    print("skip(5, a)       =", skip(5, a))
    print("skip(9, a)       =", skip(9, a))
    print("skip_wrong(9, a) =", skip_wrong(9, a))
    sys.stdout.flush()
    unittest.main(verbosity=2)
```

```
skip(5, a)       = [2, 7, 8, 8, 9]
skip(9, a)       = [2, 5, 5, 5, 7, 8, 8]
skip_wrong(9, a) = [2, 5, 5, 5, 7, 8, 8, 9]
test_cases (__main__.TestSkip.test_cases) ... ok

----------------------------------------------------------------------
Ran 1 test in 0.000s

OK
```

Baris demi baris. Kerja sungguhan metode ini hanya dua baris: saring *tail*, lalu pertahankan atau buang *head*. Pemeriksaan `only x` memperlihatkan bahwa *list* yang seluruhnya berisi `x` menjadi kosong.

Penyelesaian keliru yang menggoda. `skipWrong` mengulang pola yang kita lihat pada Soal 8.10: kasus dasar tambahan menyalin *list* satu elemen *tanpa menguji elemennya*. Setiap `x` di posisi terakhir lolos: `skipWrong(9, a)` masih berakhir dengan 9. Perhatikan bahwa `skip(5, a)`, contoh dari soal, memberikan jawaban yang benar untuk kedua versi; hanya uji kasus tepi (`last element`) yang membongkar *bug* itu.

> **UJI DIRI.** Apa yang dikembalikan `skip(8, [8, 2, 8])`, dan berapa pemanggilan rekursif yang dilakukan seluruhnya (termasuk pemanggilan pada *list* kosong)?

> *Jawaban.* `[2]`. Empat pemanggilan: pada `[8, 2, 8]`, `[2, 8]`, `[8]`, dan `[]`, satu untuk setiap elemen ditambah satu untuk *list* kosong.

8.5.8 Soal 8.15 — Menghapus duplikat

> **SOAL.** Include a recursive method `static List removeDuplicates(List a)` that returns a copy of `a` with all duplicate copies removed. Example: `[2, 5, 5, 5, 7, 8, 8, 9]` gives `[2, 5, 7, 8, 9]`. Hint: define helper methods, such as `skip` of Problem 8.14.
>
> *Dengan kata lain:* tulislah metode rekursif `static List removeDuplicates(List a)` yang mengembalikan salinan `a` dengan semua salinan ganda dihapus. Contoh: `[2, 5, 5, 5, 7, 8, 8, 9]` menghasilkan `[2, 5, 7, 8, 9]`. Petunjuk: definisikan metode bantu, seperti `skip` dari Soal 8.14.

Dengan bahasa kita sendiri. Pertahankan salinan pertama setiap nilai dan hapus salinan-salinan sesudahnya. Contoh dalam soal kebetulan terurut, tetapi tidak ada kalimat dalam soal yang menyatakan bahwa masukannya harus terurut, sehingga metode juga harus bekerja pada `[3, 1, 3, 2, 1]` dan menghasilkan `[3, 1, 2]`.

Rancangan. Pertahankan *head*; hapus setiap salinannya yang datang kemudian dari *tail* dengan `skip`; lalu hapus duplikat dari apa yang tersisa (**Tabel 8.16**).

Tabel 8.16 Rancangan `removeDuplicates`: *head* bertahan, salinan-salinannya yang datang kemudian dilewati, dan rekursi menangani sisanya.

Bagian	Jawaban untuk `removeDuplicates`
List kosong	`nil()` (tanpa elemen, tanpa duplikat)
Gabung	`cons(head, removeDuplicates(skip(head, tail)))`
Metode bantu	`SkipElement.skip` dari Soal 8.14, dipakai ulang, bukan disalin

Kode 8.37 (Java) dan **Kode 8.38** (Python) mengikuti tabel itu; uji-ujinya tertanam di setiap program.

Kode 8.37 `RemoveDuplicates.java`: menghapus duplikat dengan bantuan `skip`.

```java
// RemoveDuplicates.java - Soal 8.15: pertahankan hanya salinan pertama setiap elemen.
public class RemoveDuplicates {

    /** Pertahankan head, hapus salinan-salinan berikutnya (skip), lalu ulangi pada
sisanya. */
    static List removeDuplicates(List a) {
        if (a.empty()) {
            return List.nil();                // tanpa elemen, tanpa duplikat
        }
        List restWithoutHead = SkipElement.skip(a.head(), a.tail());
        return List.cons(a.head(), removeDuplicates(restWithoutHead));
    }

    /** Menggoda tetapi salah: hanya membandingkan tetangga, sehingga memerlukan list
terurut. */
    static List removeDuplicatesWrong(List a) {
        if (a.empty() || a.tail().empty()) {
            return a;
        }
        if (a.head() == a.tail().head()) {
            return removeDuplicatesWrong(a.tail());
        }
        return List.cons(a.head(), removeDuplicatesWrong(a.tail()));
    }

    private static int passed = 0;

    private static void check(String name, Object expected, Object actual) {
        if (!java.util.Objects.equals(expected, actual)) {
            throw new AssertionError(name + ": expected " + expected + " but got " +
actual);
        }
        passed++;
        System.out.println("PASS " + name);
    }

    public static void main(String[] args) {
        List sortedList = List.of(2, 5, 5, 5, 7, 8, 8, 9);
        List unsorted = List.of(3, 1, 3, 2, 1);
        System.out.println("removeDuplicates(" + sortedList + ") = " +
removeDuplicates(sortedList));
        System.out.println("removeDuplicates(" + unsorted + ")          = " +
removeDuplicates(unsorted));
        System.out.println("removeDuplicatesWrong(" + unsorted + ")     = " +
removeDuplicatesWrong(unsorted));
        check("sorted sample", "[2, 5, 7, 8, 9]", removeDuplicates(sortedList).toString());
        check("unsorted", "[3, 1, 2]", removeDuplicates(unsorted).toString());
        check("empty", "[]", removeDuplicates(List.nil()).toString());
        check("all equal", "[4]", removeDuplicates(List.of(4, 4, 4)).toString());
        System.out.println("All " + passed + " checks passed.");
    }
}
```

```
removeDuplicates([2, 5, 5, 5, 7, 8, 8, 9]) = [2, 5, 7, 8, 9]
removeDuplicates([3, 1, 3, 2, 1])          = [3, 1, 2]
removeDuplicatesWrong([3, 1, 3, 2, 1])     = [3, 1, 3, 2, 1]
PASS sorted sample
PASS unsorted
PASS empty
PASS all equal
All 4 checks passed.
```

Kode 8.38 `remove_duplicates.py`: metode yang sama dalam Python, dengan mengimpor `skip` dari Soal 8.14.

```python
# remove_duplicates.py - Soal 8.15: pertahankan hanya salinan pertama setiap elemen.
import sys
import unittest
from cons_list import List
from skip_element import skip


def remove_duplicates(a):
    """Pertahankan head, hapus salinan-salinan berikutnya (skip), lalu ulangi pada
sisanya."""
    if a.empty():
        return List.nil()                # tanpa elemen, tanpa duplikat
    rest_without_head = skip(a.head(), a.tail())
    return List.cons(a.head(), remove_duplicates(rest_without_head))


def remove_duplicates_wrong(a):
    """Menggoda tetapi salah: hanya membandingkan tetangga, sehingga memerlukan list
terurut."""
    if a.empty() or a.tail().empty():
        return a
    if a.head() == a.tail().head():
        return remove_duplicates_wrong(a.tail())
    return List.cons(a.head(), remove_duplicates_wrong(a.tail()))


class TestRemoveDuplicates(unittest.TestCase):
    def test_cases(self):
        self.assertEqual("[2, 5, 7, 8, 9]",
                         str(remove_duplicates(List.of(2, 5, 5, 5, 7, 8, 8, 9))))
        self.assertEqual("[3, 1, 2]", str(remove_duplicates(List.of(3, 1, 3, 2, 1))))
        self.assertEqual("[]", str(remove_duplicates(List.nil())))
        self.assertEqual("[4]", str(remove_duplicates(List.of(4, 4, 4))))


if __name__ == "__main__":
    s, u = List.of(2, 5, 5, 5, 7, 8, 8, 9), List.of(3, 1, 3, 2, 1)
    print(f"remove_duplicates({s}) = {remove_duplicates(s)}")
    print(f"remove_duplicates({u})          = {remove_duplicates(u)}")
    print(f"remove_duplicates_wrong({u})    = {remove_duplicates_wrong(u)}")
    sys.stdout.flush()
    unittest.main(verbosity=2)
```

```
remove_duplicates([2, 5, 5, 5, 7, 8, 8, 9]) = [2, 5, 7, 8, 9]
remove_duplicates([3, 1, 3, 2, 1])          = [3, 1, 2]
remove_duplicates_wrong([3, 1, 3, 2, 1])    = [3, 1, 3, 2, 1]
test_cases (__main__.TestRemoveDuplicates.test_cases) ... ok

----------------------------------------------------------------------
Ran 1 test in 0.000s

OK
```

Baris demi baris. Untuk `[3, 1, 3, 2, 1]`: pertahankan 3, lewati 3 yang lain dari `[1, 3, 2, 1]` sehingga diperoleh `[1, 2, 1]`; pertahankan 1, lewati 1 dari `[2, 1]` sehingga diperoleh `[2]`; pertahankan 2. Hasilnya `[3, 1, 2]`. Program Java memanggil `SkipElement.skip(...)`, metode statis milik kelas lain di folder yang sama, dan program Python mengimpornya dengan `from skip_element import skip`: fungsi bantu itu *dipakai ulang*, sehingga *bug* yang diperbaiki di satu tempat akan terperbaiki di mana-mana.

Harga dari kesederhanaan ini adalah waktu. Pada *list* dengan n elemen yang berbeda-beda, pemanggilan `skip` pertama menjelajahi $n - 1$ elemen, berikutnya $n - 2$, dan seterusnya, sehingga total pekerjaannya adalah

$$(n-1)+(n-2)+\cdots+1+0=\frac{n(n-1)}{2}, \tag{8.6}$$

yang tumbuh seperti n^2. Untuk *list* dalam bab ini, itu tidak berarti apa-apa; untuk sejuta rekaman pelanggan, Persamaan (8.6) memberikan sekitar 5×10^{11} langkah, dan *hash set* (Bab 9) adalah perkakas yang tepat.

Penyelesaian keliru yang menggoda. `removeDuplicatesWrong` hanya membandingkan setiap elemen dengan tetangganya, yang benar untuk masukan terurut dan memang itulah yang disiratkan contohnya. Pada *list* yang tidak terurut, ia tidak menghapus apa pun: `[3, 1, 3, 2, 1]`. Jebakan kedua adalah melempar eksepsi untuk *list* kosong, karena merasa "tidak ada yang perlu dikerjakan". *List* kosong tidak memiliki duplikat, dan jawaban yang benar adalah *list* kosong; eksepsi disediakan untuk keadaan yang tidak memiliki jawaban yang masuk akal, seperti `head()` dari *list* kosong.

> **UJI DIRI.** Apa yang dikembalikan `removeDuplicates([4, 4, 1, 4, 1, 7])`? Dengan Persamaan (8.6) sebagai pedoman, apakah banyaknya elemen yang dijelajahi pemanggilan-pemanggilan `skip` lebih banyak atau lebih sedikit daripada $\frac{6\cdot5}{2}=15$?
>
> *Jawaban.* `[4, 1, 7]`. Lebih sedikit: `skip(4, [4, 1, 4, 1, 7])` menjelajahi 5 elemen dan menyisakan `[1, 1, 7]`; lalu `skip(1, [1, 7])` menjelajahi 2 dan menyisakan `[7]`; lalu `skip(7, [])` menjelajahi 0. Totalnya 7, jauh di bawah 15, karena setiap salinan yang dihapus memperpendek semua penjelajahan sesudahnya.

8.6 Soal Baru yang Dikerjakan: Eksepsi, Berkas, dan Larik

Keenam soal yang tersisa adalah soal baru, dan masing-masing berlatar studi kasus yang menyertai buku ini, Profitina Laundry. Soal-soal ini lebih besar daripada soal *list* karena masing-masing melibatkan beberapa kelas, sehingga setiap soal dimulai dengan metode dari Subbab 8.2.

8.6.1 Soal 8.16 — Pengurai berkas pesanan yang tahan eksepsi

> **SOAL.** Profitina Laundry menyimpan pesanan hariannya dalam sebuah berkas teks, satu pesanan per baris dengan bentuk `id;customer;service;kg`, dengan layanan berupa `WASH`, `IRON`, atau `EXPRESS` dan berat berupa bilangan positif. Baris yang diawali `#` dan baris kosong diabaikan. Tulislah program yang membaca berkas itu, menyimpan setiap pesanan yang sah, dan melaporkan setiap baris yang tidak sah beserta nomor baris dan alasannya, *tanpa berhenti pada baris buruk yang pertama*. Berkas yang tidak ada harus menghasilkan pesan yang jelas, bukan *crash*.

Dengan bahasa kita sendiri. Ada dua hal berbeda yang dapat salah, dan keduanya layak diperlakukan secara berbeda. *Baris yang buruk* adalah hal yang wajar: berkas yang diketik manusia mengandung kekeliruan, dan tanggapan yang tepat adalah mencatat kekeliruan itu lalu melanjutkan. *Berkas yang tidak ada* adalah kegagalan seluruh pekerjaan; tidak ada yang dapat dilanjutkan, sehingga pemanggil yang harus memutuskan apa yang dilakukan.

Rancangan. Langkah 1 sampai 5 dari metode kita menghasilkan tiga kelas dan dua metode (**Tabel 8.17**). Berkas contoh, `laundry_orders.txt`, terletak di samping program dan memuat pesanan yang sah serta satu kekeliruan dari setiap jenis:

```
# id;customer;service;kg
A101;Sari;WASH;3.5
A102;Budi;IRON;2

A103;Dewi;EXPRESS;abc
A104;;WASH;4
A105;Rudi;DRYCLEAN;1.5
A106;Tono;WASH;-2
A107;Ayu;EXPRESS;5
A108;Lina;WASH
```

Tabel 8.17 Rancangan pengurai pesanan: setiap jenis kegagalan dilaporkan oleh metode yang mendeteksinya dan ditangani oleh metode yang dapat berbuat sesuatu terhadapnya.

Kelas atau metode	Tanggung jawab	Kegagalan yang dilaporkannya
`Order`	menyimpan satu pesanan yang sah; imutabel	tidak ada: ia hanya dibangun dari data yang sah
`OrderFormatException`	menyebut baris mana yang buruk dan mengapa	*checked*; memperluas `Exception`
`parseLine(line, lineNo)`	satu baris menjadi satu `Order`	melempar `OrderFormatException`
`readOrders(file, errors)`	seluruh berkas, baris demi baris	mengumpulkan baris buruk; melempar `IOException` untuk berkasnya
`main`	mencetak laporan	menangkap `NoSuchFileException`

Pilihan rancangan yang menentukan adalah *di mana* setiap `try` diletakkan. `try` yang menangkap `OrderFormatException` berada *di dalam* gelung, mengelilingi satu baris, sehingga satu baris buruk hanya mengorbankan satu baris. Berkasnya sendiri dibuka dengan `try`-with-resources (Java) atau `with` (Python), sehingga pasti ditutup apa pun yang terjadi [6], [7].

Penyelesaian Java. **Kode 8.39** adalah program terpanjang dalam bab ini, dan setiap bagiannya adalah gagasan dari Bab 5.

Kode 8.39 `LaundryOrderParser.java`: pengurai baris dengan eksepsi *checked* buatan sendiri, pembaca berkas dengan *try-with-resources*, dan pembaca keliru sebagai pembanding.

```java
// LaundryOrderParser.java - Soal 8.16: pengurai berkas pesanan yang aman terhadap eksepsi.
import java.io.BufferedReader;
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.NoSuchFileException;
import java.nio.file.Path;
import java.util.ArrayList;

public class LaundryOrderParser {

    /** Eksepsi checked: pemanggil parseLine wajib memutuskan tindakannya. */
    static final class OrderFormatException extends Exception {
        private static final long serialVersionUID = 1L;

        OrderFormatException(int lineNo, String reason) {
            super("line " + lineNo + ": " + reason);
        }
    }

    /** Satu pesanan yang sah. Semua field private dan final: pesanan tidak pernah berubah.
*/
    static final class Order {
        private final String id;
        private final String customer;
        private final String service;
        private final double kg;

        Order(String id, String customer, String service, double kg) {
            this.id = id;
            this.customer = customer;
            this.service = service;
            this.kg = kg;
        }

        double getKg() {
            return kg;
        }

        @Override
        public String toString() {
            return id + " " + customer + " " + service + " " + kg + " kg";
        }
    }

    private static final String[] SERVICES = {"WASH", "IRON", "EXPRESS"};
```

Kode 8.39 (Lanjutan) *LaundryOrderParser.java: pengurai baris dengan eksepsi checked buatan sendiri, pembaca berkas dengan try-with-resources, dan pembaca keliru sebagai pembanding.*

```
    /** Mengubah satu baris "id;customer;service;kg" menjadi Order, atau menjelaskan
mengapa tidak bisa. */
    static Order parseLine(String line, int lineNo) throws OrderFormatException {
        String[] parts = line.split(";", -1);          // -1 mempertahankan field kosong
        if (parts.length != 4) {
            throw new OrderFormatException(lineNo, "expected 4 fields, found " +
parts.length);
        }
        String id = parts[0].trim();
        String customer = parts[1].trim();
        String service = parts[2].trim();
        if (customer.isEmpty()) {
            throw new OrderFormatException(lineNo, "customer name is missing");
        }
        if (!java.util.Arrays.asList(SERVICES).contains(service)) {
            throw new OrderFormatException(lineNo, "unknown service '" + service + "'");
        }
        double kg;
        try {
            kg = Double.parseDouble(parts[3].trim());
        } catch (NumberFormatException e) {            // terjemahkan galat tingkat rendah
            throw new OrderFormatException(lineNo, "weight '" + parts[3].trim() + "' is not
a number");
        }
        if (kg <= 0) {
            throw new OrderFormatException(lineNo, "weight must be positive, got " + kg);
        }
        return new Order(id, customer, service, kg);
    }

    /**
     * Membaca setiap pesanan dalam berkas. Baris rusak dilaporkan di errors dan dilewati;
     * berkas yang tidak ada atau tak terbaca menjadi urusan pemanggil (IOException).
     */
    static ArrayList<Order> readOrders(Path file, ArrayList<String> errors) throws
IOException {
        ArrayList<Order> orders = new ArrayList<>();
        try (BufferedReader in = Files.newBufferedReader(file)) {   // ditutup otomatis
            String line;
            int lineNo = 0;
            while ((line = in.readLine()) != null) {
                lineNo++;
                if (line.isBlank() || line.startsWith("#")) {
                    continue;                                       // baris komentar atau
kosong
                }
                try {
                    orders.add(parseLine(line, lineNo));
                } catch (OrderFormatException e) {
                    errors.add(e.getMessage());                     // catat, lalu lanjutkan
                }
```

Kode 8.39 (Lanjutan) *LaundryOrderParser.java: pengurai baris dengan eksepsi checked buatan sendiri, pembaca berkas dengan try-with-resources, dan pembaca keliru sebagai pembanding.*

```
            }
        }
        return orders;
    }

    /** Menggoda tetapi salah: satu try mengelilingi seluruh gelung, sehingga baris rusak
pertama mengakhiri pembacaan. */
    static ArrayList<Order> readOrdersWrong(Path file) throws IOException {
        ArrayList<Order> orders = new ArrayList<>();
        try (BufferedReader in = Files.newBufferedReader(file)) {
            String line;
            int lineNo = 0;
            while ((line = in.readLine()) != null) {
                lineNo++;
                if (!line.isBlank() && !line.startsWith("#")) {
                    orders.add(parseLine(line, lineNo));
                }
            }
        } catch (OrderFormatException e) {
            System.out.println("wrong reader gave up at " + e.getMessage());
        }
        return orders;
    }

    private static int passed = 0;

    private static void check(String name, Object expected, Object actual) {
        if (!java.util.Objects.equals(expected, actual)) {
            throw new AssertionError(name + ": expected " + expected + " but got " +
actual);
        }
        passed++;
        System.out.println("PASS " + name);
    }

    public static void main(String[] args) throws IOException, OrderFormatException {
        String fileName = (args.length > 0) ? args[0] : "laundry_orders.txt";
        ArrayList<String> errors = new ArrayList<>();
        try {
            ArrayList<Order> orders = readOrders(Path.of(fileName), errors);
            double total = 0;
            for (Order o : orders) {
                System.out.println("OK    " + o);
                total += o.getKg();
            }
            for (String e : errors) {
                System.out.println("ERROR " + e);
            }
```

Kode 8.39 (Lanjutan) *LaundryOrderParser.java: pengurai baris dengan eksepsi checked buatan sendiri, pembaca berkas dengan try-with-resources, dan pembaca keliru sebagai pembanding.*

```
            System.out.println(orders.size() + " orders, " + errors.size() + " errors, " +
total + " kg");
            System.out.println("wrong reader kept " +
readOrdersWrong(Path.of(fileName)).size() + " orders");
        } catch (NoSuchFileException e) {
            System.out.println("Cannot find the file " + e.getFile());
        }

        check("valid line", "A101 Sari WASH 3.5 kg", parseLine("A101;Sari;WASH;3.5",
1).toString());
        try {
            parseLine("A103;Dewi;EXPRESS;abc", 5);
            throw new AssertionError("no exception");
        } catch (OrderFormatException e) {
            check("bad number", "line 5: weight 'abc' is not a number", e.getMessage());
        }
        try {
            readOrders(Path.of("no_such_file.txt"), new ArrayList<>());
            throw new AssertionError("no exception");
        } catch (NoSuchFileException e) {
            check("missing file", "no_such_file.txt", e.getFile());
        }
        System.out.println("All " + passed + " checks passed.");
    }
}
```

```
OK    A101 Sari WASH 3.5 kg
OK    A102 Budi IRON 2.0 kg
OK    A107 Ayu EXPRESS 5.0 kg
ERROR line 5: weight 'abc' is not a number
ERROR line 6: customer name is missing
ERROR line 7: unknown service 'DRYCLEAN'
ERROR line 8: weight must be positive, got -2.0
ERROR line 10: expected 4 fields, found 3
3 orders, 5 errors, 10.5 kg
wrong reader gave up at line 5: weight 'abc' is not a number
wrong reader kept 2 orders
PASS valid line
PASS bad number
PASS missing file
All 3 checks passed.
```

Penyelesaian Python. **Kode 8.40** memiliki struktur yang sama; `with open(...)` di Python memainkan peran *try-with-resources*.

Kode 8.40 `laundry_order_parser.py`: pengurai yang sama dalam Python, dengan uji `unittest`.

```
# laundry_order_parser.py - Soal 8.16: pengurai berkas pesanan yang aman terhadap eksepsi.
import sys
import unittest

SERVICES = ("WASH", "IRON", "EXPRESS")


class OrderFormatError(Exception):
    """Satu baris berkas pesanan salah bentuk."""

    def __init__(self, line_no, reason):
        super().__init__(f"line {line_no}: {reason}")


class Order:
    """Satu pesanan yang sah. Atribut 'private menurut konvensi' dan baca-saja."""

    def __init__(self, order_id, customer, service, kg):
        self._id = order_id
        self._customer = customer
        self._service = service
        self._kg = kg

    @property
    def kg(self):
        return self._kg

    def __str__(self):
        return f"{self._id} {self._customer} {self._service} {self._kg} kg"


def parse_line(line, line_no):
    """Mengubah satu baris "id;customer;service;kg" menjadi Order, atau menjelaskan mengapa
tidak bisa."""
    parts = line.split(";")
    if len(parts) != 4:
        raise OrderFormatError(line_no, f"expected 4 fields, found {len(parts)}")
    order_id, customer, service, weight = (p.strip() for p in parts)
    if not customer:
        raise OrderFormatError(line_no, "customer name is missing")
    if service not in SERVICES:
        raise OrderFormatError(line_no, f"unknown service '{service}'")
    try:
        kg = float(weight)
    except ValueError:                          # terjemahkan galat tingkat rendah
        raise OrderFormatError(line_no, f"weight '{weight}' is not a number") from None
    if kg <= 0:
```

Kode 8.40 (Lanjutan) *laundry_order_parser.py: pengurai yang sama dalam Python, dengan uji unittest.*

```
        raise OrderFormatError(line_no, f"weight must be positive, got {kg}")
    return Order(order_id, customer, service, kg)


def read_orders(file_name, errors):
    """Membaca setiap pesanan dalam berkas. Baris rusak dilaporkan di errors dan
    dilewati; berkas yang tidak ada memunculkan FileNotFoundError bagi pemanggil."""
    orders = []
    with open(file_name, encoding="utf-8") as f:      # ditutup otomatis
        for line_no, line in enumerate(f, start=1):
            line = line.rstrip("\n")
            if not line.strip() or line.startswith("#"):
                continue                              # baris komentar atau kosong
            try:
                orders.append(parse_line(line, line_no))
            except OrderFormatError as e:
                errors.append(str(e))                 # catat, lalu lanjutkan
    return orders


class TestParser(unittest.TestCase):
    def test_valid_line(self):
        self.assertEqual("A101 Sari WASH 3.5 kg", str(parse_line("A101;Sari;WASH;3.5", 1)))

    def test_bad_number(self):
        with self.assertRaises(OrderFormatError) as ctx:
            parse_line("A103;Dewi;EXPRESS;abc", 5)
        self.assertEqual("line 5: weight 'abc' is not a number", str(ctx.exception))

    def test_missing_file(self):
        with self.assertRaises(FileNotFoundError):
            read_orders("no_such_file.txt", [])


def main(file_name):
    errors = []
    try:
        orders = read_orders(file_name, errors)
    except FileNotFoundError:
        print("Cannot find the file", file_name)
        return
    for o in orders:
        print("OK   ", o)
    for e in errors:
        print("ERROR", e)
    print(f"{len(orders)} orders, {len(errors)} errors, {sum(o.kg for o in orders)} kg")
```

Kode 8.40 (Lanjutan) *laundry_order_parser.py: pengurai yang sama dalam Python, dengan uji unittest.*

```


if __name__ == "__main__":
    main(sys.argv[1] if len(sys.argv) > 1 else "laundry_orders.txt")
    sys.stdout.flush()
    unittest.main(argv=sys.argv[:1], verbosity=2)
```

```
OK    A101 Sari WASH 3.5 kg
OK    A102 Budi IRON 2.0 kg
OK    A107 Ayu EXPRESS 5.0 kg
ERROR line 5: weight 'abc' is not a number
ERROR line 6: customer name is missing
ERROR line 7: unknown service 'DRYCLEAN'
ERROR line 8: weight must be positive, got -2.0
ERROR line 10: expected 4 fields, found 3
3 orders, 5 errors, 10.5 kg
test_bad_number (__main__.TestParser.test_bad_number) ... ok
test_missing_file (__main__.TestParser.test_missing_file) ... ok
test_valid_line (__main__.TestParser.test_valid_line) ... ok

----------------------------------------------------------------------
Ran 3 tests in 0.000s

OK
```

Baris demi baris. `OrderFormatException` memperluas `Exception`, bukan `RuntimeException`, sehingga ia *checked*: `parseLine` harus menyatakan `throws OrderFormatException`, dan setiap pemanggil harus menangkapnya atau pada gilirannya ikut menyatakannya. Itu disengaja. Baris yang buruk adalah kejadian biasa yang harus dipikirkan pemanggil, dan kini *compiler* memastikan mereka memikirkannya. `line.split(";", -1)` memecah pada titik koma; `-1` mempertahankan *field* kosong di bagian akhir, sehingga baris yang berakhir dengan `;` dihitung dengan benar. Dalam konversi berat, `Double.parseDouble` melempar `NumberFormatException` dari pustaka untuk `abc`; blok `catch` *menerjemahkannya* menjadi eksepsi kita sendiri dengan nomor baris dan kalimat yang dapat ditindaklanjuti manusia, yang jauh lebih berguna daripada `For input string: "abc"`. Di `readOrders`, kepala `try (BufferedReader in = ...)` membuka berkas dan menjamin `in.close()` ketika blok berakhir, baik secara normal maupun karena eksepsi. `Files.newBufferedReader` melempar `NoSuchFileException` (subkelas `IOException`) untuk berkas yang tidak ada, dan `main` mengubahnya menjadi pesan `Cannot find the file ...`: jalankan `java LaundryOrderParser nothing.txt` untuk melihatnya. Laporannya memperlihatkan tiga pesanan sah dengan total 10.5 kg dan lima galat, satu untuk setiap jenis kekeliruan dalam berkas; baris kosong 4 dan baris komentar 1 dilewati tanpa suara. Ketiga pemeriksaan menguji baris yang sah, bilangan yang buruk beserta pesannya yang persis, dan berkas yang tidak ada.

Di Python, `float("abc")` memunculkan `ValueError`, yang kita terjemahkan dengan `raise ... from None`; `from None` menyembunyikan galat aslinya dari *traceback*, karena pesan kita sudah mengatakan segalanya. `open` memunculkan `FileNotFoundError` untuk berkas yang tidak ada. Pemanggilan `unittest.main(argv=sys.argv[:1], ...)` hanya meneruskan nama program ke `unittest`, agar nama berkas yang diberikan pada baris perintah tidak disangka nama uji.

Penyelesaian keliru yang menggoda. `readOrdersWrong` meletakkan satu `try` mengelilingi *seluruh* gelung. Kodenya lebih pendek, dan dapat dikompilasi. Namun baris buruk pertama melempar eksepsi keluar dari gelung, blok `catch` dijalankan, dan pembacaan pun selesai: keluarannya berkata `wrong reader gave up at line 5` dan `wrong reader kept 2 orders`, kehilangan pesanan sah `A107` yang datang kemudian. Kesalahan klasik kedua adalah `catch (Exception e) {}`, penangkap-segala yang kosong yang menyembunyikan setiap galat, termasuk galat pemrograman seperti `NullPointerException`. Tangkaplah eksepsi yang paling spesifik yang dapat Anda tangani, sedekat mungkin dengan tempat Anda dapat menanganinya.

> **UJI DIRI.** Sebuah aturan baru menyatakan bahwa satu pesanan tidak boleh melebihi 20 kg. Di mana pemeriksaannya diletakkan, apa kata laporan untuk baris `A109;Nia;WASH;25`, dan adakah metode lain yang berubah?
>
> *Jawaban.* Di `parseLine`, di samping uji berat positif: `if (kg > 20) throw new OrderFormatException(lineNo, "weight " + kg + " exceeds 20 kg")`. Laporannya akan memuat `ERROR line 11: weight 25.0 exceeds 20 kg` (Python juga mencetak `25.0`). Tidak ada lagi yang berubah, karena hanya `parseLine` yang mengetahui aturan untuk sebuah baris.

8.6.2 Soal 8.17 — Larik terdokumentasi yang tumbuh sendiri

> **SOAL.** Larik Java memiliki panjang yang tetap. Tulislah kelas `GrowableIntArray` yang menyimpan nilai `int` dalam larik biasa dan menggandakan kapasitas larik itu setiap kali penuh. Kelas ini menyediakan `add(value)`, `get(index)`, `size()`, dan `capacity()`. `get` harus menolak setiap indeks di luar `0` sampai `size() - 1` dengan sebuah eksepsi. Dokumentasikan setiap metode publik dengan Javadoc (Python: *docstring*) dan ujilah kelasnya, termasuk batas-batasnya.

Dengan bahasa kita sendiri. Beginilah `ArrayList` bekerja di dalamnya (Bab 5), dibangun dengan tangan. Kelas ini membedakan dua bilangan: *ukuran* (*size*), yaitu berapa nilai yang telah ditambahkan pemakai, dan *kapasitas* (*capacity*), yaitu berapa sel yang dimiliki larik di baliknya. Hanya `size` sel pertama yang menyimpan data pemakai; sisanya ruang cadangan.

Rancangan. Dua *field private*, `int[] data` dan `int size`, serta satu aturan, yang disebut *invarian kelas* (*class invariant*) karena setiap metode publik harus mempertahankannya:

$$0 \le size \le capacity = \texttt{data.length}, \quad \text{dan hanya } \texttt{data}[0..size-1] \text{ yang menyimpan nilai.} \tag{8.7}$$

Tabel 8.18 mendaftar apa yang dilakukan setiap metode untuk menjaga Persamaan (8.7) tetap benar.

Tabel 8.18 Rancangan `GrowableIntArray`. Prasyarat `get` diperiksa terhadap `size`, bukan terhadap panjang larik.

Metode	Prasyarat	Yang dikerjakannya	Eksepsi
konstruktor	kapasitas ≥ 1	`data = new int[capacity]`, `size = 0`	`IllegalArgumentException`
`add(v)`	tidak ada	bila penuh, salin ke larik dua kali lebih panjang; lalu simpan, `size++`	tidak ada
`get(i)`	$0 \le i < size$	kembalikan `data[i]`	`IndexOutOfBoundsException`
`size()`, `capacity()`	tidak ada	melaporkan kedua bilangan itu	tidak ada

Penyelesaian Java. Pada **Kode 8.41**, setiap anggota publik memiliki komentar Javadoc. Menjalankan `javadoc -d docs GrowableIntArray.java` mengubahnya menjadi halaman web; dengan opsi `-quiet`, perintah itu tidak mencetak apa pun untuk berkas ini, yang berarti tidak ada komentar yang kekurangan tag `@param`, `@return`, atau `@throws`.

Kode 8.41 `GrowableIntArray.java`: larik yang menggandakan diri, dengan Javadoc lengkap dan pemeriksaan batas.

```java
// GrowableIntArray.java - Soal 8.17: larik terdokumentasi dan teruji yang tumbuh sendiri.
import java.util.Arrays;

/**
 * Daftar nilai {@code int} yang disimpan dalam larik biasa yang menggandakan
 * kapasitasnya setiap kali penuh. Hanya {@link #size()} sel pertama yang berisi data.
 */
public class GrowableIntArray {

    private int[] data;
    private int size;

    /**
     * Membuat larik kosong dengan ruang untuk {@code initialCapacity} nilai.
     *
     * @param initialCapacity kapasitas awal, paling sedikit 1
     * @throws IllegalArgumentException bila {@code initialCapacity < 1}
     */
    public GrowableIntArray(int initialCapacity) {
        if (initialCapacity < 1) {
            throw new IllegalArgumentException("capacity must be at least 1: " +
initialCapacity);
        }
        data = new int[initialCapacity];
        size = 0;
    }

    /**
     * Menambahkan {@code value} di akhir, menggandakan kapasitas lebih dahulu bila perlu.
     *
     * @param value nilai yang ditambahkan
     */
    public void add(int value) {
        if (size == data.length) {
            data = Arrays.copyOf(data, 2 * data.length);   // larik baru yang lebih besar
        }
        data[size] = value;
        size++;
    }
```

Kode 8.41 (Lanjutan) *GrowableIntArray.java: larik yang menggandakan diri, dengan Javadoc lengkap dan pemeriksaan batas.*

```
    /**
     * Mengembalikan nilai pada posisi {@code index}.
     *
     * @param index posisi dari 0 sampai {@code size() - 1}
     * @return nilai yang tersimpan di sana
     * @throws IndexOutOfBoundsException bila {@code index} di luar rentang itu
     */
    public int get(int index) {
        if (index < 0 || index >= size) {
            throw new IndexOutOfBoundsException("index " + index + ", size " + size);
        }
        return data[index];
    }

    /** Menggoda tetapi salah: {@code >} alih-alih {@code >=} meloloskan index == size. */
    int getWrong(int index) {
        if (index < 0 || index > size) {
            throw new IndexOutOfBoundsException("index " + index + ", size " + size);
        }
        return data[index];
    }

    /**
     * Memberi tahu berapa banyak nilai yang sudah ditambahkan.
     *
     * @return banyaknya nilai yang tersimpan
     */
    public int size() {
        return size;
    }

    /**
     * Memberi tahu berapa banyak nilai yang muat sebelum penggandaan berikutnya.
     *
     * @return panjang larik di baliknya
     */
    public int capacity() {
        return data.length;
    }

    @Override
    public String toString() {
        return Arrays.toString(Arrays.copyOf(data, size));  // hanya bagian yang terpakai
    }
```

Kode 8.41 (Lanjutan) *GrowableIntArray.java: larik yang menggandakan diri, dengan Javadoc lengkap dan pemeriksaan batas.*

```
    private static int passed = 0;

    private static void check(String name, Object expected, Object actual) {
        if (!java.util.Objects.equals(expected, actual)) {
            throw new AssertionError(name + ": expected " + expected + " but got " +
actual);
        }
        passed++;
        System.out.println("PASS " + name);
    }

    /**
     * Memperagakan kelas ini dan menjalankan pemeriksaannya.
     *
     * @param args tidak dipakai
     */
    public static void main(String[] args) {
        GrowableIntArray a = new GrowableIntArray(2);
        for (int kg = 3; kg <= 7; kg++) {
            a.add(kg);
            System.out.println("add(" + kg + ") -> " + a + "  size " + a.size() + ",
capacity " + a.capacity());
        }
        check("values", "[3, 4, 5, 6, 7]", a.toString());
        check("get(0)", 3, a.get(0));
        check("get(4)", 7, a.get(4));
        check("capacity doubled twice", 8, a.capacity());
        for (int bad : new int[] {5, -1}) {
            try {
                a.get(bad);
                throw new AssertionError("no exception for " + bad);
            } catch (IndexOutOfBoundsException e) {
                check("get(" + bad + ") rejected", "index " + bad + ", size 5",
e.getMessage());
            }
        }
        try {
            new GrowableIntArray(0);
        } catch (IllegalArgumentException e) {
            check("capacity 0 rejected", "capacity must be at least 1: 0", e.getMessage());
        }
        System.out.println("getWrong(5) = " + a.getWrong(5) + "   <- a value that was never
added");
        System.out.println("All " + passed + " checks passed.");
    }
}
```

```
add(3) -> [3]  size 1, capacity 2
add(4) -> [3, 4]  size 2, capacity 2
add(5) -> [3, 4, 5]  size 3, capacity 4
add(6) -> [3, 4, 5, 6]  size 4, capacity 4
add(7) -> [3, 4, 5, 6, 7]  size 5, capacity 8
PASS values
PASS get(0)
PASS get(4)
PASS capacity doubled twice
PASS get(5) rejected
PASS get(-1) rejected
PASS capacity 0 rejected
getWrong(5) = 0   <- a value that was never added
All 7 checks passed.
```

Penyelesaian Python. **Kode 8.42** meniru blok berukuran tetap dengan `[0] * capacity`, karena *list* Python akan tumbuh sendiri dan menyembunyikan pelajarannya. *Docstring*-nya mengikuti tata letak "Raises:" yang lazim, dan `help(GrowableIntArray.get)` mencetaknya.

Kode 8.42 `growable_int_array.py`: kelas yang sama dalam Python, dengan *docstring* dan metode `setUp` yang dipakai bersama oleh uji-ujinya.

```
# growable_int_array.py - Soal 8.17: larik terdokumentasi dan teruji yang tumbuh sendiri.
import sys
import unittest


class GrowableIntArray:
    """Daftar int yang disimpan dalam blok berukuran tetap yang menggandakan kapasitasnya
    setiap kali penuh. Hanya size() sel pertama yang berisi data."""

    def __init__(self, initial_capacity):
        """Membuat larik kosong dengan ruang untuk initial_capacity nilai.

        Raises:
            ValueError: bila initial_capacity < 1.
        """
        if initial_capacity < 1:
            raise ValueError(f"capacity must be at least 1: {initial_capacity}")
        self._data = [0] * initial_capacity     # blok tetap, seperti new int[n]
        self._size = 0

    def add(self, value):
        """Menambahkan value di akhir, menggandakan kapasitas lebih dahulu bila perlu."""
        if self._size == len(self._data):
            self._data = self._data + [0] * len(self._data)   # blok baru yang lebih besar
        self._data[self._size] = value
        self._size += 1

    def get(self, index):
        """Mengembalikan nilai pada posisi index (0 sampai size() - 1).

        Raises:
            IndexError: bila index di luar rentang itu.
        """
        if index < 0 or index >= self._size:
            raise IndexError(f"index {index}, size {self._size}")
        return self._data[index]

    def size(self):
        """Berapa banyak nilai yang sudah ditambahkan."""
        return self._size

    def capacity(self):
        """Berapa banyak nilai yang muat sebelum penggandaan berikutnya."""
        return len(self._data)
```

Kode 8.42 (Lanjutan) *growable_int_array.py: kelas yang sama dalam Python, dengan docstring dan metode setUp yang dipakai bersama oleh uji-ujinya.*

```
    def __str__(self):
        return str(self._data[:self._size])        # hanya bagian yang terpakai


class TestGrowableIntArray(unittest.TestCase):
    def setUp(self):                                # dijalankan sebelum setiap uji
        self.a = GrowableIntArray(2)
        for kg in range(3, 8):
            self.a.add(kg)

    def test_values_and_capacity(self):
        self.assertEqual("[3, 4, 5, 6, 7]", str(self.a))
        self.assertEqual(7, self.a.get(4))
        self.assertEqual(8, self.a.capacity())

    def test_bad_index(self):
        for bad in (5, -1):
            with self.assertRaises(IndexError):
                self.a.get(bad)

    def test_bad_capacity(self):
        with self.assertRaises(ValueError):
            GrowableIntArray(0)


if __name__ == "__main__":
    a = GrowableIntArray(2)
    for kg in range(3, 8):
        a.add(kg)
        print(f"add({kg}) -> {a}  size {a.size()}, capacity {a.capacity()}")
    sys.stdout.flush()
    unittest.main(verbosity=2)
```

```
add(3) -> [3]  size 1, capacity 2
add(4) -> [3, 4]  size 2, capacity 2
add(5) -> [3, 4, 5]  size 3, capacity 4
add(6) -> [3, 4, 5, 6]  size 4, capacity 4
add(7) -> [3, 4, 5, 6, 7]  size 5, capacity 8
test_bad_capacity (__main__.TestGrowableIntArray.test_bad_capacity) ... ok
test_bad_index (__main__.TestGrowableIntArray.test_bad_index) ... ok
test_values_and_capacity (__main__.TestGrowableIntArray.test_values_and_capacity) ... ok

----------------------------------------------------------------------
Ran 3 tests in 0.000s

OK
```

Baris demi baris. Lima baris keluaran pertama memperlihatkan penggandaan yang sedang bekerja: kapasitas 2 penuh pada `add` kedua, menjadi 4 pada yang ketiga, dan 8 pada yang kelima. `Arrays.copyOf(data, 2 * data.length)` menciptakan larik baru yang dua kali lebih panjang, menyalin nilai-nilai lama ke dalamnya, dan mengisi sisanya dengan nol; larik lama kemudian menjadi sampah. Karena kapasitasnya berlipat dua, n penambahan hanya menyebabkan sekitar $\log_2 n$ penyalinan, dan total pekerjaan menyalin tetap di bawah $2n$; itulah sebabnya `ArrayList.add` rata-rata cepat. Pemeriksaannya mencakup nilai-nilai biasa, kedua ujung rentang yang sah (`get(0)` dan `get(4)`), kedua sisi yang tepat di luarnya (`get(5)` dan `get(-1)`), dan aturan konstruktor itu sendiri. Di Python, metode `setUp` dijalankan sebelum *setiap* uji, sehingga setiap uji dimulai dari larik lima elemen yang sama; inilah padanan `unittest` untuk `@BeforeEach` milik JUnit.

Penyelesaian keliru yang menggoda. `getWrong` menguji `index > size` alih-alih `index >= size`. Untuk `get(5)` pada larik lima elemen, pemeriksaannya lolos (5 tidak lebih besar daripada 5), dan `data[5]` dibaca. Tidak terjadi *crash*, karena lariknya berkapasitas 8, sehingga sel 5 memang ada; sel itu hanya berisi 0 yang tidak pernah ditambahkan siapa pun. Baris keluaran `getWrong(5) = 0 <- a value that was never added` adalah jawaban salah yang *diam*, jenis yang paling berbahaya. `ArrayIndexOutOfBoundsException` bawaan Java tidak dapat menolong, karena ia memeriksa terhadap kapasitas, bukan terhadap ukuran. Hanya uji tepat pada `index == size` yang dapat menangkapnya.

> **UJI DIRI.** Dimulai dengan kapasitas 1, berapa kali larik tumbuh selama 100 nilai ditambahkan, dan berapa kapasitasnya di akhir?
>
> *Jawaban.* Larik tumbuh ketika menambahkan nilai ke-2, ke-3, ke-5, ke-9, ke-17, ke-33, dan ke-65: 7 kali, dan di akhir kapasitasnya $2^7 = 128$.

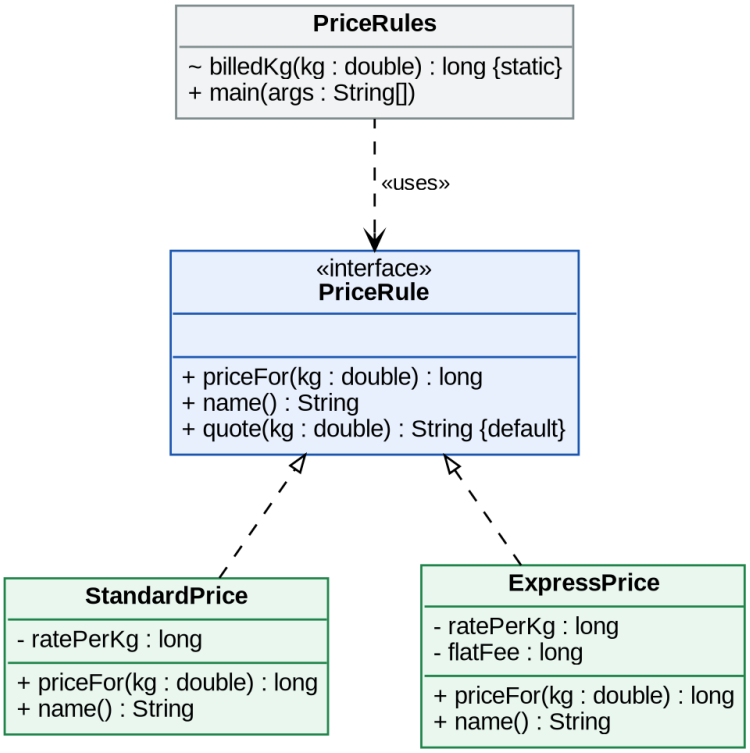
8.7 Soal Baru yang Dikerjakan: Antarmuka dan Ujinya

8.7.1 Soal 8.18 — Satu antarmuka, dua implementasi, satu set uji

> **SOAL.** Profitina Laundry menghargai pesanan menurut beratnya. Aturan *standar* mengenakan Rp7.000 per kilogram; aturan *ekspres* mengenakan Rp12.000 per kilogram ditambah biaya tetap Rp5.000. Setiap kilogram yang sudah dimulai ditagih penuh (2,3 kg ditagih sebagai 3 kg), dan berat nol atau kurang adalah galat. Definisikan antarmuka `PriceRule` dengan kedua aturan itu sebagai implementasinya, beri antarmuka itu metode `quote(kg)` yang menghasilkan penawaran harga yang dapat dicetak untuk aturan mana pun, dan ujilah kedua implementasi.

Dengan bahasa kita sendiri. Bagian lain program seharusnya dapat memegang "suatu aturan harga" tanpa mengetahui aturan yang mana; justru untuk itulah antarmuka ada (Bab 6). Bagian yang sama, yaitu pembulatan ke atas dan penolakan berat yang buruk, harus ditulis sekali saja.

Rancangan. **Gambar 8.7** memperlihatkan rancangannya sebagai diagram kelas UML. Antarmuka itu menyatakan dua metode abstrak, `priceFor` dan `name`, dan satu *default method* (metode bawaan), `quote`, yang diimplementasikan di dalam antarmuka itu sendiri dengan memakai kedua metode yang lain. Setiap implementasi menyimpan tarifnya dalam *field* `private final`.



Gambar 8.7 Diagram kelas UML untuk Soal 8.18. Garis putus-putus dengan segitiga berongga berarti "*implements*"; `quote` adalah *default method* dari antarmuka itu.

Sebelum menulis kode, tulislah nilai-nilai yang diharapkan dengan tangan. Standar, 2,3 kg: $\lceil 2.3 \rceil = 3$ kg, $3 \times 7000 = 21000$. Ekspres, 2,3 kg: $3 \times 12000 + 5000 = 41000$. Standar, 0,1 kg: $\lceil 0.1 \rceil = 1$ kg, 7000. Di sini $\lceil x \rceil$, *ceiling* (pembulatan ke atas) dari x, adalah bilangan bulat terkecil yang tidak kurang dari x.

Penyelesaian Java. **Kode 8.43** menempatkan antarmuka, kedua implementasi, dan pemeriksaannya dalam satu berkas.

Kode 8.43 `PriceRules.java`: antarmuka dengan *default method*, dua implementasi, dan uji yang memperlakukan keduanya melalui antarmuka.

```java
// PriceRules.java - Soal 8.18: satu antarmuka, dua implementasi, satu himpunan uji.
public class PriceRules {

    /** Cara layanan laundry mengubah berat menjadi harga dalam rupiah. */
    interface PriceRule {
        /** Harga untuk kg kilogram; setiap kilogram yang dimulai ditagih penuh. */
        long priceFor(double kg);

        String name();

        /** Metode default: ditulis sekali, berdasarkan dua metode abstrak. */
        default String quote(double kg) {
            return name() + " " + kg + " kg = Rp" + priceFor(kg);
        }
    }

    /** Kilogram utuh yang ditagih, dengan validasi yang dipakai bersama kedua aturan. */
    static long billedKg(double kg) {
        if (kg <= 0) {
            throw new IllegalArgumentException("weight must be positive: " + kg);
        }
        return (long) Math.ceil(kg);              // 2.3 kg ditagih sebagai 3 kg
    }

    static final class StandardPrice implements PriceRule {
        private final long ratePerKg;

        StandardPrice(long ratePerKg) {
            this.ratePerKg = ratePerKg;
        }

        @Override
        public long priceFor(double kg) {
            return ratePerKg * billedKg(kg);
        }

        @Override
        public String name() {
            return "standard";
        }
    }

    static final class ExpressPrice implements PriceRule {
        private final long ratePerKg;
        private final long flatFee;
```

Kode 8.43 (Lanjutan) *PriceRules.java: antarmuka dengan default method, dua implementasi, dan uji yang memperlakukan keduanya melalui antarmuka.*

```
        ExpressPrice(long ratePerKg, long flatFee) {
            this.ratePerKg = ratePerKg;
            this.flatFee = flatFee;
        }

        @Override
        public long priceFor(double kg) {
            return ratePerKg * billedKg(kg) + flatFee;
        }

        @Override
        public String name() {
            return "express";
        }
    }

    /** Menggoda tetapi salah: kalikan dahulu, lalu potong desimalnya. */
    static long standardPriceWrong(long ratePerKg, double kg) {
        return (long) (ratePerKg * kg);
    }

    private static int passed = 0;

    private static void check(String name, Object expected, Object actual) {
        if (!java.util.Objects.equals(expected, actual)) {
            throw new AssertionError(name + ": expected " + expected + " but got " +
actual);
        }
        passed++;
        System.out.println("PASS " + name);
    }

    public static void main(String[] args) {
        PriceRule[] rules = {new StandardPrice(7000), new ExpressPrice(12000, 5000)};
        for (PriceRule rule : rules) {              // gelung hanya mengenal antarmukanya
            System.out.println(rule.quote(2.3));
        }
        System.out.println("wrong standard 2.3 kg = Rp" + standardPriceWrong(7000, 2.3));
```

Kode 8.43 (Lanjutan) *PriceRules.java: antarmuka dengan default method, dua implementasi, dan uji yang memperlakukan keduanya melalui antarmuka.*

```
        PriceRule standard = rules[0];
        PriceRule express = rules[1];
        check("standard exact kg", 21000L, standard.priceFor(3.0));
        check("standard rounds up", 21000L, standard.priceFor(2.3));
        check("express adds fee", 41000L, express.priceFor(2.3));
        check("smallest bill", 7000L, standard.priceFor(0.1));
        for (PriceRule rule : rules) {
            try {
                rule.priceFor(0);
                throw new AssertionError("no exception");
            } catch (IllegalArgumentException e) {
                check(rule.name() + " rejects 0 kg", "weight must be positive: 0.0",
e.getMessage());
            }
        }
        System.out.println("All " + passed + " checks passed.");
    }
}
```

```
standard 2.3 kg = Rp21000
express 2.3 kg = Rp41000
wrong standard 2.3 kg = Rp16099
PASS standard exact kg
PASS standard rounds up
PASS express adds fee
PASS smallest bill
PASS standard rejects 0 kg
PASS express rejects 0 kg
All 6 checks passed.
```

Penyelesaian Python. Padanan antarmuka Java di Python adalah *abstract base class* (kelas dasar abstrak): kelas yang mewarisi `ABC` dan menandai metode dengan `@abstractmethod` tidak dapat diinstansiasi, dan subkelasnya harus meng-*override* metode-metode itu [8]. **Kode 8.44** juga memperlihatkan pola pengujian yang berguna: sebuah kelas *kontrak*, `PriceRuleContract`, menyimpan uji yang harus dilalui setiap `PriceRule`, dan setiap implementasi mendapat kelas uji kecil yang mencampurkan kontrak itu dan menyatakan cara membangun aturannya.

Kode 8.44 `price_rules.py`: rancangan yang sama dengan *abstract base class*, dan uji kontrak yang dapat dipakai ulang dan dijalankan terhadap kedua implementasi.

```python
# price_rules.py - Soal 8.18: satu antarmuka, dua implementasi, satu himpunan uji.
import math
import sys
import unittest
from abc import ABC, abstractmethod


class PriceRule(ABC):
    """Cara layanan laundry mengubah berat menjadi harga dalam rupiah."""

    @abstractmethod
    def price_for(self, kg):
        """Harga untuk kg kilogram; setiap kilogram yang dimulai ditagih penuh."""

    @abstractmethod
    def name(self):
        """Nama pendek untuk dicetak."""

    def quote(self, kg):
        """Metode konkret yang ditulis sekali, berdasarkan dua metode abstrak."""
        return f"{self.name()} {kg} kg = Rp{self.price_for(kg)}"


def billed_kg(kg):
    """Kilogram utuh yang ditagih, dengan validasi yang dipakai bersama kedua aturan."""
    if kg <= 0:
        raise ValueError(f"weight must be positive: {kg}")
    return math.ceil(kg)                    # 2.3 kg ditagih sebagai 3 kg


class StandardPrice(PriceRule):
    def __init__(self, rate_per_kg):
        self._rate_per_kg = rate_per_kg

    def price_for(self, kg):
        return self._rate_per_kg * billed_kg(kg)

    def name(self):
        return "standard"


class ExpressPrice(PriceRule):
    def __init__(self, rate_per_kg, flat_fee):
        self._rate_per_kg = rate_per_kg
        self._flat_fee = flat_fee
```

Kode 8.44 (Lanjutan) *price_rules.py: rancangan yang sama dengan abstract base class, dan uji kontrak yang dapat dipakai ulang dan dijalankan terhadap kedua implementasi.*

```
    def price_for(self, kg):
        return self._rate_per_kg * billed_kg(kg) + self._flat_fee

    def name(self):
        return "express"


class PriceRuleContract:
    """Uji yang wajib dilulusi setiap PriceRule; dicampurkan ke satu TestCase per kelas."""

    def make_rule(self):
        raise NotImplementedError

    def test_rejects_zero(self):
        with self.assertRaises(ValueError):
            self.make_rule().price_for(0)

    def test_rounds_up(self):
        rule = self.make_rule()
        self.assertEqual(rule.price_for(3.0), rule.price_for(2.3))


class TestStandard(PriceRuleContract, unittest.TestCase):
    def make_rule(self):
        return StandardPrice(7000)

    def test_prices(self):
        self.assertEqual(21000, self.make_rule().price_for(2.3))
        self.assertEqual(7000, self.make_rule().price_for(0.1))


class TestExpress(PriceRuleContract, unittest.TestCase):
    def make_rule(self):
        return ExpressPrice(12000, 5000)

    def test_prices(self):
        self.assertEqual(41000, self.make_rule().price_for(2.3))


class TestAbstract(unittest.TestCase):
    def test_cannot_instantiate_interface(self):
        with self.assertRaises(TypeError):
            PriceRule()
```

Kode 8.44 (Lanjutan) *price_rules.py: rancangan yang sama dengan abstract base class, dan uji kontrak yang dapat dipakai ulang dan dijalankan terhadap kedua implementasi.*

```
if __name__ == "__main__":
    for rule in [StandardPrice(7000), ExpressPrice(12000, 5000)]:
        print(rule.quote(2.3))
    print("wrong standard 2.3 kg = Rp" + str(int(7000 * 2.3)))
    sys.stdout.flush()
    unittest.main(verbosity=2)
```

```
standard 2.3 kg = Rp21000
express 2.3 kg = Rp41000
wrong standard 2.3 kg = Rp16099
test_cannot_instantiate_interface (__main__.TestAbstract.test_cannot_instantiate_interface)
... ok
test_prices (__main__.TestExpress.test_prices) ... ok
test_rejects_zero (__main__.TestExpress.test_rejects_zero) ... ok
test_rounds_up (__main__.TestExpress.test_rounds_up) ... ok
test_prices (__main__.TestStandard.test_prices) ... ok
test_rejects_zero (__main__.TestStandard.test_rejects_zero) ... ok
test_rounds_up (__main__.TestStandard.test_rounds_up) ... ok

----------------------------------------------------------------------
Ran 7 tests in 0.000s

OK
```

Baris demi baris. Di `main`, lariknya bertipe `PriceRule[]`, dan variabel gelungnya bertipe `PriceRule`: gelung itu tidak tahu dengan kelas mana ia berbicara, dan `rule.quote(2.3)` tetap memanggil `priceFor` yang benar, karena objeklah yang memutuskan, bukan variabelnya. *Default method* `quote` ditulis satu kali untuk setiap aturan, yang sekarang maupun yang akan datang. Metode bantu `billedKg` memegang kedua aturan bersama, yaitu pembulatan ke atas dengan `Math.ceil` dan penolakan berat yang tidak positif, sehingga tidak ada implementasi yang mengulangnya. Uang disimpan dalam rupiah bertipe `long`, tidak pernah dalam `double`; bilangan bulat itu eksak. Eksekusi Python memperlihatkan tujuh uji dari empat kelas: kedua uji kontrak dijalankan satu kali untuk setiap implementasi, dan itulah inti pola ini, sedangkan `test_cannot_instantiate_interface` memastikan bahwa `PriceRule()` sendiri memunculkan `TypeError`.

Penyelesaian keliru yang menggoda. `standardPriceWrong` menghitung `(long) (ratePerKg * kg)`. Ada dua *bug* di sana, dan keluarannya memperlihatkan jumlah keduanya: `Rp16099`. Yang pertama *bug* rancangan: ia menagih 2,3 kg, bukan 3 kg. Yang kedua *bug* bilangan titik mengambang: 7000×2.3 tidak persis 16100 dalam titik mengambang biner, melainkan 16099.999999999998, dan *cast* ke `long` *memotong* ke arah nol, turun menjadi 16099. Aturannya: ubah berat menjadi satuan tagihan yang bulat lebih dulu (`Math.ceil`), dan baru sesudah itu kalikan dengan tarif bilangan bulat.

8.7.2 Uji yang sama dalam JUnit 6

Metode bantu `check` dalam bab ini mengerjakan, dalam ukuran mini, apa yang dikerjakan JUnit 6 [9]. **Tabel 8.19** memberikan padanannya. Folder bab ini memuat versi JUnit dari dua program penguji, di dalam subfolder `junit5` (nama yang dipertahankan karena API-nya berasal dari JUnit 5): `junit5/PriceRulesTest.java` dan `junit5/GrowableIntArrayTest.java`; setiap pemeriksaan telah menjadi metode yang ditandai `@Test`.

Tabel 8.19 Bagaimana program-program yang memeriksa dirinya sendiri dalam bab ini dipetakan ke JUnit 6 dan `unittest`.

Bab ini (Java biasa)	JUnit 6	`unittest` Python
`check("name", expected, actual)`	`assertEquals(expected, actual)` dalam metode `@Test`	`self.assertEqual(expected, actual)`
`try { ...; throw new AssertionError(); } catch (X e) {...}`	`X e = assertThrows(X.class, () -> ...)`	`with self.assertRaises(X):`
kode di awal `main`	metode yang ditandai `@BeforeEach`	`setUp(self)`
`All N checks passed.`	ringkasan dari *runner*	`Ran N tests ... OK`

Untuk menjalankannya, Anda memerlukan "*console launcher*" JUnit 6, satu berkas bernama `junit-platform-console-standalone-<version>.jar`, yang diunduh sekali dari situs web JUnit atau dari Maven Central. Dari folder `java` bab ini:

```
javac -d out -cp junit-platform-console-standalone.jar *.java junit5/*.java
java -jar junit-platform-console-standalone.jar execute --class-path out --scan-class-
path
```

Perintah pertama mengompilasi program dan uji ke dalam folder `out`; perintah kedua menemukan setiap metode `@Test` lalu menjalankannya, dengan mencetak pohon hasil dan ringkasannya. Di VS Code, Extension Pack for Java (Bab 1) mengenali metode `@Test` begitu JAR itu ada di *class path* proyek, dan menampilkan tombol jalankan di samping masing-masing. *Launcher* itu tidak dapat diunduh di mesin yang menyusun buku ini, sehingga kedua berkas JUnit dikompilasi terhadap pengganti minimal untuk anotasi JUnit 6, dan sebuah *runner* kecil melaporkan kedelapan metode `@Test` lolos. Itu bukan JUnit yang sesungguhnya, sehingga tidak ada keluaran JUnit yang dikutip di sini; perintah-perintah di atas akan menampilkan laporan JUnit yang asli.

> **UJI DIRI.** Tambahkan aturan ketiga, `IronOnlyPrice`, Rp5.000 per kilogram dengan tagihan minimum Rp10.000. Berapa tagihannya untuk 1,2 kg dan untuk 2,5 kg, dan metode apa saja yang harus didefinisikan kelas itu?
>
> *Jawaban.* Kelas itu mendefinisikan `priceFor` dan `name`; `quote` didapat cuma-cuma dari antarmuka. 1,2 kg: $\lceil 1.2 \rceil = 2$, $2 \times 5000 = 10000$, sama dengan minimumnya: Rp10.000. 2,5 kg: $3 \times 5000 = 15000$. Di Python, menambahkan `class TestIronOnly(PriceRuleContract, unittest.TestCase)` menjalankan seluruh kontrak terhadapnya tanpa biaya tambahan.

8.8 Soal Baru yang Dikerjakan: Pewarisan, Polimorfisme, dan Kontrol Akses

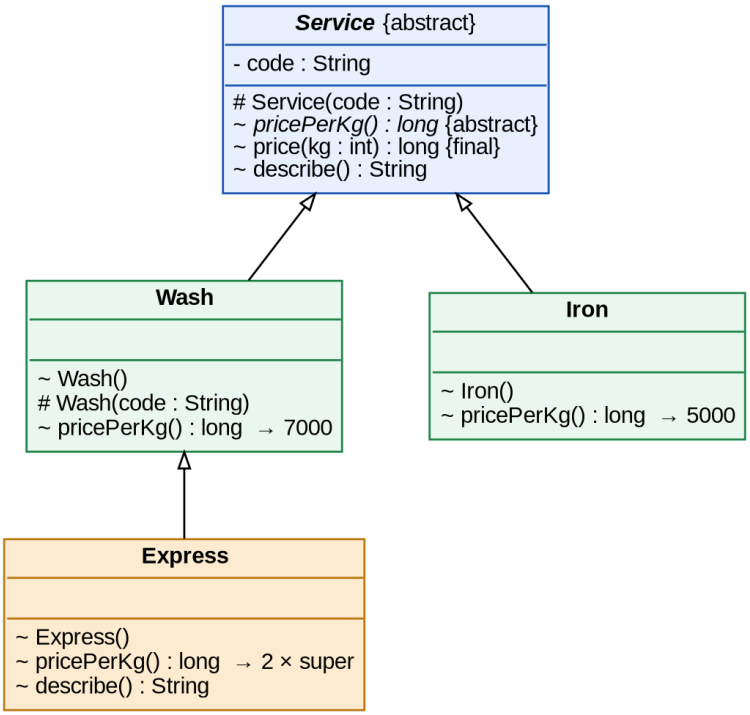
8.8.1 Soal 8.19 — Hierarki kelas yang ditelusuri dengan tangan

> **SOAL.** Profitina Laundry memiliki tiga layanan: `Wash` (Rp7.000 per kg), `Iron` (Rp5.000 per kg), dan `Express`, yaitu cuci khusus dengan tarif dua kali tarif cuci yang deskripsinya menambahkan "(same day)". Ketiganya berbagi superkelas abstrak `Service` dengan metode *final* `price(kg)` dan metode `describe()`. Ada juga dua metode statis yang di-*overload*, `label(Service)` dan `label(Wash)`. Sebelum menjalankan **Kode 8.45**, perkirakan dengan tangan setiap baris yang dicetaknya, dan jelaskan setiap perkiraan.

Dengan bahasa kita sendiri. Soal ini tentang dua saat yang berbeda ketika Java memilih metode mana yang dijalankan. *Overloading* (Bab 6) diselesaikan oleh *compiler*, berdasarkan tipe argumen yang *dideklarasikan*. *Overriding* (Bab 7) diselesaikan ketika program berjalan, berdasarkan kelas

sebenarnya dari objek. Mencampuradukkan keduanya adalah salah paham tentang polimorfisme yang paling umum [10].

Rancangan. **Gambar 8.8** memperlihatkan hierarkinya. `price` bersifat `final` di `Service`, sehingga tidak ada subkelas yang dapat mengubahnya; metode ini memanggil `pricePerKg` yang abstrak, yang disediakan oleh setiap subkelas. Metode di superkelas yang menetapkan urutan langkah dan membiarkan subkelas mengisi satu langkah disebut pola *template method* [11].



Gambar 8.8 Hierarki layanan pada Soal 8.19. `Express` memperluas `Wash`, meng-*override* `pricePerKg` dan `describe`, dan memakai `super` untuk menjangkau versi milik `Wash`.

Kode 8.45 `ServiceDispatch.java`: kelas abstrak, tiga subkelas, *overriding* dengan `super`, dan dua *overload* dari `label`.

```java
// ServiceDispatch.java - Soal 8.19: hierarki kelas yang ditelusuri dengan tangan.
public class ServiceDispatch {

    abstract static class Service {
        private final String code;

        protected Service(String code) {
            this.code = code;
        }

        abstract long pricePerKg();

        /** Kode yang sama untuk setiap layanan; pricePerKg() dipilih saat program
berjalan. */
        final long price(int kg) {
            return pricePerKg() * kg;
        }

        String describe() {
            return code + " @ " + pricePerKg() + "/kg";
        }
    }

    static class Wash extends Service {
        Wash() {
            this("WASH");
        }

        protected Wash(String code) {           // memungkinkan subkelas memilih kodenya
            super(code);
        }

        @Override
        long pricePerKg() {
            return 7000;
        }
    }

    static class Iron extends Service {
        Iron() {
            super("IRON");
        }
```

Kode 8.45 (Lanjutan) *ServiceDispatch.java: kelas abstrak, tiga subkelas, overriding dengan super, dan dua overload dari label.*

```
        @Override
        long pricePerKg() {
            return 5000;
        }
    }

    static class Express extends Wash {
        Express() {
            super("EXPRESS");
        }

        @Override
        long pricePerKg() {
            return 2 * super.pricePerKg();      // dua kali tarif Wash
        }

        @Override
        String describe() {
            return super.describe() + " (same day)";
        }
    }

    // Dua OVERLOAD: kompilator memilih satu berdasarkan tipe argumen yang dideklarasikan.
    static String label(Service s) {
        return "service: " + s.describe();
    }

    static String label(Wash w) {
        return "washing: " + w.describe();
    }

    private static int passed = 0;

    private static void check(String name, Object expected, Object actual) {
        if (!java.util.Objects.equals(expected, actual)) {
            throw new AssertionError(name + ": expected " + expected + " but got " +
actual);
        }
        passed++;
        System.out.println("PASS " + name);
    }
```

Kode 8.45 (Lanjutan) *ServiceDispatch.java: kelas abstrak, tiga subkelas, overriding dengan super, dan dua overload dari label.*

```
    public static void main(String[] args) {
        Service[] services = {new Wash(), new Iron(), new Express()};
        for (Service s : services) {                          // baris A
            System.out.println(s.describe() + ", 3 kg = " + s.price(3));
        }
        Service s = new Express();                            // baris B
        Wash w = new Express();                               // baris C
        System.out.println(label(s));                         // baris D
        System.out.println(label(w));                         // baris E
        System.out.println(label((Wash) s));                  // baris F

        check("express price", 42000L, services[2].price(3));
        check("overload by declared type", "service: EXPRESS @ 14000/kg (same day)",
label(s));
        check("cast changes the overload", "washing: EXPRESS @ 14000/kg (same day)",
label((Wash) s));
        check("express is a wash", true, s instanceof Wash);
        System.out.println("All " + passed + " checks passed.");
    }
}
```

Penelusuran dengan tangan. **Tabel 8.20** adalah perkiraannya, dibuat sebelum program dijalankan. Untuk setiap pemanggilan, kolom kedua adalah tipe variabel yang *dideklarasikan*, yang diketahui *compiler*; kolom ketiga adalah kelas dari *objek*-nya, yang baru diketahui saat program berjalan.

Tabel 8.20 Penelusuran `ServiceDispatch.main` dengan tangan. *Overload* dipilih dari tipe yang dideklarasikan; metode yang di-*override* dipilih dari kelas objeknya.

Baris	Tipe yang dideklarasikan	Kelas objek	`label` yang mana (*compiler*)	`describe`/`pricePerKg` yang mana (saat berjalan)	Mencetak
A, ke-1	`Service`	`Wash`	—	`Service.describe`, `Wash.pricePerKg`	`WASH @ 7000/kg, 3 kg = 21000`
A, ke-2	`Service`	`Iron`	—	`Service.describe`, `Iron.pricePerKg`	`IRON @ 5000/kg, 3 kg = 15000`
A, ke-3	`Service`	`Express`	—	`Express.describe`, `Express.pricePerKg`	`EXPRESS @ 14000/kg (same day), 3 kg = 42000`
D	`Service`	`Express`	`label(Service)`	`Express.describe`	`service: EXPRESS @ 14000/kg (same day)`
E	`Wash`	`Express`	`label(Wash)`	`Express.describe`	`washing: EXPRESS @ 14000/kg (same day)`
F	`Wash` (*cast*)	`Express`	`label(Wash)`	`Express.describe`	`washing: EXPRESS @ 14000/kg (same day)`

Sekarang jalankan dan bandingkan:

```
WASH @ 7000/kg, 3 kg = 21000
IRON @ 5000/kg, 3 kg = 15000
EXPRESS @ 14000/kg (same day), 3 kg = 42000
service: EXPRESS @ 14000/kg (same day)
washing: EXPRESS @ 14000/kg (same day)
washing: EXPRESS @ 14000/kg (same day)
PASS express price
PASS overload by declared type
PASS cast changes the overload
PASS express is a wash
All 4 checks passed.
```

Baris demi baris sepanjang penelusuran. Pada putaran ketiga gelung A, `s.describe()` menjalankan `Express.describe`, yang memanggil `super.describe()`, versi di `Service`. Di dalamnya, `pricePerKg()` dipanggil pada objek yang sama, yang tetap sebuah `Express`, sehingga yang dijalankan adalah `Express.pricePerKg`; metode itu memanggil `super.pricePerKg()`, versi milik `Wash`, mendapatkan 7000, lalu menggandakannya. *Dynamic dispatch* (pemilihan metode saat berjalan) berlaku untuk *setiap* pemanggilan pada objek itu, termasuk pemanggilan dari dalam kode superkelas. Baris D adalah pelajaran utama soal ini: `s` memegang sebuah `Express`, yang *adalah* sebuah `Wash`, namun yang dipilih `label(Service)`, karena *compiler* hanya melihat `Service s` dan harus memutuskan sebelum program berjalan. Di dalam `label`, `s.describe()` kembali dipilih saat berjalan, sehingga teksnya adalah deskripsi `Express`. Baris F memperlihatkan bahwa *cast* mengubah tipe yang dideklarasikan, dan karena itu mengubah *overload*-nya. Konstruktor `Wash(String code)` bersifat `protected`: `Express` memerlukannya untuk meneruskan kodenya sendiri ke atas, tetapi pemakai biasa sebaiknya membuat layanan cuci dengan `new Wash()`.

Versi Python. Python tidak memiliki *overloading* berdasarkan tipe yang dideklarasikan (`def label` kedua hanya akan menggantikan yang pertama), dan memang tidak ada tipe deklarasi yang dapat dijadikan pegangan. `functools.singledispatch` [8] dari pustaka baku menawarkan sesuatu yang berkaitan tetapi berbeda: ia memilih fungsi berdasarkan kelas argumen pertama *saat berjalan*. Karena itu **Kode 8.46** mencetak `washing:` untuk objek `Express`, berbeda dari baris D di Java; perbedaan itulah intinya.

Kode 8.46 `service_dispatch.py`: hierarki yang sama dalam Python, dengan `singledispatch` yang memilih berdasarkan kelas saat berjalan.

```python
# service_dispatch.py - Soal 8.19: hierarki kelas yang ditelusuri dengan tangan.
import sys
import unittest
from abc import ABC, abstractmethod
from functools import singledispatch


class Service(ABC):
    def __init__(self, code):
        self._code = code

    @abstractmethod
    def price_per_kg(self):
        """Rupiah per kilogram."""

    def price(self, kg):
        """Kode yang sama untuk setiap layanan; price_per_kg() dipilih saat program
berjalan."""
        return self.price_per_kg() * kg

    def describe(self):
        return f"{self._code} @ {self.price_per_kg()}/kg"


class Wash(Service):
    def __init__(self, code="WASH"):
        super().__init__(code)

    def price_per_kg(self):
        return 7000


class Iron(Service):
    def __init__(self):
        super().__init__("IRON")

    def price_per_kg(self):
        return 5000


class Express(Wash):
    def __init__(self):
        super().__init__("EXPRESS")

    def price_per_kg(self):
        return 2 * super().price_per_kg()   # dua kali tarif Wash
```

Kode 8.46 (Lanjutan) *service_dispatch.py: hierarki yang sama dalam Python, dengan singledispatch yang memilih berdasarkan kelas saat berjalan.*

```
    def describe(self):
        return super().describe() + " (same day)"


# Python tidak mengenal overloading menurut tipe yang dideklarasikan. singledispatch
memilih
# berdasarkan tipe argumen SAAT BERJALAN.
@singledispatch
def label(s):
    return "service: " + s.describe()


@label.register
def _(w: Wash):
    return "washing: " + w.describe()


class TestDispatch(unittest.TestCase):
    def test_prices(self):
        self.assertEqual([21000, 15000, 42000],
                         [s.price(3) for s in (Wash(), Iron(), Express())])

    def test_label_uses_runtime_type(self):
        s = Express()
        self.assertEqual("washing: EXPRESS @ 14000/kg (same day)", label(s))

    def test_label_for_plain_service(self):
        self.assertEqual("service: IRON @ 5000/kg", label(Iron()))


if __name__ == "__main__":
    for s in [Wash(), Iron(), Express()]:
        print(f"{s.describe()}, 3 kg = {s.price(3)}")
    s = Express()
    print(label(s))
    print(label(Iron()))
    sys.stdout.flush()
    unittest.main(verbosity=2)
```

```
WASH @ 7000/kg, 3 kg = 21000
IRON @ 5000/kg, 3 kg = 15000
EXPRESS @ 14000/kg (same day), 3 kg = 42000
washing: EXPRESS @ 14000/kg (same day)
service: IRON @ 5000/kg
test_label_for_plain_service (__main__.TestDispatch.test_label_for_plain_service) ... ok
test_label_uses_runtime_type (__main__.TestDispatch.test_label_uses_runtime_type) ... ok
test_prices (__main__.TestDispatch.test_prices) ... ok

----------------------------------------------------------------------
Ran 3 tests in 0.000s

OK
```

Penyelesaian keliru yang menggoda. Ada dua, keduanya tentang `super`. *Perkiraan* keliru yang menggoda adalah "baris D mencetak `washing:`, karena objeknya sebuah `Express`, yang adalah `Wash`": itu penalaran saat-berjalan yang diterapkan pada keputusan saat-kompilasi. *Kode* keliru yang menggoda adalah menulis `return 2 * pricePerKg();` di `Express`, lupa menulis `super.`. Pemanggilan itu lalu merujuk ke `Express.pricePerKg` sendiri, yang memanggil dirinya lagi, terus-menerus, sampai Java melempar `StackOverflowError` (Python: `RecursionError`). `super.m()` berarti "versi `m` di superkelas saya"; `m()` biasa berarti "versi untuk objek ini", yang di dalam sebuah *override* adalah *override* itu sendiri.

UJI DIRI. Misalkan `Iron` juga memperluas `Wash` alih-alih `Service`, dengan tetap memiliki `pricePerKg` sendiri dan kode konstruktor "IRON" melalui konstruktor *protected*. Apa yang akan dicetak `label(new Iron())`, dengan tipe argumen yang dideklarasikan adalah `Iron`?

Jawaban. Compiler memilih *overload* yang berlaku dan paling spesifik. `Iron` kini subkelas `Wash`, sehingga `label(Service)` maupun `label(Wash)` berlaku, dan `label(Wash)` lebih spesifik: ia mencetak `washing: IRON @ 5000/kg`.

8.8.2 Soal 8.20 — Me-*refactor* dua kelas yang berulang menjadi sebuah hierarki

SOAL. Sebuah permainan kecil memiliki dua kelas, `Player` dan `Enemy`. Keduanya memiliki *field private* `name`, `health`, `power`, dan `defense`, *getter* yang sama, `setHealth` yang publik, metode `attack` (milik pemain menerima `Enemy`, milik musuh menerima `Player`), serta metode `takeDamage(damage)` yang identik, yang mengurangkan `damage - defense` dari `health` bila hasilnya positif dan mencetak "`<name> died!`" ketika *health* mencapai nol atau kurang. Sebuah program uji menciptakan seorang pahlawan (*health* 100, *power* 25, *defense* 10) dan seorang penjahat (80, 20, 5) dan melakukan tiga serangan: pahlawan, penjahat, pahlawan. Jelaskan apa yang dicetak program itu, kenali masalah-masalah rancangannya, dan *refactor* kedua kelas itu menjadi sebuah hierarki yang di dalamnya pemain juga dapat menyembuhkan diri dengan ramuan dan musuh bertarung lebih keras ketika terluka parah.

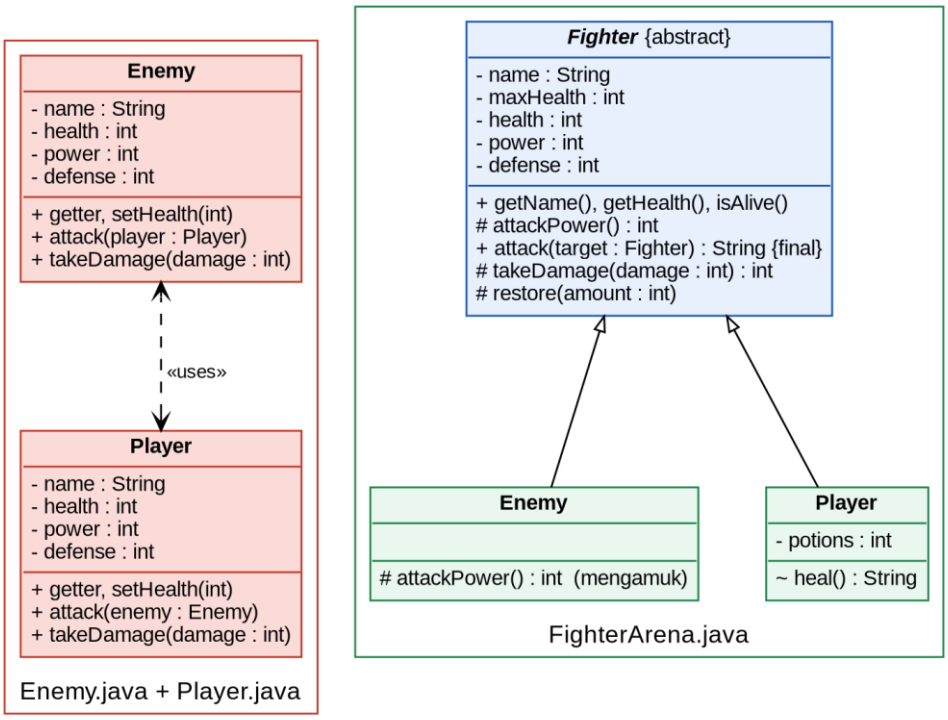
Dengan bahasa kita sendiri. Dua kelas yang saling menyalin adalah perangkap pemeliharaan: aturan yang diubah di satu kelas dan terlupa di kelas lainnya membuat permainan tidak adil dengan cara yang tidak disadari siapa pun. *Refactoring* yang disebut *extract superclass* (mengekstrak superkelas) [12] memindahkan bagian yang sama ke satu kelas abstrak. Sesudah itu, setiap subkelas hanya memuat apa yang membuatnya berbeda.

Rancangan. Pertama, penelusuran program asli dengan tangan. Pahlawan menyerang: penjahat menerima $25 - 5 = 20$, *health* 80 → 60. Penjahat menyerang: pahlawan menerima $20 - 10 = 10$, *health* 100 → 90. Pahlawan menyerang: 60 → 40. Tidak ada yang mencapai nol, dan `takeDamage` hanya mencetak ketika mati, sehingga **program asli tidak mencetak apa pun**. Menjalankannya menegaskan hal itu: program selesai tanpa sebaris keluaran pun. Uji yang tidak mencetak dan tidak memeriksa apa pun tidak memberi tahu kita apa pun; itulah masalah pertama. **Tabel 8.21** mendaftar masalah-masalah lainnya, masing-masing dengan perbaikannya.

Tabel 8.21 Masalah-masalah rancangan pada kelas `Player` dan `Enemy` yang berulang, dan bagaimana *refactoring* memperbaiki masing-masing.

Masalah pada versi asli	Bab	Perbaikan di `FighterArena`
Dua salinan identik dari *field* dan metode	7	superkelas abstrak `Fighter` menyimpannya satu kali
`attack(Enemy)` dan `attack(Player)`: dua *overload* untuk satu gagasan	6, 7	satu `attack(Fighter target)`, polimorfis
`setHealth` yang publik membuat siapa pun dapat menetapkan *health* berapa pun	7	tanpa *setter*; *health* hanya berubah melalui `takeDamage` dan `restore`
Health dapat turun di bawah nol	5	`Math.max(0, ...)` menjaganya dalam $[0, maxHealth]$
`takeDamage` mencetak ke konsol	6	metode mengembalikan nilai; `main` yang memutuskan apa yang dicetak
Tidak ada validasi untuk statistik awal	5	konstruktor melempar `IllegalArgumentException`
Program uji tanpa pemeriksaan	6	pemanggilan `check` dengan nilai hasil penelusuran tangan

Gambar 8.9 memperlihatkan kelas-kelas itu sebelum dan sesudahnya. Metode *hook* (kait) `attackPower()` yang *protected* adalah satu-satunya tempat subkelas boleh mengubah aturan bertarung; metode `final` `attack` menjamin bahwa setiap serangan mengikuti langkah-langkah yang sama.



Gambar 8.9 Sebelum (kiri): dua salinan dari *field* dan metode yang sama, masing-masing kelas memakai kelas yang lain. Sesudah (kanan): satu `Fighter` abstrak, dan subkelas-subkelas yang hanya menambahkan apa yang berbeda.

Kode 8.47 dan **Kode 8.48** mengimplementasikan rancangan **Gambar 8.9** dan memutar ulang ketiga pukulan dari program uji aslinya.

Kode 8.47 `FighterArena.java`: hierarki hasil *refactoring* dengan *template method* yang *final*, *hook* yang *protected*, dan tanpa *setter* publik.

```java
// FighterArena.java - Soal 8.20: dua kelas kembar difaktorkan ulang menjadi sebuah hierarki.
public class FighterArena {

    /** Semua yang dahulu diulang oleh Player dan Enemy, ditulis sekali. */
    abstract static class Fighter {
        private final String name;
        private final int maxHealth;
        private int health;                    // hanya diubah oleh takeDamage/heal
        private final int power;
        private final int defense;

        protected Fighter(String name, int health, int power, int defense) {
            if (health <= 0 || power < 0 || defense < 0) {
                throw new IllegalArgumentException("bad stats for " + name);
            }
            this.name = name;
            this.maxHealth = health;
            this.health = health;
            this.power = power;
            this.defense = defense;
        }

        public String getName() {
            return name;
        }

        public int getHealth() {
            return health;
        }

        public boolean isAlive() {
            return health > 0;
        }

        protected int getPower() {
            return power;
        }

        protected int getMaxHealth() {
            return maxHealth;
        }
```

Kode 8.47 (Lanjutan) *FighterArena.java: hierarki hasil refactoring dengan template method yang final, hook yang protected, dan tanpa setter publik.*

```
        /** Seberapa keras saya memukul saat ini. Subkelas boleh mengubahnya (sebuah hook).
*/
        protected int attackPower() {
            return power;
        }

        /** Sama untuk setiap petarung, jadi final: subkelas tidak dapat mengubahnya. */
        public final String attack(Fighter target) {
            int dealt = target.takeDamage(attackPower());
            return name + " hits " + target.getName() + " for " + dealt
                    + " -> " + target.getName() + " has " + target.getHealth()
                    + (target.isAlive() ? "" : " (defeated)");
        }

        /** Pertahanan menyerap sebagian pukulan; health tidak pernah turun di bawah 0. */
        protected int takeDamage(int damage) {
            int reduced = Math.max(0, damage - defense);
            health = Math.max(0, health - reduced);
            return reduced;
        }

        protected void restore(int amount) {
            health = Math.min(maxHealth, health + amount);
        }
    }

    static final class Player extends Fighter {
        private int potions;

        Player(String name, int health, int power, int defense, int potions) {
            super(name, health, power, defense);
            this.potions = potions;
        }

        /** Meminum ramuan (+30 health, tidak pernah melebihi maksimum) bila masih ada. */
        String heal() {
            if (potions == 0) {
                return getName() + " has no potions left";
            }
            potions--;
            restore(30);
            return getName() + " drinks a potion -> " + getHealth();
        }
    }
```

Kode 8.47 (Lanjutan) *FighterArena.java: hierarki hasil refactoring dengan template method yang final, hook yang protected, dan tanpa setter publik.*

```
    static final class Enemy extends Fighter {
        Enemy(String name, int health, int power, int defense) {
            super(name, health, power, defense);
        }

        /** Di bawah separuh health, musuh mengamuk dan memukul 10 lebih keras. */
        @Override
        protected int attackPower() {
            boolean enraged = getHealth() < getMaxHealth() / 2;
            return enraged ? getPower() + 10 : getPower();
        }
    }

    private static int passed = 0;

    private static void check(String name, Object expected, Object actual) {
        if (!java.util.Objects.equals(expected, actual)) {
            throw new AssertionError(name + ": expected " + expected + " but got " +
actual);
        }
        passed++;
        System.out.println("PASS " + name);
    }

    public static void main(String[] args) {
        Player hero = new Player("Hero", 100, 25, 10, 1);
        Enemy villain = new Enemy("Villain", 80, 20, 5);

        // Tiga pukulan dari EnemyPlayerTest.java:
        System.out.println(hero.attack(villain));
        System.out.println(villain.attack(hero));
        System.out.println(hero.attack(villain));
        check("villain after two blows", 40, villain.getHealth());
        check("hero after one blow", 90, hero.getHealth());

        // Lanjutkan pertarungan: kedua petarung bergiliran sampai salah satu kalah.
        Fighter[] turn = {villain, hero};      // villain bergerak berikutnya
        int i = 0;
        while (hero.isAlive() && villain.isAlive()) {
            Fighter attacker = turn[i % 2];
            Fighter defender = turn[(i + 1) % 2];
            System.out.println(attacker.attack(defender));
            i++;
        }
        System.out.println(hero.heal());
        System.out.println(hero.heal());
        check("villain defeated", false, villain.isAlive());
```

Kode 8.47 (Lanjutan) *FighterArena.java: hierarki hasil refactoring dengan template method yang final, hook yang protected, dan tanpa setter publik.*

```
        check("health never negative", 0, villain.getHealth());
        System.out.println("All " + passed + " checks passed.");
    }
}
```

```
Hero hits Villain for 20 -> Villain has 60
Villain hits Hero for 10 -> Hero has 90
Hero hits Villain for 20 -> Villain has 40
PASS villain after two blows
PASS hero after one blow
Villain hits Hero for 10 -> Hero has 80
Hero hits Villain for 20 -> Villain has 20
Villain hits Hero for 20 -> Hero has 60
Hero hits Villain for 20 -> Villain has 0 (defeated)
Hero drinks a potion -> 90
Hero has no potions left
PASS villain defeated
PASS health never negative
All 4 checks passed.
```

Kode 8.48 `fighter_arena.py`: hierarki yang sama dalam Python, dengan *property* `health` yang hanya-baca.

```python
# fighter_arena.py - Soal 8.20: dua kelas kembar difaktorkan ulang menjadi sebuah hierarki.
import sys
import unittest
from abc import ABC


class Fighter(ABC):
    """Semua yang dahulu diulang oleh Player dan Enemy, ditulis sekali."""

    def __init__(self, name, health, power, defense):
        if health <= 0 or power < 0 or defense < 0:
            raise ValueError(f"bad stats for {name}")
        self._name = name
        self._max_health = health
        self._health = health               # hanya diubah oleh take_damage/restore
        self._power = power
        self._defense = defense

    @property
    def name(self):
        return self._name

    @property
    def health(self):                       # baca-saja: tidak ada setter
        return self._health

    def is_alive(self):
        return self._health > 0

    def attack_power(self):
        """Seberapa keras saya memukul saat ini. Subkelas boleh mengubahnya (sebuah
hook)."""
        return self._power

    def attack(self, target):
        """Sama untuk setiap petarung; menurut konvensi, subkelas tidak mengubahnya."""
        dealt = target._take_damage(self.attack_power())
        return (f"{self._name} hits {target.name} for {dealt} -> {target.name} has "
                f"{target.health}" + ("" if target.is_alive() else " (defeated)"))

    def _take_damage(self, damage):
        """Pertahanan menyerap sebagian pukulan; health tidak pernah turun di bawah 0."""
        reduced = max(0, damage - self._defense)
        self._health = max(0, self._health - reduced)
        return reduced
```

Kode 8.48 (Lanjutan) *fighter_arena.py: hierarki yang sama dalam Python, dengan property health yang hanya-baca.*

```
    def _restore(self, amount):
        self._health = min(self._max_health, self._health + amount)


class Player(Fighter):
    def __init__(self, name, health, power, defense, potions):
        super().__init__(name, health, power, defense)
        self._potions = potions

    def heal(self):
        """Meminum ramuan (+30 health, tidak pernah melebihi maksimum) bila masih ada."""
        if self._potions == 0:
            return f"{self._name} has no potions left"
        self._potions -= 1
        self._restore(30)
        return f"{self._name} drinks a potion -> {self._health}"


class Enemy(Fighter):
    def attack_power(self):
        """Di bawah separuh health, musuh mengamuk dan memukul 10 lebih keras."""
        enraged = self._health < self._max_health // 2
        return self._power + 10 if enraged else self._power


class TestArena(unittest.TestCase):
    def test_three_blows(self):
        hero, villain = Player("Hero", 100, 25, 10, 1), Enemy("Villain", 80, 20, 5)
        hero.attack(villain)
        villain.attack(hero)
        hero.attack(villain)
        self.assertEqual((90, 40), (hero.health, villain.health))

    def test_health_is_read_only(self):
        with self.assertRaises(AttributeError):
            Player("Hero", 100, 25, 10, 1).health = -500

    def test_bad_stats(self):
        with self.assertRaises(ValueError):
            Enemy("Ghost", 0, 1, 1)
```

Kode 8.48 (Lanjutan) *fighter_arena.py: hierarki yang sama dalam Python, dengan property health yang hanya-baca.*

```
if __name__ == "__main__":
    hero = Player("Hero", 100, 25, 10, 1)
    villain = Enemy("Villain", 80, 20, 5)
    print(hero.attack(villain))
    print(villain.attack(hero))
    print(hero.attack(villain))
    turn = [villain, hero]                  # villain bergerak berikutnya
    i = 0
    while hero.is_alive() and villain.is_alive():
        print(turn[i % 2].attack(turn[(i + 1) % 2]))
        i += 1
    print(hero.heal())
    print(hero.heal())
    sys.stdout.flush()
    unittest.main(verbosity=2)
```

```
Hero hits Villain for 20 -> Villain has 60
Villain hits Hero for 10 -> Hero has 90
Hero hits Villain for 20 -> Villain has 40
Villain hits Hero for 10 -> Hero has 80
Hero hits Villain for 20 -> Villain has 20
Villain hits Hero for 20 -> Hero has 60
Hero hits Villain for 20 -> Villain has 0 (defeated)
Hero drinks a potion -> 90
Hero has no potions left
test_bad_stats (__main__.TestArena.test_bad_stats) ... ok
test_health_is_read_only (__main__.TestArena.test_health_is_read_only) ... ok
test_three_blows (__main__.TestArena.test_three_blows) ... ok

----------------------------------------------------------------------
Ran 3 tests in 0.000s

OK
```

Baris demi baris. Tiga baris keluaran pertama mereproduksi penelusuran tangan dengan tepat (60, 90, 40), dan dua pemeriksaan sesudahnya mengunci bilangan-bilangan itu. Kemudian pertarungan berlanjut dengan kedua petarung di dalam larik bertipe `Fighter[]`; gelung memanggil `attacker.attack(defender)` tanpa mengetahui siapa yang mana. Ikuti kekuatan penjahat: pada *health* 40 ia belum mengamuk ($40 < 80/2$ bernilai salah), sehingga pukulannya $20 - 10 = 10$; pada *health* 20 ia *sudah* mengamuk, `attackPower()` mengembalikan 30, dan pahlawan kehilangan $30 - 10 = 20$. Perilaku khusus musuh sepenuhnya berada dalam *override*-nya terhadap *hook* itu; `attack` sendiri, karena `final`, tidak dapat diubah oleh subkelas mana pun. Setelah penjahat dikalahkan, pemain meminum satu-satunya ramuan ($60 + 30 = 90$), dan `heal()` yang kedua melaporkan bahwa ramuannya sudah habis.

Kontrol aksesnya layak dibaca dengan saksama. `health` bersifat `private` di `Fighter`; subkelas tidak dapat menyentuhnya secara langsung, sehingga bahkan `Player.heal()` harus melalui metode `protected restore`, yang menegakkan batas maksimumnya. `takeDamage` bersifat `protected`: metode `attack` milik seorang petarung memanggilnya pada petarung *lain*, dan itu diperbolehkan karena pemanggilannya dilakukan dari kode di dalam `Fighter`. Satu catatan yang jujur: semua kelas ini bersarang di dalam satu berkas, dan Java mengizinkan kode di dalam kelas tingkat atas yang sama melihat bahkan anggota `private` dari kelas-kelas bersarangnya. Tingkat akses menunjukkan kekuatan penuhnya ketika kelas-kelas berada dalam berkas dan paket yang terpisah (Bab 7); rancangan di sini ditulis seolah-olah memang demikian. Di Python, `health` adalah `property` yang hanya-baca, sehingga `hero.health = -500` memunculkan `AttributeError`, seperti yang diperlihatkan uji `test_health_is_read_only`.

Penyelesaian keliru yang menggoda. "*Refactoring*" yang paling cepat adalah membuat `Enemy extends Player`, karena kodenya memang sama. Cara itu menghilangkan perulangan kode dan dapat dikompilasi, tetapi ia menyatakan bahwa setiap musuh *adalah* seorang pemain: musuh akan mewarisi ramuan dan `heal()`, dan setiap metode yang menerima `Player` akan menerima musuh. Pewarisan menyatakan hubungan "adalah jenis khusus dari" yang sungguhan; bila dua kelas bersaudara, bagian yang sama milik superkelas bersama, bukan milik salah satunya.

> **UJI DIRI.** Tambahkan `Boss` yang memperluas `Enemy` dan mengabaikan 15 poin pertama dari setiap pukulan, sebagai pengganti *defense*-nya. Metode mana yang di-*override*-nya, dan apa yang terjadi pada serangan pahlawan berkekuatan 25 terhadap *boss* dengan *defense* 5?
>
> *Jawaban.* `Boss` meng-*override* metode *protected* `takeDamage`. Karena `health` bersifat *private* di `Fighter`, *override* itu tidak dapat mengurangkannya secara langsung; penyelesaian yang bersih menambahkan metode bantu *protected* `loseHealth(int amount)` ke `Fighter`,

bayangan cermin dari `restore`, dan `Boss.takeDamage` menghitung `Math.max(0, damage - 15)`, memanggil `loseHealth`, lalu mengembalikan jumlahnya. Pukulan pahlawan menimbulkan kerusakan $25 - 15 = 10$.

8.8.3 Soal 8.21 — Perburuan *bug* enkapsulasi

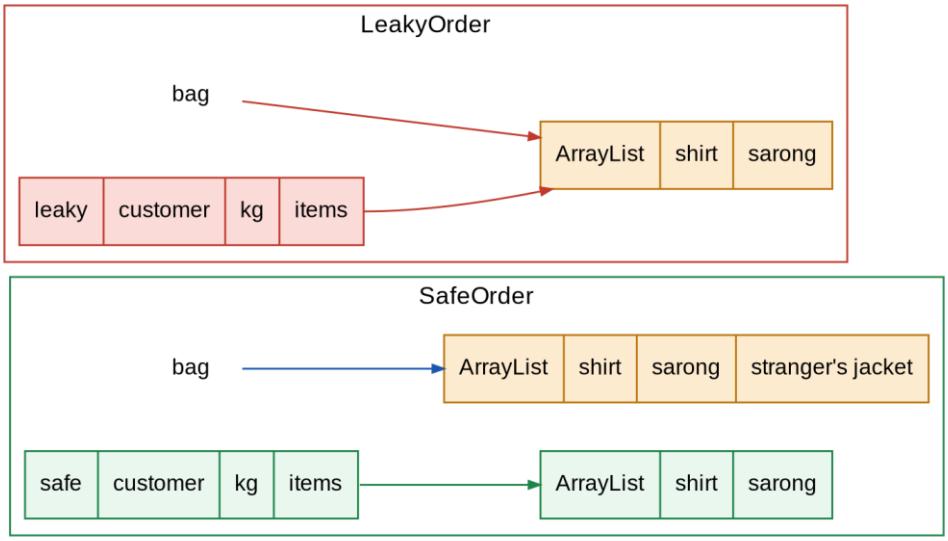
SOAL. Kelas `LeakyOrder` pada **Kode 8.49** dapat dikompilasi dan dijalankan, namun kode di luar kelas dapat menempatkan sebuah pesanan dalam keadaan yang mustahil melalui empat cara yang berbeda. Temukan keempat *bug* itu, peragakan masing-masing, dan tulislah `SafeOrder`, yang memiliki data yang sama tetapi tidak dapat dirusak dari luar.

Dengan bahasa kita sendiri. *Enkapsulasi* (Bab 7) berarti bahwa hanya objek itu sendiri yang mengendalikan keadaannya, sehingga aturan tentang keadaan itu dapat ditegakkan di satu tempat. *Bug* enkapsulasi adalah jalan apa pun yang memungkinkan kode luar mengubah keadaan tanpa melewati pemeriksaan.

Rancangan. Bacalah kelas itu pelan-pelan dan tanyakan untuk setiap *field*: "siapa yang dapat mengubah ini?" Keempat jawabannya adalah keempat *bug* itu (**Tabel 8.22**). *Bug* 2 dan 4 sama-sama *aliasing*: dua referensi ke satu objek yang dapat berubah, salah satunya dipegang oleh orang asing. **Gambar 8.10** menggambarkannya.

Tabel 8.22 Keempat *bug* enkapsulasi pada `LeakyOrder`. Setiap *bug* adalah jalan menuju keadaan objek yang melewati pemeriksaan.

Bug	Di mana	Bagaimana kode luar merusak pesanan	Perbaikan
1	`public String customer`	`leaky.customer = null`	*field* `private final`, hanya *getter*
2	konstruktor menyimpan `items`	pemanggil menyimpan `bag` dan mengubahnya kemudian	salin *list* saat *masuk*: `new ArrayList<>(items)`
3	`setKg` tanpa pemeriksaan	`setKg(-2)`	validasi, lempar `IllegalArgumentException`
4	`getItems()` mengembalikan *field*-nya	`getItems().clear()`	kembalikan tampilan hanya-baca saat *keluar*



Gambar 8.10 *Aliasing*. Atas: `LeakyOrder` dan pemanggil berbagi satu *list*, sehingga keduanya dapat mengubahnya. Bawah: `SafeOrder` menyalin *list* itu, sehingga perubahan yang kemudian dilakukan pemanggil tidak sampai ke pesanan.

Kode 8.49 `OrderBugHunt.java`: kelas dengan empat *bug* enkapsulasi yang diperagakan, dan kelas perbaikannya yang diuji.

```
// OrderBugHunt.java - Soal 8.21: temukan empat bug enkapsulasi, lalu perbaiki.
import java.util.ArrayList;
import java.util.Collections;

public class OrderBugHunt {

    /** Kelas seperti mula-mula ditulis. Ia terkompilasi, dan rusak dalam empat cara. */
    static class LeakyOrder {
        public String customer;                              // bug 1: field public
        private double kg;
        private final ArrayList<String> items;

        LeakyOrder(String customer, double kg, ArrayList<String> items) {
            this.customer = customer;
            this.kg = kg;
            this.items = items;                              // bug 2: menyimpan list milik
pemanggil
        }

        void setKg(double kg) {
            this.kg = kg;                                    // bug 3: tanpa validasi
        }

        ArrayList<String> getItems() {
            return items;                                    // bug 4: menyerahkan isi
dalamnya
        }

        double getKg() {
            return kg;
        }
    }

    /** Pesanan yang sama dengan setiap bug diperbaiki. */
    static final class SafeOrder {
        private final String customer;
        private double kg;
        private final ArrayList<String> items;

        SafeOrder(String customer, double kg, ArrayList<String> items) {
            if (customer == null || customer.isBlank()) {
                throw new IllegalArgumentException("customer is required");
            }
            this.customer = customer;
            setKg(kg);                                       // satu tempat memeriksa berat
            this.items = new ArrayList<>(items);             // salinan defensif MASUK
        }
```

Kode 8.49 (Lanjutan) *OrderBugHunt.java: kelas dengan empat bug enkapsulasi yang diperagakan, dan kelas perbaikannya yang diuji.*

```
        void setKg(double kg) {
            if (kg <= 0 || kg > 50) {
                throw new IllegalArgumentException("weight out of range: " + kg);
            }
            this.kg = kg;
        }

        java.util.List<String> getItems() {
            return Collections.unmodifiableList(items);      // tampilan baca-saja KELUAR
        }

        String getCustomer() {
            return customer;
        }

        double getKg() {
            return kg;
        }
    }

    private static int passed = 0;

    private static void check(String name, Object expected, Object actual) {
        if (!java.util.Objects.equals(expected, actual)) {
            throw new AssertionError(name + ": expected " + expected + " but got " +
actual);
        }
        passed++;
        System.out.println("PASS " + name);
    }

    public static void main(String[] args) {
        ArrayList<String> bag = new ArrayList<>();
        bag.add("shirt");
        bag.add("sarong");

        LeakyOrder leaky = new LeakyOrder("Sari", 3.5, bag);
        leaky.customer = null;                               // bug 1 beraksi
        bag.add("stranger's jacket");                        // bug 2 beraksi
        leaky.setKg(-2);                                     // bug 3 beraksi
        leaky.getItems().clear();                            // bug 4 beraksi
        System.out.println("leaky: customer=" + leaky.customer + ", kg=" + leaky.getKg()
                + ", items=" + leaky.getItems() + ", caller's bag=" + bag);
```

Kode 8.49 (Lanjutan) *OrderBugHunt.java: kelas dengan empat bug enkapsulasi yang diperagakan, dan kelas perbaikannya yang diuji.*

```
        bag = new ArrayList<>();
        bag.add("shirt");
        bag.add("sarong");
        SafeOrder safe = new SafeOrder("Sari", 3.5, bag);
        bag.add("stranger's jacket");
        try {
            safe.setKg(-2);
        } catch (IllegalArgumentException e) {
            System.out.println("setKg(-2) refused: " + e.getMessage());
        }
        try {
            safe.getItems().clear();
        } catch (UnsupportedOperationException e) {
            System.out.println("getItems().clear() refused:
UnsupportedOperationException");
        }
        System.out.println("safe : customer=" + safe.getCustomer() + ", kg=" + safe.getKg()
                + ", items=" + safe.getItems());

        check("items unaffected by caller", "[shirt, sarong]", safe.getItems().toString());
        check("weight kept", 3.5, safe.getKg());
        try {
            new SafeOrder(" ", 1, new ArrayList<>());
            throw new AssertionError("no exception");
        } catch (IllegalArgumentException e) {
            check("blank customer refused", "customer is required", e.getMessage());
        }
        System.out.println("All " + passed + " checks passed.");
    }
}
```

```
leaky: customer=null, kg=-2.0, items=[], caller's bag=[]
setKg(-2) refused: weight out of range: -2.0
getItems().clear() refused: UnsupportedOperationException
safe : customer=Sari, kg=3.5, items=[shirt, sarong]
PASS items unaffected by caller
PASS weight kept
PASS blank customer refused
All 3 checks passed.
```

Kode 8.50 `order_bug_hunt.py`: perburuan yang sama dalam Python, diperbaiki dengan *property*, *setter* yang memvalidasi, dan *tuple*.

```
# order_bug_hunt.py - Soal 8.21: temukan empat bug enkapsulasi, lalu perbaiki.
import sys
import unittest


class LeakyOrder:
    """Kelas seperti mula-mula ditulis. Ia berjalan, dan rusak dalam empat cara."""

    def __init__(self, customer, kg, items):
        self.customer = customer        # bug 1: atribut publik, tanpa pemeriksaan
        self.kg = kg                    # bug 3: siapa pun boleh memberi berat berapa pun
        self.items = items              # bug 2 dan 4: list milik pemanggil, dipakai
bersama


class SafeOrder:
    """Pesanan yang sama dengan setiap bug diperbaiki."""

    def __init__(self, customer, kg, items):
        if not customer or not customer.strip():
            raise ValueError("customer is required")
        self._customer = customer
        self.kg = kg                    # melewati setter property di bawah
        self._items = list(items)       # salinan defensif MASUK

    @property
    def customer(self):                 # baca-saja: tidak ada setter yang didefinisikan
        return self._customer

    @property
    def kg(self):
        return self._kg

    @kg.setter
    def kg(self, value):
        if value <= 0 or value > 50:
            raise ValueError(f"weight out of range: {value}")
        self._kg = value

    @property
    def items(self):
        return tuple(self._items)       # salinan imutabel KELUAR
```

Kode 8.50 (Lanjutan) *order_bug_hunt.py: perburuan yang sama dalam Python, diperbaiki dengan property, setter yang memvalidasi, dan tuple.*

```
class TestSafeOrder(unittest.TestCase):
    def test_caller_cannot_change_items(self):
        bag = ["shirt", "sarong"]
        order = SafeOrder("Sari", 3.5, bag)
        bag.append("stranger's jacket")
        self.assertEqual(("shirt", "sarong"), order.items)

    def test_bad_weight_refused(self):
        order = SafeOrder("Sari", 3.5, [])
        with self.assertRaises(ValueError):
            order.kg = -2
        self.assertEqual(3.5, order.kg)

    def test_customer_read_only_and_required(self):
        with self.assertRaises(AttributeError):
            SafeOrder("Sari", 1, []).customer = None
        with self.assertRaises(ValueError):
            SafeOrder(" ", 1, [])


if __name__ == "__main__":
    bag = ["shirt", "sarong"]
    leaky = LeakyOrder("Sari", 3.5, bag)
    leaky.customer = None
    bag.append("stranger's jacket")
    leaky.kg = -2
    leaky.items.clear()
    print(f"leaky: customer={leaky.customer}, kg={leaky.kg}, items={leaky.items}, caller's
bag={bag}")

    bag = ["shirt", "sarong"]
    safe = SafeOrder("Sari", 3.5, bag)
    bag.append("stranger's jacket")
    try:
        safe.kg = -2
    except ValueError as e:
        print("safe.kg = -2 refused:", e)
    print(f"safe : customer={safe.customer}, kg={safe.kg}, items={safe.items}")
    sys.stdout.flush()
    unittest.main(verbosity=2)
```

```
leaky: customer=None, kg=-2, items=[], caller's bag=[]
safe.kg = -2 refused: weight out of range: -2
safe : customer=Sari, kg=3.5, items=('shirt', 'sarong')
test_bad_weight_refused (__main__.TestSafeOrder.test_bad_weight_refused) ... ok
test_caller_cannot_change_items (__main__.TestSafeOrder.test_caller_cannot_change_items)
... ok
test_customer_read_only_and_required
(__main__.TestSafeOrder.test_customer_read_only_and_required) ... ok

----------------------------------------------------------------------
Ran 3 tests in 0.000s

OK
```

Baris demi baris. Baris keluaran pertama memperlihatkan keempat *bug* sekaligus: tanpa pelanggan, berat negatif, dan daftar barang kosong, dengan `bag` milik pemanggil ikut kosong, karena `leaky.getItems().clear()` mengosongkan *list* yang persis sama dengan yang dirujuk `bag`. Di `SafeOrder`, konstruktor memeriksa pelanggan, mengirim berat melalui `setKg` sehingga aturan “antara 0 dan 50 kg” hanya tinggal di satu metode, dan menyalin *list* yang masuk; `bag.add("stranger's jacket")` yang datang kemudian hanya mengubah *list* milik pemanggil.

`getItems()` mengembalikan `Collections.unmodifiableList(items)`, sebuah *tampilan* (*view*) hanya-baca: membaca berjalan, dan setiap upaya mengubahnya melempar `UnsupportedOperationException`. Tipe kembaliannya ditulis lengkap `java.util.List<String>` karena folder bab ini juga memuat kelas kita sendiri yang bernama `List` (**Kode 8.3**); nama sederhana `List` di berkas ini akan berarti kelas kita. Di Python (**Kode 8.50**), `kg` adalah *property* dengan *setter* yang memvalidasi, sehingga bahkan penugasan `self.kg = kg` di dalam `__init__` ikut diperiksa; `customer` tidak memiliki *setter*, sehingga ia hanya-baca; dan `items` mengembalikan *tuple*, yang sama sekali tidak dapat diubah.

Penyelesaian keliru yang menggoda. Menjadikan *field list* itu `final` terasa seperti perbaikan untuk *bug* 2 dan 4, padahal bukan. `final` berarti *referensinya* tidak pernah dapat ditugaskan ulang; *list* yang dirujuknya tetap dapat dikosongkan, ditambahi, dan dipakai bersama. `LeakyOrder.items` sudah `final`, dan keluarannya memperlihatkan *list* itu dikosongkan dari luar. Imutabilitas sebuah objek menuntut penyalinan dan tampilan hanya-baca, bukan sekadar *field* `final` [2].

> **UJI DIRI.** Sebuah kelas menyimpan `private final int[] slots` yang diterima konstruktornya dan memiliki `int[] getSlots() { return slots; }`. Sebutkan kedua kebocorannya dan perbaikannya.
>
> *Jawaban.* Konstruktor menjadikan larik milik pemanggil sebagai *alias* (perbaikan: `this.slots = slots.clone()` atau `Arrays.copyOf(slots, slots.length)`), dan *getter* menyerahkan larik internal (perbaikan: kembalikan salinannya dengan cara yang sama). Larik tidak memiliki tampilan hanya-baca, sehingga penyalinan adalah satu-satunya pilihan.

8.9 Perbandingan Soal-Soal dalam Bab Ini

Tabel 8.23 menjajarkan kedua puluh satu soal. Bacalah kolom terakhir dari atas ke bawah: setiap penyelesaian keliru gagal pada satu jenis masukan tertentu, dan pada kebanyakan soal masukan itu bukan contoh yang diberikan dalam pernyataan soal. Itulah argumen terkuat yang dapat diajukan bab ini untuk Langkah 7 dari metode kita.

Tabel 8.23 Kedua puluh satu soal dalam bab ini: apa yang dilatih masing-masing dan satu kasus yang membuat penyelesaian keliru yang menggoda gagal.

Soal	Struktur atau topik	Gagasan utama	Kasus tepi yang terlewat oleh penyelesaian keliru
8.1	pohon	membangun dari bawah ke atas	anak kiri dan kanan tertukar
8.2	pohon	mengubah di tempat melalui delegasi	subpohon yang tidak ada (`null`)
8.3	pohon	penelusuran *pre-order*; menguji keluaran cetak	urutan simpul dan subpohon
8.4	pohon	salinan dalam	subpohon yang dipakai bersama setelah `mirror()`
8.5	pohon, larik	mengembalikan posisi terakhir yang dipakai	awal subpohon kanan (selisih satu)
8.6	BST	satu lintasan, dibatasi tinggi	nilai tidak ada (subpohon `null`)
8.7	BST	memasang daun baru	daun diciptakan tetapi tidak dipasang
8.8	*list*	*map*	lupa melakukan rekursi
8.9	*list*	meneruskan sinyal	`-1` tergeser menjadi posisi yang sah
8.10	*list*	AND logis, kebenaran hampa	elemen negatif di akhir
8.11	*list*	*filter* yang menjaga urutan	elemen negatif di tengah
8.12	*list*	membandingkan tetangga	tetangga yang sama (`<` vs `<=`)
8.13	dua *list*	*merge* dengan dua kasus dasar	satu *list* kosong
8.14	*list*	*filter* berdasarkan kesamaan	`x` di posisi terakhir
8.15	*list*	memakai ulang metode bantu; $\Theta(n^2)$	masukan tak terurut; *list* kosong
8.16	berkas, eksepsi	`try` per baris, berkas selalu ditutup	baris buruk di tengah; berkas tidak ada
8.17	larik, Javadoc	invarian ukuran vs kapasitas	`index == size`
8.18	antarmuka	*default method*, uji kontrak	pembulatan ke atas; pemotongan *float*
8.19	hierarki	*overload* saat kompilasi, *override* saat berjalan	tipe yang dideklarasikan vs kelas objek
8.20	hierarki	*extract superclass*, *hook*, `final`	kelas bersaudara bukan *is-a*
8.21	enkapsulasi	salin saat masuk, hanya-baca saat keluar, validasi	*aliasing* melalui *field* `final`

8.10 Profitina dalam Praktik

Profitina, platform visibilitas AI (GEO) yang beroperasi dalam produksi di app.profitina.com, mengukur seberapa siap situs web sebuah merek untuk dikutip oleh mesin jawaban AI, lalu membantu memperbaiki apa yang ditemukannya. Tiga gagasan dari bab ini tampak dalam bentuk publik produk itu.

Satu antarmuka, banyak implementasi. Pemindai memeriksa sebuah situs melalui sekumpulan *sinyal*, seperti seberapa mudah isinya dapat diekstraksi, apakah situs itu menerbitkan data terstruktur, dan tanda-tanda bahwa situs itu secara teknis tepercaya. Setiap sinyal diberi skor dan dijelaskan secara mandiri, dan kode pemindai memperlakukan semuanya melalui satu antarmuka bersama, persis seperti gelung pada **Kode 8.43** memperlakukan setiap `PriceRule` dengan cara yang sama. Menambahkan sinyal berarti menambahkan kelas, bukan menyunting gelungnya; menguji sebuah sinyal berarti menjalankan kontrak uji yang sama terhadapnya, seperti yang dilakukan `PriceRuleContract` pada **Kode 8.44**.

Masukan yang tak tepercaya, diurai tanpa *crash*. Pemindai membaca halaman yang ditulis orang-orang yang tidak dikenal, dan fitur komunitasnya menyaring setiap tautan yang diunggah dengan sebuah lantai aturan (*rule floor*) yang menggabungkan beberapa sinyal. Keadaannya adalah Soal 8.16 dalam skala yang lebih besar: sebagian besar masukan baik, sebagian rusak, dan satu butir yang buruk harus mengorbankan satu butir saja, tidak pernah seluruh pekerjaan. Disiplinnya sama: tangkap kegagalan yang spesifik di tempat ia dapat ditangani, catat dengan konteks yang cukup untuk ditindaklanjuti, dan selalu lepaskan sumber daya yang sudah dibuka.

Enkapsulasi sebagai kewajiban hukum, dan uji sebagai gerbang rilis. Fitur akun Profitina dibangun untuk undang-undang pelindungan data pribadi Indonesia, UU PDP 27/2022, dengan persetujuan eksplisit, ekspor data, dan penghapusan akun secara penuh. Aturan seperti ini hanya dapat ditegakkan bila data pribadi diubah di satu tempat yang terkendali, dan itulah yang dilakukan `SafeOrder` pada Soal 8.21 untuk sebuah pesanan binatu. Selain itu, tidak ada rilis yang keluar sebelum seluruh rangkaian uji otomatis, jauh lebih dari seribu uji, lolos; program-program yang memeriksa dirinya sendiri dalam bab ini adalah gagasan yang sama dalam ukuran mini.

Seperti biasa, produk memberi saran dan penggunanya yang memutuskan: skor visibilitas adalah pengukuran beserta penjelasannya, bukan jaminan. Versi yang dapat dijalankan dari setiap gagasan di atas adalah kode Profitina Laundry dalam bab ini.

8.11 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya

Melupakan subpohon yang tidak ada. Setiap metode pohon yang rekursif harus menangani anak yang `null`: jagalah setiap delegasi dengan `if (left != null)` (Python `is not None`), atau jadikan pohon yang tidak ada itu sendiri sebagai kasus dasar. Ujilah dengan rantai dan dengan simpul yang hanya memiliki satu anak (Soal 8.2, 8.6).

Memanggil metode lalu mengabaikan hasilnya. Pemanggilan rekursif yang nilainya dibuang (`left.createFreshCopy();`) bekerja sia-sia. Bila sebuah metode mengembalikan sesuatu, pemanggil hampir selalu memerlukannya (Soal 8.4, 8.7).

Kasus dasar kedua yang melewatkan pemeriksaan. "*List* satu elemen sudah pasti beres" hanya benar bila Anda melihat elemennya. Pakailah *list* kosong sebagai satu-satunya kasus dasar, kecuali langkah gabungnya memerlukan dua elemen (Soal 8.10, 8.14; pengecualiannya 8.12).

Memperlakukan nilai sinyal sebagai data. `-1`, `null`, dan *string* kosong berarti "tidak ada jawaban"; aritmetika terhadapnya menghasilkan omong kosong yang tampak masuk akal. Periksa sinyalnya sebelum memakai nilainya (Soal 8.9).

Selisih satu di sebuah batas. `<` lawan `<=`, `>` lawan `>=`, "terakhir dipakai" lawan "bebas berikutnya". Tuliskan konvensinya dalam komentar dokumentasi dan letakkan uji tepat pada batasnya (Soal 8.5, 8.12, 8.17).

Satu `try` untuk segalanya. `try` yang mengelilingi seluruh gelung mengubah satu rekaman buruk menjadi satu berkas yang hilang, dan `catch (Exception e) {}` menyembunyikan setiap *bug*. Letakkan `try` di sekeliling satuan terkecil yang dapat gagal secara mandiri dan tangkap eksepsi yang paling spesifik (Soal 8.16).

Menghitung uang dengan bilangan titik mengambang. Ubah ke satuan bulat lebih dulu dan pakai tipe bilangan bulat; *cast* dari `double` memotong (Soal 8.18).

Mencampuradukkan *overloading* dan *overriding*. *Overload* dipilih oleh *compiler* dari tipe yang dideklarasikan; *override* dipilih saat berjalan dari kelas objeknya. Telusurilah keduanya dalam dua kolom terpisah, seperti pada **Tabel 8.20**.

Melupakan `super` dalam sebuah *override*. `return 2 * pricePerKg();` di dalam `pricePerKg` memanggil dirinya sendiri tanpa akhir. Tulislah `super.pricePerKg()` (Soal 8.19).

Pewarisan demi kenyamanan. `extends` menyatakan "adalah". `NegateAll extends List` dan `Enemy extends Player` adalah pernyataan keliru yang dapat dikompilasi. Pakailah kelas utilitas atau superkelas bersama sebagai gantinya (Subbab 8.3.2 dan 8.8.2).

Mengira `final` membuat objek imutabel. `final` mengunci referensinya, bukan objeknya. Salinlah masukan yang dapat berubah saat masuk, serahkan tampilan hanya-baca atau salinan saat keluar, dan validasilah setiap perubahan (Soal 8.21).

Hanya menguji contoh dari pernyataan soal. Pada Soal 8.1, 8.10, 8.14, 8.15, dan 8.17, uji yang hanya dibangun dari contoh biasa akan meloloskan penyelesaian keliru yang menggoda. Tulislah uji kasus tepi lebih dulu.

8.12 Ringkasan Bab

- Soal perancangan dengan banyak kelas diselesaikan dalam delapan langkah (Subbab 8.2): kumpulkan kata benda dan kata kerja, pilih kelas, beri setiap metode seorang pemilik, pilih hubungannya, tetapkan tingkat akses, gambarkan rancangannya, tulis uji beserta kasus tepinya, dan kerjakan satu metode sekali jalan.
- Metode rekursif pada pohon atau *list* ditulis dengan resep delegasi: kasus dasar, delegasi kepada bagian-bagian yang lebih kecil, dan tugas simpul itu sendiri. Untuk *list*, Persamaan (8.4) adalah cetakan setiap penyelesaian.
- Pohon: bangun dari bawah ke atas (8.1); ubah di tempat melalui delegasi (8.2); posisi tugas simpul itu sendiri menentukan *pre-*, *in-*, atau *post-order* (8.3); salinan dalam menyimpan hasil pemanggilan rekursifnya (8.4); pembukuan larik memerlukan satu konvensi yang tertulis (8.5); pencarian dan penyisipan BST menyusuri satu lintasan yang panjangnya paling banyak sama dengan tinggi pohon (8.6, 8.7).
- *List*: *map* (8.8), pencarian dengan sinyal (8.9), AND logis dengan kebenaran hampa (8.10), *filter* (8.11, 8.14), perbandingan tetangga (8.12), *merge* dengan dua kasus dasar (8.13), dan metode bantu yang dipakai ulang dengan biaya kuadratik, Persamaan (8.6) (8.15).
- Eksepsi dan berkas: eksepsi *checked* untuk setiap rekaman buruk, ditangkap di dalam gelung; berkas ditutup oleh *try-with-resources* atau `with`; galat tingkat rendah diterjemahkan menjadi pesan yang dapat ditindaklanjuti manusia (8.16).
- Larik dan dokumentasi: ukuran dan kapasitas, invarian kelas, Javadoc dan *docstring* yang menyatakan prasyarat dan eksepsi, serta uji tepat pada `index == size` (8.17).
- Antarmuka dan uji: *default method* yang ditulis sekali; implementasi yang berbagi validasi; uji kontrak yang dijalankan terhadap setiap implementasi; JUnit 6 sebagai bentuk baku dari pemeriksaan yang sama (8.18).
- Pewarisan, polimorfisme, dan kontrol akses: *overload* diselesaikan saat kompilasi, *override* saat berjalan (8.19); kelas-kelas yang berulang di-*refactor* menjadi superkelas abstrak dengan *template method* yang *final* dan *hook* yang *protected* (8.20); enkapsulasi dipertahankan dengan *field private*, validasi, salinan defensif, dan tampilan hanya-baca (8.21).

8.13 Soal Latihan

Latihan 8.1 sampai 8.6 dan 8.11 meminta kode; penyelesaiannya berupa program di Subbab 8.14. Latihan lainnya meminta penalaran dan dijawab dengan kata-kata.

1. **(Tinggi pohon.)** Tulislah `int height()` untuk `Tree`, dengan daun bertinggi 1. Berapa tinggi pohon pada **Gambar 8.1**?
2. **(Jumlah pohon.)** Tulislah `int sum()` yang menjumlahkan semua simpul sebuah pohon. Apa yang dikembalikannya untuk pohon pada **Gambar 8.1**?
3. **(Daun.)** Tulislah `int leaves()` yang menghitung banyaknya daun. Apa yang dikembalikannya untuk pohon pada **Gambar 8.1**?
4. **(Minimum BST.)** Tulislah `int min()` untuk pohon pencarian biner, dengan mengunjungi sesedikit mungkin simpul. Simpul mana saja yang dikunjunginya pada pohon **Gambar 8.6**?
5. **(Jumlah *list*.)** Tulislah `static int sum(List a)` dengan hanya memakai operasi-operasi `List`. Berapa jumlah `[2, -5, 8, 0, 8]`, dan seharusnya berapa jumlah *list* kosong?
6. **(Take.)** Tulislah `static List take(int n, List a)`, yaitu `n` elemen pertama `a`, atau semuanya bila `a` lebih pendek. `n` yang negatif adalah galat. Eksepsi mana yang sebaiknya dilemparkannya, *checked* atau *unchecked*, dan mengapa?
7. **(Eksepsi dengan tangan.)** Perkirakan, baris demi baris, keluaran **Kode 8.51** sebelum menjalankannya. Kemudian lakukan hal yang sama untuk **Kode 8.52**, dan jelaskan satu tempat ketika program Python memerlukan kode tambahan agar berperilaku seperti program Java.
8. **(Dispatch dengan tangan.)** Pada **Kode 8.45**, misalkan `describe()` di `Service` dinyatakan `final` dan *override* di `Express` dihapus. Apa yang akan dicetak baris A (putaran ketiga), D, dan E?
9. **(Enkapsulasi.)** Sebuah kelas `Receipt` memiliki `private final ArrayList<Integer> amounts`, konstruktor yang menyimpan *list* yang diberikan kepadanya, metode `total()` yang menjumlahkan nilai-nilainya, dan `public ArrayList<Integer> getAmounts()`. Uraikan rangkaian tiga pernyataan, ditulis di luar kelas, yang membuat `total()` mengembalikan nilai yang berbeda tanpa satu pun metode `Receipt` dipanggil di antaranya, lalu perbaiki kelas itu.
10. **(Rancangan.)** Profitina Laundry menambahkan kurir. Seorang kurir memiliki nama dan kendaraan (sepeda motor atau mobil); sepeda motor membawa paling banyak 10 kg per perjalanan dan mobil 40 kg. Sebuah pesanan ditugaskan kepada satu kurir, dan menugaskan pesanan yang terlalu berat untuk kendaraan kurir itu adalah galat. Terapkan Langkah 1 sampai 6 dari metode kita: daftarkan kelas-kelasnya, *field* dan metodenya beserta tingkat aksesnya, hubungan-hubungannya, dan eksepsinya.
11. **(Penyisipan persisten.)** Tulislah `Tree insertPersistent(int n)` untuk BST yang tidak mengubah simpul lama mana pun: ia mengembalikan pohon baru yang berbagi setiap subpohon yang tidak tersentuh dengan pohon lama. Berapa simpul baru yang diciptakan `insertPersistent(9)` pada pohon **Gambar 8.6**?
12. **(*Checked* atau *unchecked*.)** Pada Soal 8.16, `OrderFormatException` bersifat *checked* dan `ListException` pada Subbab 8.3.2 bersifat *unchecked*. Berikan aturan yang menentukan pilihan di antara keduanya, dan sebutkan jenis mana yang akan Anda pilih untuk eksepsi yang dilemparkan ketika sebuah aturan harga menerima berat negatif, beserta alasannya.

Kedua program untuk Latihan 7 adalah:

Kode 8.51 `TraceFinally.java`: `try` dengan `catch` dan `finally`, serta sebuah eksepsi yang tidak ditangani oleh `catch`.

```java
// TraceFinally.java - Soal Latihan 8.7: tebak keluarannya sebelum Anda menjalankannya.
public class TraceFinally {

    static int parseKg(String text) {
        try {
            System.out.println("try " + text);
            int kg = Integer.parseInt(text);
            if (kg <= 0) {
                throw new IllegalArgumentException("not positive");
            }
            return kg;
        } catch (NumberFormatException e) {
            System.out.println("catch NumberFormatException");
            return -1;
        } finally {
            System.out.println("finally " + text);
        }
    }

    public static void main(String[] args) {
        System.out.println("result " + parseKg("4"));
        System.out.println("result " + parseKg("x"));
        try {
            System.out.println("result " + parseKg("0"));
        } catch (IllegalArgumentException e) {
            System.out.println("main caught " + e.getMessage());
        }
    }
}
```

Kode 8.52 `trace_finally.py`: alur yang sama dalam Python, dengan kedua galat berupa `ValueError`.

```python
# trace_finally.py - Soal Latihan 8.7: tebak keluarannya sebelum Anda menjalankannya.


def parse_kg(text):
    try:
        print("try", text)
        kg = int(text)
        if kg <= 0:
            raise ValueError("not positive")
        return kg
    except ValueError as e:
        if str(e) == "not positive":
            raise                       # munculkan ulang: bukan urusan kita
        print("catch ValueError")
        return -1
    finally:
        print("finally", text)


print("result", parse_kg("4"))
print("result", parse_kg("x"))
try:
    print("result", parse_kg("0"))
except ValueError as e:
    print("main caught", e)
```

8.14 Penyelesaian Soal Latihan

Latihan 1 sampai 6 dan 11. **Kode 8.53** dan **Kode 8.54** memuat semua metode yang diminta, masing-masing diperiksa terhadap nilai-nilai yang dihitung di bawah ini.

Kode 8.53 `ExerciseSolutions.java`: tinggi, jumlah, daun, minimum BST, jumlah *list*, *take*, dan penyisipan persisten, beserta pemeriksaannya. Kompilasikan bersama `List.java` dan `ListException.java`.

```
// ExerciseSolutions.java - penyelesaian soal latihan pemrograman Bab 8.
public class ExerciseSolutions {

    static final class Tree {
        private final int root;
        private final Tree left, right;

        Tree(int root, Tree left, Tree right) {
            this.root = root;
            this.left = left;
            this.right = right;
        }

        Tree(int root) {
            this(root, null, null);
        }

        /** Soal Latihan 8.1: banyaknya tingkat; daun bertinggi 1. */
        int height() {
            int l = (left == null) ? 0 : left.height();
            int r = (right == null) ? 0 : right.height();
            return 1 + Math.max(l, r);
        }

        /** Soal Latihan 8.2: jumlah semua simpul. */
        int sum() {
            return root + (left == null ? 0 : left.sum()) + (right == null ? 0 :
right.sum());
        }

        /** Soal Latihan 8.3: banyaknya daun. */
        int leaves() {
            if (left == null && right == null) {
                return 1;
            }
            return (left == null ? 0 : left.leaves()) + (right == null ? 0 :
right.leaves());
        }

        /** Soal Latihan 8.4: nilai terkecil BST ada di ujung tulang punggung kiri. */
        int min() {
            return (left == null) ? root : left.min();
        }
```

Kode 8.53 (Lanjutan) *ExerciseSolutions.java: tinggi, jumlah, daun, minimum BST, jumlah list, take, dan penyisipan persisten, beserta pemeriksaannya. Kompilasikan bersama List.java dan ListException.java.*

```
        /** Soal Latihan 8.11: menyisipkan tanpa mengubah simpul yang sudah ada (penyalinan
lintasan). */
        Tree insertPersistent(int n) {
            if (n < root) {
                return new Tree(root, left == null ? new Tree(n) :
left.insertPersistent(n), right);
            } else if (n > root) {
                return new Tree(root, left, right == null ? new Tree(n) :
right.insertPersistent(n));
            }
            return this;
        }

        @Override
        public String toString() {
            if (left == null && right == null) {
                return Integer.toString(root);
            }
            String l = (left == null) ? "-" : left.toString();
            String r = (right == null) ? "-" : right.toString();
            return root + "(" + l + "," + r + ")";
        }
    }

    /** Soal Latihan 8.5: jumlah elemen sebuah List. */
    static int sum(List a) {
        return a.empty() ? 0 : a.head() + sum(a.tail());
    }

    /** Soal Latihan 8.6: n elemen pertama (semuanya bila list lebih pendek). */
    static List take(int n, List a) {
        if (n < 0) {
            throw new IllegalArgumentException("n must not be negative: " + n);
        }
        if (n == 0 || a.empty()) {
            return List.nil();
        }
        return List.cons(a.head(), take(n - 1, a.tail()));
    }

    private static int passed = 0;
```

Kode 8.53 (Lanjutan) *ExerciseSolutions.java: tinggi, jumlah, daun, minimum BST, jumlah list, take, dan penyisipan persisten, beserta pemeriksaannya. Kompilasikan bersama List.java dan ListException.java.*

```
    private static void check(String name, Object expected, Object actual) {
        if (!java.util.Objects.equals(expected, actual)) {
            throw new AssertionError(name + ": expected " + expected + " but got " +
actual);
        }
        passed++;
        System.out.println("PASS " + name);
    }

    public static void main(String[] args) {
        Tree t = new Tree(1,
                new Tree(2, new Tree(4, null, new Tree(8)), new Tree(5)),
                new Tree(3, new Tree(6), new Tree(7, new Tree(9), null)));
        check("8.1 height", 4, t.height());
        check("8.2 sum", 45, t.sum());
        check("8.3 leaves", 4, t.leaves());
        Tree bst = new Tree(5, new Tree(3, new Tree(1), new Tree(4)), new Tree(8, new
Tree(7), new Tree(10)));
        check("8.4 min", 1, bst.min());
        check("8.5 sum", 13, sum(List.of(2, -5, 8, 0, 8)));
        check("8.6 take 2", "[7, 5]", take(2, List.of(7, 5, 3)).toString());
        check("8.6 take 9", "[7, 5, 3]", take(9, List.of(7, 5, 3)).toString());
        Tree bigger = bst.insertPersistent(9);
        check("8.11 new tree", "5(3(1,4),8(7,10(9,-)))", bigger.toString());
        check("8.11 old tree kept", "5(3(1,4),8(7,10))", bst.toString());
        check("8.11 left half shared", true, bigger.left == bst.left);
        System.out.println("All " + passed + " checks passed.");
    }
}
```

```
PASS 8.1 height
PASS 8.2 sum
PASS 8.3 leaves
PASS 8.4 min
PASS 8.5 sum
PASS 8.6 take 2
PASS 8.6 take 9
PASS 8.11 new tree
PASS 8.11 old tree kept
PASS 8.11 left half shared
All 10 checks passed.
```

Kode 8.54 `exercise_solutions.py`: penyelesaian yang sama dalam Python.

```
# exercise_solutions.py - penyelesaian soal latihan pemrograman Bab 8.
import unittest
from cons_list import List


class Tree:
    def __init__(self, root, left=None, right=None):
        self._root = root
        self._left = left
        self._right = right

    def height(self):
        """Soal Latihan 8.1: banyaknya tingkat; daun bertinggi 1."""
        l = 0 if self._left is None else self._left.height()
        r = 0 if self._right is None else self._right.height()
        return 1 + max(l, r)

    def sum(self):
        """Soal Latihan 8.2: jumlah semua simpul."""
        return (self._root
                + (0 if self._left is None else self._left.sum())
                + (0 if self._right is None else self._right.sum()))

    def leaves(self):
        """Soal Latihan 8.3: banyaknya daun."""
        if self._left is None and self._right is None:
            return 1
        return ((0 if self._left is None else self._left.leaves())
                + (0 if self._right is None else self._right.leaves()))

    def min(self):
        """Soal Latihan 8.4: nilai terkecil BST mengakhiri tulang punggung kiri."""
        return self._root if self._left is None else self._left.min()

    def insert_persistent(self, n):
        """Soal Latihan 8.11: menyisipkan tanpa mengubah simpul yang sudah ada."""
        if n < self._root:
            new_left = Tree(n) if self._left is None else self._left.insert_persistent(n)
            return Tree(self._root, new_left, self._right)
        if n > self._root:
            new_right = Tree(n) if self._right is None else
self._right.insert_persistent(n)
            return Tree(self._root, self._left, new_right)
        return self
```

Kode 8.54 (Lanjutan) *exercise_solutions.py: penyelesaian yang sama dalam Python.*

```
    def __str__(self):
        if self._left is None and self._right is None:
            return str(self._root)
        l = "-" if self._left is None else str(self._left)
        r = "-" if self._right is None else str(self._right)
        return f"{self._root}({l},{r})"


def list_sum(a):
    """Soal Latihan 8.5: jumlah elemen sebuah List."""
    return 0 if a.empty() else a.head() + list_sum(a.tail())


def take(n, a):
    """Soal Latihan 8.6: n elemen pertama (semuanya bila list lebih pendek)."""
    if n < 0:
        raise ValueError(f"n must not be negative: {n}")
    if n == 0 or a.empty():
        return List.nil()
    return List.cons(a.head(), take(n - 1, a.tail()))


class TestExercises(unittest.TestCase):
    def setUp(self):
        self.t = Tree(1, Tree(2, Tree(4, None, Tree(8)), Tree(5)),
                      Tree(3, Tree(6), Tree(7, Tree(9))))
        self.bst = Tree(5, Tree(3, Tree(1), Tree(4)), Tree(8, Tree(7), Tree(10)))

    def test_tree_measures(self):
        self.assertEqual((4, 45, 4), (self.t.height(), self.t.sum(), self.t.leaves()))
        self.assertEqual(1, self.bst.min())

    def test_list_functions(self):
        self.assertEqual(13, list_sum(List.of(2, -5, 8, 0, 8)))
        self.assertEqual("[7, 5]", str(take(2, List.of(7, 5, 3))))
        self.assertEqual("[7, 5, 3]", str(take(9, List.of(7, 5, 3))))

    def test_persistent_insert(self):
        bigger = self.bst.insert_persistent(9)
        self.assertEqual("5(3(1,4),8(7,10(9,-)))", str(bigger))
        self.assertEqual("5(3(1,4),8(7,10))", str(self.bst))
        self.assertIs(self.bst._left, bigger._left)
```

Kode 8.54 (Lanjutan) *exercise_solutions.py: penyelesaian yang sama dalam Python.*

```
if __name__ == "__main__":
    unittest.main(verbosity=2)
```

```
test_list_functions (__main__.TestExercises.test_list_functions) ... ok
test_persistent_insert (__main__.TestExercises.test_persistent_insert) ... ok
test_tree_measures (__main__.TestExercises.test_tree_measures) ... ok

----------------------------------------------------------------------
Ran 3 tests in 0.000s

OK
```

1. `height()` adalah 1 ditambah tinggi yang lebih besar di antara kedua subpohon, dengan 0 untuk subpohon yang tidak ada. Lintasan terpanjang pada **Gambar 8.1** adalah 1, 2, 4, 8 (atau 1, 3, 7, 9), sehingga tingginya 4.
2. Simpul-simpulnya 1 sampai 9, sehingga jumlahnya $\frac{9 \cdot 10}{2} = 45$.

3. Daunnya 5, 6, 8, dan 9: empat. Sebuah simpul adalah daun tepat ketika kedua subpohonnya tidak ada.
4. Nilai terkecil BST berada di ujung lintasan yang selalu ke kiri: 5, 3, 1. Tiga kunjungan; subpohon-subpohon kanan tidak pernah disentuh.
5. $2 - 5 + 8 + 0 + 8 = 13$. Jumlah *list* kosong adalah 0, unsur netral penjumlahan, sama seperti *list* kosong bernilai `true` untuk AND logis pada Soal 8.10.
6. Lihat `take` pada kodenya. `n` yang negatif adalah galat pemrograman oleh pemanggil, bukan keadaan yang harus diantisipasi program yang benar, sehingga `IllegalArgumentException` yang *unchecked* (Python `ValueError`) adalah pilihan yang tepat.
7. Keluaran sebenarnya dari **Kode 8.51** ada di bawah; **Kode 8.52** mencetak baris-baris yang sama kecuali `catch ValueError` pada baris kelima.

```
try 4
finally 4
result 4
try x
catch NumberFormatException
finally x
result -1
try 0
finally 0
main caught not positive
```

Untuk `"4"`: `try`, lalu `return 4` *disiapkan*, lalu `finally` dijalankan sebelum metode benar-benar kembali, lalu `main` mencetak `result 4`. Untuk `"x"`: `parseInt` melempar eksepsi, `catch` mencetak dan menyiapkan `return -1`, `finally` dijalankan, `main` mencetak `result -1`. Untuk `"0"`: `IllegalArgumentException` bukan `NumberFormatException`, sehingga `catch` tidak cocok; `finally` tetap dijalankan, dan eksepsinya merambat naik ke `main`, tempat ia ditangkap. Di Python kedua galat berupa `ValueError`, sehingga klausa `except` menangkap keduanya; baris tambahan `if str(e) == "not positive": raise` memunculkan ulang galat yang kedua agar sampai ke pemanggil, seperti di Java. Rancangan Python yang lebih bersih akan mendefinisikan kelas eksepsinya sendiri untuk "not positive".

8. Metode `final` tidak dapat di-*override*, sehingga `describe()` selalu versi milik `Service`, yang mencetak `code @ pricePerKg/kg`. `pricePerKg()` tetap di-*override*. Baris A (putaran ketiga): `EXPRESS @ 14000/kg, 3 kg = 42000`. Baris D: `service: EXPRESS @ 14000/kg`. Baris E: `washing: EXPRESS @ 14000/kg`. *Overload*-nya dipilih persis seperti sebelumnya; hanya "(same day)" yang hilang.
9. `ArrayList<Integer> list = new ArrayList<>(); Receipt r = new Receipt(list); list.add(1000000);` sekarang `r.total()` memuat sejuta rupiah yang tidak pernah dibayar siapa pun. (Demikian pula `r.getAmounts().add(...)`.) Perbaikan: salin di konstruktor (`this.amounts = new ArrayList<>(amounts)`) dan kembalikan `Collections.unmodifiableList(amounts)`, yang dideklarasikan sebagai `java.util.List<Integer>`.
10. Kelas: `Courier` (`- name : String`, `- vehicle : Vehicle`, `+ assign(order : Order)`, `+ getAssigned()`), `Vehicle` sebagai antarmuka atau kelas abstrak dengan `+ maxKg() : double`, diimplementasikan oleh `Motorcycle` (10) dan `Car` (40), serta `Order` yang sudah ada. Hubungan: kurir *has-a* kendaraan; kurir *memiliki* daftar pesanan yang ditugaskan (diserahkan keluar dalam bentuk hanya-baca, seperti pada Soal 8.21); sepeda motor dan mobil *can-do* `Vehicle`. `assign` memeriksa `order.getKg() <= vehicle.maxKg()` dan bila tidak terpenuhi melempar `OverweightException`; *checked* bila penugasan adalah hal yang harus direncanakan pemanggil

(petugas pengiriman yang memilih kurir lain), *unchecked* bila pemanggil yang benar selalu memeriksa lebih dulu.

11. Lihat `insertPersistent`. Menyisipkan 9 ke pohon **Gambar 8.6** menciptakan empat simpul baru: salinan baru 5, 8, dan 10 di sepanjang lintasan, serta daun 9. Separuh kiri `3(1,4)` dan daun 7 dipakai bersama, seperti ditegaskan pemeriksaan `8.11 left half shared`, dan pohon lama masih mencetak `5(3(1,4),8(7,10))`. Berbagi seperti itu aman hanya karena tidak ada yang mengubah simpul setelah diciptakan; itulah sebabnya rancangan ini juga cocok untuk `List` yang imutabel pada Subbab 8.3.2.

12. Aturannya [2]: pakailah eksepsi *checked* untuk keadaan yang harus diantisipasi program yang benar dan dapat dipulihkan (baris rusak dalam berkas yang diketik manusia, berkas yang tidak ada); pakailah eksepsi *unchecked* untuk prasyarat yang dilanggar, yaitu galat pemrograman di pihak pemanggil (`head()` dari *list* kosong, indeks negatif). Berat negatif yang diberikan ke `priceFor` adalah galat pemanggil, karena berat seharusnya sudah divalidasi ketika pesanan dibaca, sehingga yang tepat adalah `IllegalArgumentException` yang *unchecked*, seperti pada **Kode 8.43**.

Daftar Pustaka

[1] Python Software Foundation, “unittest — Unit testing framework,” *The Python Standard Library*, Python 3.14. [Online]. Available: https://docs.python.org/3/library/unittest.html

[2] J. Bloch, *Effective Java*, 3rd ed. Boston, MA, USA: Addison-Wesley, 2018.

[3] G. Booch, J. Rumbaugh, and I. Jacobson, *The Unified Modeling Language User Guide*, 2nd ed. Boston, MA, USA: Addison-Wesley, 2005.

[4] H. Abelson and G. J. Sussman, *Structure and Interpretation of Computer Programs*, 2nd ed. Cambridge, MA, USA: MIT Press, 1996.

[5] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 4th ed. Cambridge, MA, USA: MIT Press, 2022.

[6] Oracle, “The Java Tutorials: Exceptions” and “The try-with-resources Statement.” [Online]. Available: https://docs.oracle.com/javase/tutorial/essential/exceptions/

[7] Python Software Foundation, “Errors and Exceptions” and “Reading and Writing Files,” *The Python Tutorial*, Python 3.14. [Online]. Available: https://docs.python.org/3/tutorial/

[8] Python Software Foundation, “abc — Abstract Base Classes” and “functools — Higher-order functions and operations on callable objects,” *The Python Standard Library*, Python 3.14. [Online]. Available: https://docs.python.org/3/library/

[9] The JUnit Team, *JUnit User Guide*, version 6.1.3, Aug. 2026. Accessed: Sep. 24, 2026. [Online]. Available: https://docs.junit.org/current/

[10] J. Gosling, B. Joy, G. Steele, G. Bracha, A. Buckley, D. Smith, and G. Bierman, *The Java Language Specification, Java SE 21 Edition*. Oracle America, 2023. [Online]. Available: https://docs.oracle.com/javase/specs/

[11] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Reading, MA, USA: Addison-Wesley, 1994.

[12] M. Fowler, *Refactoring: Improving the Design of Existing Code*, 2nd ed. Boston, MA, USA: Addison-Wesley, 2018.

Aturan untuk memilih eksepsi *checked* dan *unchecked*, saran tentang salinan defensif dan imutabilitas, serta pengutamaan antarmuka di atas pewarisan mengikuti Bloch [2]; penyelesaian metode yang di-*overload* pada saat kompilasi dan keempat tingkat akses didefinisikan dalam spesifikasi bahasa [10], dan penanganan eksepsi dengan *try-with-resources* diuraikan dalam tutorial Java [6]. Uji Python memakai `unittest` [1], *abstract base class* dan `singledispatch` [8],

serta penanganan berkas dengan `with` [7]; versi JUnit 6 mengikuti panduan penggunanya [9]. *Refactoring extract superclass* pada Soal 8.20 berasal dari Fowler [12], dan *template method* pada Soal 8.19 dan 8.20 dari Gamma dkk. [11]. *List cons* dan kaitan antara rekursi dan induksi berasal dari Abelson dan Sussman [4]; pohon pencarian biner, penggabungan, dan laju pertumbuhan pada Soal 8.6, 8.13, dan 8.15 dibahas dalam Cormen dkk. [5]; dan diagram kelas memakai notasi dari panduan pengguna UML [3].

Bab 9

Koleksi dan Generik

Pada akhir bab ini Anda akan sanggup memilih, untuk setiap pekerjaan yang melibatkan banyak objek, wadah yang mengerjakannya dengan baik, lalu mempertahankan pilihan itu dengan angka. Anda akan memakai *Collections Framework* milik Java — antarmuka `List`, `Set`, `Map`, `Queue`, dan `Deque` beserta kelas-kelas andalannya, `ArrayList`, `LinkedList`, `HashSet`, `LinkedHashSet`, `TreeSet`, `HashMap`, `LinkedHashMap`, `TreeMap`, `ArrayDeque`, dan `PriorityQueue` — berdampingan dengan `list`, `tuple`, `set`, `frozenset`, `dict`, `deque`, dan `heapq` milik Python. Anda akan menelusuri sebuah koleksi dengan aman, menghapus isinya tanpa tersandung kaki sendiri, dan mengurutkannya menurut aturan apa pun dengan `Comparable`, `Comparator`, serta fungsi `key=` milik Python. Anda akan mempelajari generik (*generics*) dari nol: apa itu parameter tipe, bagaimana menulis kelas generik dan metode generik, apa yang diberikan oleh tipe terbatas seperti `<T extends Comparable<T>>`, mengapa `List<Cat>` bukan `List<Animal>` dan bagaimana *wildcard* `? extends` dan `? super` memperbaikinya, serta apa yang tersisa dari *type erasure* (penghapusan tipe) ketika program berjalan; dan Anda akan menjumpai gagasan yang sama pada modul `typing` Python, yang diperiksa oleh perkakas mypy. Di sepanjang jalan kita memungut dua perkakas kecil yang diperlukan bagian lain bab ini: konstanta yang dideklarasikan di dalam antarmuka, dan metode rekursif yang memakai pembantu. Semuanya bertemu dalam sebuah buku pesanan untuk Profitina Laundry yang dapat menemukan pesanan mana pun menurut nomornya, pelanggannya, atau statusnya, menjaga antrean cucian yang menunggu dengan pesanan kilat di depan, dan mencetak laporan pendapatan yang terurut.

9.1 Di Mana Kita Berhenti

Bab 5 memberi kita larik dan list tumbuh kita yang pertama, `ArrayList`, sambil berjanji bahwa `<String>` yang misterius pada `ArrayList<String>` akan dijelaskan di Bab 9. Inilah bab itu. Bab 6 memperlihatkan bahwa *antarmuka* adalah janji tanpa badan, bahwa `List` adalah antarmuka semacam itu dan `ArrayList` salah satu implementasinya, serta bahwa sebuah kelas dapat menjanjikan urutan alami dengan mengimplementasikan `Comparable`. Bab 7 membangun hierarki kelas, sehingga satu variabel bertipe `Service` dapat memegang layanan cuci, setrika, atau kilat, dan Bab 8 melatih semua itu pada struktur rekursif, termasuk list imutabel kecil berisi bilangan bulat yang dibangun dari `nil` dan `cons`.

Sekarang lihatlah binatu itu pada akhir hari Sabtu yang sibuk. Ada beberapa ratus pesanan. Pemiliknya mengajukan empat pertanyaan dalam satu tarikan napas: *Di mana pesanan L-217? Apa saja yang dititipkan Bu Sari? Pesanan mana yang siap diambil? Kantong mana yang masuk ke mesin berikutnya yang kosong?* Dengan bekal kita sejauh ini, setiap pertanyaan adalah perulangan atas satu list besar berisi semua pesanan, memeriksa pesanan satu per satu. Cara itu jalan untuk sepuluh pesanan. Untuk sepuluh ribu pesanan, yang ditanyakan beberapa ratus kali sehari, itulah beda antara jawaban yang muncul seketika dan antrean di loket yang mengular sampai ke jalan.

Obatnya bukan perulangan yang lebih cerdik, melainkan *wadah* yang lebih baik: sebuah *map* dari nomor pesanan ke pesanan untuk “temukan L-217”, sebuah map dari pelanggan ke pesanan-pesanannya untuk “apa saja titipan Bu Sari”, sebuah *set* (himpunan) untuk setiap status untuk “mana yang siap”, dan sebuah *priority queue* (antrean prioritas) untuk “kantong mana berikutnya,

yang kilat dulu". Java menyebut keluarga wadah ini **Collections Framework**; Python membangun sebagian besarnya langsung ke dalam bahasanya. Bab ini mempelajari keluarga itu anggota demi anggota, lalu memperlihatkan cara memilih anggota yang tepat dengan menghitung pekerjaan yang dilakukan masing-masing.

Paruh kedua ceritanya adalah bagaimana satu kelas wadah dapat menampung `String` dalam satu program dan `LaundryOrder` dalam program lain, sementara kompilator tetap memeriksa setiap pemakaiannya. Itulah **generik**, gagasan di balik `<String>`.

> ✓ **WAWASAN KUNCI.** Sebuah koleksi adalah jawaban atas *pertanyaan yang akan sering Anda ajukan*. Tanyakan "menurut apa saya akan mencari, dan dalam urutan apa saya membutuhkannya?" sebelum bertanya "di depan kelas mana saya harus menulis `new`?". Pertanyaanlah yang memilih koleksi; koleksi membuat pertanyaan itu murah.

9.2 Dua Perkakas Bekal: Konstanta dalam Antarmuka dan Pembantu Rekursif

Sebelum membuka koleksi, kita memungut dua perkakas kecil. Yang pertama adalah fakta tentang antarmuka yang belum diperlukan di Bab 6: antarmuka dapat memuat *konstanta* selain kepala metode. Yang kedua adalah cara menulis metode rekursif yang menjaga pemanggilnya tetap sederhana, yang nanti kita pakai untuk list generik.

9.2.1 Konstanta dalam antarmuka

Ada nilai yang tidak pernah berubah selama program berjalan: banyaknya kilogram dalam satu pon, poin untuk sebuah kemenangan, kode yang mewakili hasil sebuah pertandingan. Bab 4 menulis nilai semacam itu sebagai field `public static final` sebuah kelas, misalnya `KILOGRAMS_PER_POUND`. Antarmuka pun dapat memuatnya. Di dalam antarmuka `Measurable` kita dapat menulis

```
int LOSE = 0;
int DRAW = 1;
int WIN = 2;
```

lalu memakainya di mana saja sebagai `Measurable.WIN`. Anda tidak boleh mendeklarasikan field ini sebagai `public` atau `private` sendiri: setiap field antarmuka **otomatis** bersifat `public static final` [1]. `public`, karena antarmuka adalah janji publik; `static`, karena antarmuka tidak punya objek sehingga nilainya milik antarmuka itu sendiri; `final`, karena antarmuka tidak dapat menyimpan keadaan yang berubah. Kelas yang mengimplementasikan antarmuka itu mewarisi konstantanya dan boleh memakainya tanpa awalan, sama seperti ia mewarisi kepala-kepala metodenya.

Kode 9.1 mempraktikkan gagasan ini. `Measurable` membawa tiga kode hasil pertandingan dan metode `getMeasure()` dari Bab 6; `Team` mengimplementasikannya dan mengukur sebuah tim dengan poin klasemennya; antarmuka kedua, `Units`, memuat sebuah faktor konversi.

Kode 9.1 `InterfaceConstants.java`: konstanta di dalam antarmuka `Measurable` dan `Units`, dipakai dengan dan tanpa awalan, serta sebuah `enum` sebagai alternatif modern.

```
import java.util.Locale;

public class InterfaceConstants {
    public static void main(String[] args) {
        System.out.println("LOSE = " + Measurable.LOSE + ", DRAW = " + Measurable.DRAW
                + ", WIN = " + Measurable.WIN);

        int[] results = {Measurable.WIN, Measurable.DRAW, Measurable.WIN, Measurable.LOSE};
        Team team = new Team("Surabaya Stars", results);
        System.out.println(team.getName() + " scored " + team.getMeasure() + " points");

        double pounds = 11.0;
        System.out.println(String.format(Locale.US, "%.1f lb = %.3f kg",
                pounds, pounds * Units.KG_PER_POUND));

        // Measurable.WIN = 5;   // tidak lolos kompilasi: WIN final

        // Enum adalah rumah modern bagi himpunan tetap nilai bernama.
        for (Outcome o : Outcome.values()) {
            System.out.println(o + ": " + o.points() + " pt");
        }
        System.out.println("Outcome.valueOf(\"DRAW\") = " + Outcome.valueOf("DRAW"));
    }
}

interface Measurable {
    int LOSE = 0;          // otomatis public static final
    int DRAW = 1;
    int WIN = 2;

    double getMeasure();
}

interface Units {
    double KG_PER_POUND = 0.45359237;   // eksak, menurut kesepakatan internasional
}

class Team implements Measurable {
    private final String name;
    private final int[] results;

    Team(String name, int[] results) {
        this.name = name;
        this.results = results.clone();
    }
```

Kode 9.1 (Lanjutan) *InterfaceConstants.java: konstanta di dalam antarmuka Measurable dan Units, dipakai dengan dan tanpa awalan, serta sebuah enum sebagai alternatif modern.*

```
    String getName() {
        return name;
    }

    @Override
    public double getMeasure() {
        int points = 0;
        for (int r : results) {
            if (r == WIN) {             // konstanta warisan: tanpa awalan
                points += 3;
            } else if (r == DRAW) {
                points += 1;
            }
        }
        return points;
    }
}

enum Outcome {
    LOSE(0), DRAW(1), WIN(3);

    private final int points;

    Outcome(int points) {
        this.points = points;
    }

    int points() {
        return points;
    }
}
```

```
LOSE = 0, DRAW = 1, WIN = 2
Surabaya Stars scored 7.0 points
11.0 lb = 4.990 kg
LOSE: 0 pt
DRAW: 1 pt
WIN: 3 pt
Outcome.valueOf("DRAW") = DRAW
```

Bacalah listing ini dari atas. Di dalam `main`, `Measurable.LOSE`, `Measurable.DRAW`, dan `Measurable.WIN` dibaca melalui nama antarmuka, persis seperti field statis sebuah kelas. Larik `results` memuat empat hasil pertandingan, dan `Team.getMeasure()` mengubahnya menjadi poin klasemen: tiga untuk menang, satu untuk seri. Di dalam `Team` perbandingannya berbunyi `r == WIN` dan `r == DRAW` tanpa awalan, karena `Team implements Measurable` sehingga mewarisi nama-nama itu. Dua kemenangan dan satu seri memberi $3 + 1 + 3 + 0 = 7$ poin. Konversinya memakai `Units.KG_PER_POUND`, nilai internasional yang eksak 0.45359237, sehingga sebelas pon adalah $11 \times 0.45359237 = 4.98951607$ kg, dicetak dengan tiga angka desimal sebagai `4.990`.

Baris yang dijadikan komentar, `// Measurable.WIN = 5;`, memperlihatkan arti `final`. Hapus kedua garis miringnya dan `javac` menolak program itu:

```
InterfaceConstants.java:16: error: cannot assign a value to static final variable WIN
        Measurable.WIN = 5;
                   ^
1 error
```

Konstanta antarmuka punya satu kelemahan: kode-kode hasil pertandingan hanyalah `int` biasa, sehingga tidak ada yang mencegah pemanggil mengirim `7`, dan tidak ada yang mengikat poin kemenangan dengan kode kemenangan. **Enum** (enumerasi) Java menyelesaikan keduanya: `enum`

`Outcome { LOSE(0), DRAW(1), WIN(3); ... }` mendeklarasikan tipe yang tepat memiliki tiga kemungkinan nilai, masing-masing sebuah objek yang dapat membawa data (`points`) dan metode (`points()`). Metode yang menerima `Outcome` tidak dapat menerima yang lain. `Outcome.values()` mendaftar konstanta menurut urutan deklarasinya, dan `Outcome.valueOf("DRAW")` menemukan satu konstanta menurut namanya. Saran rujukan baku tentang gaya Java jelas: pakailah antarmuka untuk mendefinisikan tipe, dan taruh sekelompok konstanta yang berkaitan di dalam enum atau kelas utilitas kecil, bukan di dalam antarmuka yang diimplementasikan kelas hanya untuk mewarisi nama-namanya [2]. Konstanta antarmuka tetap perlu dikenal, karena banyak kode yang sudah ada memakainya.

Python tidak punya kata kunci `final` dan tidak punya konstanta antarmuka dalam arti Java. **Kode 9.2** memperlihatkan tiga hal yang dimilikinya sebagai gantinya.

Kode 9.2 `interface_constants.py`: atribut kelas yang ditandai `Final`, konstanta modul, dan sebuah `Enum`, yang anggotanya tidak dapat ditugasi ulang.

```
"""interface_constants.py: konstanta bernama di Python, dan mengapa Enum lebih aman."""
from abc import ABC, abstractmethod
from enum import Enum
from typing import Final

KG_PER_POUND: Final = 0.45359237   # eksak, menurut kesepakatan internasional


class Measurable(ABC):
    LOSE: Final = 0   # Final adalah janji yang diperiksa mypy, bukan Python
    DRAW: Final = 1
    WIN: Final = 2

    @abstractmethod
    def get_measure(self) -> float: ...


class Team(Measurable):
    def __init__(self, name, results):
        self.name = name
        self._results = list(results)

    def get_measure(self):
        points = 0
        for r in self._results:
            if r == self.WIN:          # atribut kelas warisan
                points += 3
            elif r == self.DRAW:
                points += 1
        return float(points)


class Outcome(Enum):
    LOSE = 0
    DRAW = 1
    WIN = 3

    def points(self):
        return self.value


def main() -> None:
    print(f"LOSE = {Measurable.LOSE}, DRAW = {Measurable.DRAW}, WIN = {Measurable.WIN}")
```

Kode 9.2 (Lanjutan) *interface_constants.py: atribut kelas yang ditandai Final, konstanta modul, dan sebuah Enum, yang anggotanya tidak dapat ditugasi ulang.*

```
    results = [Measurable.WIN, Measurable.DRAW, Measurable.WIN, Measurable.LOSE]
    team = Team("Surabaya Stars", results)
    print(f"{team.name} scored {team.get_measure()} points")

    pounds = 11.0
    print(f"{pounds:.1f} lb = {pounds * KG_PER_POUND:.3f} kg")

    # Enum adalah rumah modern bagi himpunan tetap nilai bernama.
    for o in Outcome:
        print(f"{o.name}: {o.points()} pt")
    print(f'Outcome["DRAW"] = {Outcome["DRAW"].name}')

    # Python tidak mencegah penugasan ini saat eksekusi ...
    Measurable.WIN = 5
    print("after Measurable.WIN = 5:", team.get_measure(), "points")
    # ... tetapi anggota Enum tidak dapat ditugasi ulang.
    try:
        Outcome.WIN = 5
    except AttributeError as e:
        print("AttributeError:", e)


if __name__ == "__main__":
    main()
```

```
LOSE = 0, DRAW = 1, WIN = 2
Surabaya Stars scored 7.0 points
11.0 lb = 4.990 kg
LOSE: 0 pt
DRAW: 1 pt
WIN: 3 pt
Outcome["DRAW"] = DRAW
after Measurable.WIN = 5: 1.0 points
AttributeError: cannot reassign member 'WIN'
```

Tujuh baris pertama sama persis dengan Java. Dua baris terakhir memperlihatkan perbedaannya. `Measurable.WIN = 5` berjalan tanpa keluhan: atribut kelas di Python adalah variabel biasa yang boleh diikat ulang oleh siapa saja. Lebih buruk lagi, penugasan itu diam-diam merusak skor tim, yang turun dari 7 menjadi 1, karena list `results` masih memuat kode lama 2 sementara `WIN` kini berarti 5, sehingga hanya hasil seri yang dikenali. Anotasi `Final` adalah janji yang ditegakkan oleh *pemeriksa tipe*, bukan oleh interpreter. Jalankan mypy, pemeriksa statis baku untuk petunjuk tipe (*type hints*) Python [3], dan ia melaporkan kedua penugasan itu sebelum program sempat berjalan:

```
interface_constants.py:58: error: Cannot assign to final attribute "WIN"  [misc]
interface_constants.py:62: error: Cannot assign to final attribute "WIN"  [misc]
Found 2 errors in 1 file (checked 1 source file)
```

Sebaliknya, anggota `Enum` melindungi dirinya sendiri saat eksekusi: `Outcome.WIN = 5` memunculkan `AttributeError`. Seperti di Java, enum adalah rumah paling aman bagi himpunan tetap nilai-nilai bernama.

> ⚠ **PERINGATAN.** mypy hanya memeriksa fungsi yang membawa petunjuk tipe. Itulah sebabnya `main` di **Kode 9.2** dideklarasikan `def main() -> None:`; tanpa `-> None`, mypy akan melewati badannya dan tidak melaporkan apa pun. Bila Anda ingin sebuah program Python diperiksa, anotasilah fungsi-fungsi yang ingin diperiksa.

9.2.2 Rekursi sekali lagi: kasus dasar dan panggilan rekursif

Bab 8 memakai rekursi pada pohon dan list. Di sini kita melihatnya sekali lagi pada bilangan biasa, karena salah satu tekniknya, *metode pembantu* (*helper method*), muncul kembali pada list generik di Subbab 9.8.

Metode rekursif hanyalah metode yang memanggil dirinya sendiri. Rekursi dapat memberi penyelesaian yang jauh lebih anggun daripada perulangan, tetapi pada pandangan pertama juga dapat membingungkan: mudah sekali menatap metode rekursif lalu kehilangan jejak apa yang sedang terjadi, sementara versi perulangannya terbaca lurus dari atas ke bawah. Cara terbaik memikirkan rekursi adalah bahwa ia memecah satu masalah besar menjadi masalah sejenis yang lebih kecil ditambah sedikit pekerjaan — *divide and conquer* (bagi dan kuasai). Setiap metode rekursif memiliki dua bagian:

- **kasus dasar** (*base case*), versi paling sederhana dari masalahnya, yang dijawab langsung; inilah lapisan terdalam dari rangkaian panggilan dan inilah yang menghentikan pengulangan;
- **panggilan rekursif** (*recursive call*), tempat metode memanggil dirinya sendiri dengan argumen yang *berbeda* dan lebih kecil, sehingga prosesnya terus bergerak menuju kasus dasar.

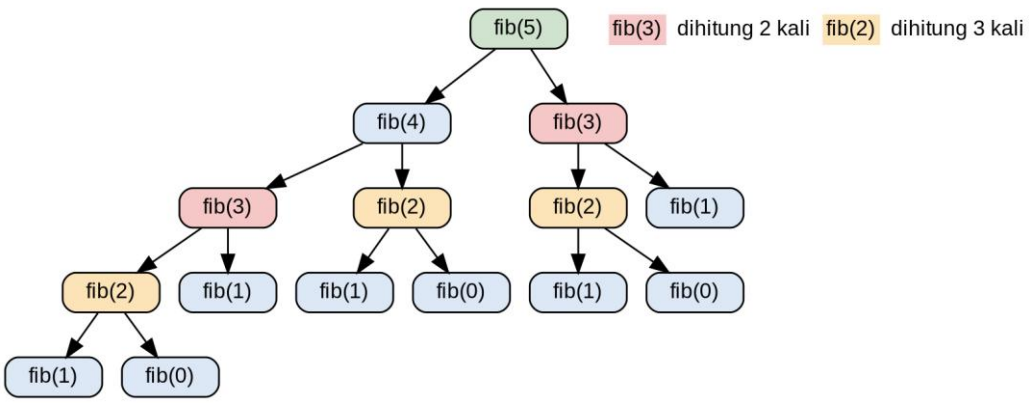
Hilangkan kasus dasar, atau tulis panggilan rekursif yang tidak bergerak menujunya, maka panggilan tidak akan pernah berhenti: Java melempar `StackOverflowError`, Python `RecursionError`.

Contoh terkecil menghitung mundur. `printNums(x)` mencetak `x,` lalu meminta `printNums(x - 1)` mencetak sisanya; `printNums(0)` adalah kasus dasar dan hanya mencetak `0`. Jadi `printNums(10)` mencetak `10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0`.

Contoh klasiknya adalah barisan Fibonacci 0,1,1,2,3,5,8,13, ..., yang setiap bilangannya adalah jumlah dua bilangan sebelumnya:

$$F(0) = 0, \qquad F(1) = 1, \qquad F(n) = F(n-1) + F(n-2) \text{ untuk } n \geq 2 \tag{9.1}$$

Persamaan (9.1) dapat diketik ke dalam Java hampir kata demi kata, dan versi "standar" itu muncul di **Kode 9.3** sebagai `fibonacci(int n)`: dua kasus dasar, lalu dua panggilan rekursif. Versi itu benar. Ia juga luar biasa lambat, dan alasannya layak dilihat. **Gambar 9.1** menggambar setiap panggilan yang dilakukan oleh `fibonacci(5)`.



Gambar 9.1 Panggilan-panggilan yang dilakukan `fibonacci(5)` rekursif standar: lima belas panggilan, karena submasalah yang sama diselesaikan berulang-ulang — `fib(3)` dua kali dan `fib(2)` tiga kali.

`fib(5)` meminta `fib(4)` dan `fib(3)`; tetapi `fib(4)` *juga* meminta `fib(3)`, dan tidak ada yang mengingat jawaban pertama. Misalkan $C(n)$ adalah banyaknya panggilan untuk menghitung $F(n)$.

Satu panggilan dilakukan untuk n itu sendiri, ditambah panggilan-panggilan dari kedua panggilan rekursifnya:

$$C(0) = C(1) = 1, \qquad C(n) = 1 + C(n-1) + C(n-2) \quad \Rightarrow \quad C(n) = 2F(n+1) - 1 \tag{9.2}$$

Bentuk tertutup di sebelah kanan mudah diperiksa dengan induksi: ia berlaku untuk $n = 0$ dan $n = 1$ karena $2F(1) - 1 = 2F(2) - 1 = 1$, dan bila berlaku untuk $n-1$ dan $n-2$ maka $C(n) = 1 + (2F(n) - 1) + (2F(n-1) - 1) = 2(F(n) + F(n-1)) - 1 = 2F(n+1) - 1$. Untuk $n = 5$ hasilnya $2 \times 8 - 1 = 15$, kelima belas kotak di **Gambar 9.1**. Untuk $n = 30$ hasilnya $2 \times 1{,}346{,}269 - 1 = 2{,}692{,}537$ panggilan demi menghitung bilangan yang dapat dihitung seorang anak dengan tiga puluh penjumlahan. Karena $F(n)$ tumbuh seperti $\varphi^n/\sqrt{5}$ dengan $\varphi = (1+\sqrt{5})/2 \approx 1.618$, pekerjaannya tumbuh secara eksponensial: setiap langkah tambahan melipatkannya sekitar 1,6 kali.

9.2.3 Metode pembantu dengan parameter tambahan

Versi lambat itu menghitung ulang karena setiap panggilan melupakan apa yang telah dipelajari panggilan lain. Perbaikannya adalah *membawa* dua nilai terakhir sebagai parameter, sehingga setiap langkah hanya memerlukan satu panggilan rekursif:

```
static long fibonacci(int n, long sum, long prev, int count) {
    if (count >= n - 1) {
        return sum + prev;
    }
    return fibonacci(n, sum + prev, sum, count + 1);  // satu panggilan, bukan dua
}
```

`sum` memegang $F(count)$ dan `prev` memegang $F(count-1)$. Setiap panggilan maju satu langkah di sepanjang barisan — `sum` yang baru adalah `sum + prev`, `prev` yang baru adalah `sum` yang lama — sampai `count` mencapai $n-1$, ketika `sum + prev` sama dengan $F(n)$. Panggilan rekursifnya adalah hal paling akhir yang dikerjakan metode, yang disebut **rekursi ekor** (*tail recursion*). Seluruh perhitungan hanya melakukan sekitar n panggilan, bukan $2F(n+1) - 1$.

Kelemahannya ada pada cara memanggil. Untuk menghitung $F(10)$ kita harus menulis `fibonacci(10, 1, 0, 1)`, dan pemanggil yang mengetik `fibonacci(10, 0, 0, 0)` mendapat jawaban salah tanpa peringatan apa pun. Penyelesaiannya adalah **metode pembantu**: metode kedua yang sederhana, yang menyediakan parameter tambahan dan yang sesungguhnya dipakai pemanggil. Di **Kode 9.3** metode itu bernama `fib(int n)`; ia mengembalikan `fibonacci(n, 1, 0, 1)`, dan pemanggil cukup menulis `fib(10)`. Gagasan yang sama muncul pada setiap metode rekursif yang memerlukan *akumulator*: metode publik dengan tanda tangan yang ramah, dan pembantu privat yang mengerjakan rekursi dengan keadaan tambahan. `toString` pada list generik di Subbab 9.8.2 dibangun persis dengan cara ini.

Pembaca yang cermat akan memperhatikan bahwa kasus dasarnya ditulis `count >= n - 1`, bukan `count == n - 1`. Dengan `==`, panggilan untuk $n = 1$ dimulai dengan `count` bernilai 1 dan $n - 1 = 0$; `count` hanya bertambah, sehingga tidak pernah sama dengan 0, kasus dasar tidak pernah tercapai, dan program mati dengan `StackOverflowError`. Tanda `>=` membuat kasus dasar tercapai untuk setiap $n \geq 1$, dan `fib` menangani $n = 0$ sendiri. *Kasus dasar baru pantas disebut kasus dasar bila setiap rantai panggilan mencapainya.*

Cara ketiga mengembalikan *dua* nilai dari setiap panggilan, yaitu pasangan $\{F(n), F(n-1)\}$; pasangan untuk n hanya berjarak satu penjumlahan dari pasangan untuk $n-1$. Algoritma

berwaktu linear ini adalah kegemaran Goodrich, Tamassia, dan Goldwasser [4]; di Java ia mengembalikan `long[]` sepanjang 2.

Kode 9.3 `Fibonacci.java`: hitung mundur, Fibonacci standar dengan dua panggilan dan penghitung panggilan, versi rekursif ekor di balik metode pembantu, dan versi linear yang mengembalikan dua nilai sekaligus.

```
public class Fibonacci {
    static long calls = 0;   // berapa kali fibonacci(int) telah dimasuki

    // Menghitung mundur ke 0: kasus dasar, lalu panggilan rekursif pada masalah yang lebih
kecil.
    static void printNums(int x) {
        if (x == 0) {
            System.out.println(0);
        } else {
            System.out.print(x + ", ");
            printNums(x - 1);
        }
    }

    // Versi "standar": dua panggilan rekursif per langkah.
    static long fibonacci(int n) {
        calls++;
        if (n == 0) {
            return 0;
        } else if (n == 1) {
            return 1;
        } else {
            return fibonacci(n - 1) + fibonacci(n - 2);
        }
    }

    // Versi rekursif ekor: parameter tambahan membawa hasil kerja.
    static long fibonacci(int n, long sum, long prev, int count) {
        if (count >= n - 1) {
            return sum + prev;
        }
        return fibonacci(n, sum + prev, sum, count + 1);
    }

    // Metode pembantu menyembunyikan parameter tambahan dari pemanggil.
    static long fib(int n) {
        if (n == 0) {
            return 0;
        }
        return fibonacci(n, 1, 0, 1);
    }
```

Kode 9.3 (Lanjutan) *Fibonacci.java: hitung mundur, Fibonacci standar dengan dua panggilan dan penghitung panggilan, versi rekursif ekor di balik metode pembantu, dan versi linear yang mengembalikan dua nilai sekaligus.*

```
    // Waktu linear dengan mengembalikan dua nilai sekaligus: {F(n), F(n-1)}.
    static long[] fibonacciPair(int n) {
        if (n <= 1) {
            long[] answer = {n, 0};
            return answer;
        } else {
            long[] tmp = fibonacciPair(n - 1);
            long[] answer = {tmp[0] + tmp[1], tmp[0]};
            return answer;
        }
    }

    public static void main(String[] args) {
        printNums(10);

        for (int n : new int[] {10, 20, 30}) {
            calls = 0;
            long f = fibonacci(n);
            System.out.println("fibonacci(" + n + ") = " + f + " after " + calls + "
calls");
        }
        System.out.println("fibonacci(10, 1, 0, 1) = " + fibonacci(10, 1, 0, 1));
        System.out.println("fib(10) = " + fib(10) + ", fib(1) = " + fib(1) + ", fib(0) = "
+ fib(0));
        System.out.println("fibonacciPair(10)[0] = " + fibonacciPair(10)[0]);
        System.out.println("fibonacciPair(90)[0] = " + fibonacciPair(90)[0]);
    }
}
```

```
10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0
fibonacci(10) = 55 after 177 calls
fibonacci(20) = 6765 after 21891 calls
fibonacci(30) = 832040 after 2692537 calls
fibonacci(10, 1, 0, 1) = 55
fib(10) = 55, fib(1) = 1, fib(0) = 0
fibonacciPair(10)[0] = 55
fibonacciPair(90)[0] = 2880067194370816120
```

Jumlah panggilan dalam keluaran persis seperti Persamaan (9.2): $2F(11) - 1 = 2 \times 89 - 1 = 177$, $2F(21) - 1 = 21{,}891$, dan $2F(31) - 1 = 2{,}692{,}537$. Tiga metode lainnya sepakat dengan metode yang lambat dan menghitung $F(90) = 2{,}880{,}067{,}194{,}370{,}816{,}120$ dalam sembilan puluh langkah; metode lambat akan memerlukan sekitar 9×10^{18} panggilan, beberapa ratus tahun pada sebuah laptop.

Perhatikan bagian yang canggung dari `fibonacciPair`: ia mengembalikan larik hanya karena sebuah metode cuma dapat mengembalikan satu hal, dan pemanggil harus ingat bahwa indeks 0 berarti $F(n)$ dan indeks 1 berarti $F(n-1)$. Sebuah kelas kecil yang memegang dua nilai dengan tipe pilihan, sebuah `Pair`, akan mengatakan apa maksudnya. Subbab 9.8.3 menulis kelas itu, sekali saja, untuk *sembarang* dua tipe — tepat masalah yang dipecahkan oleh generik.

Kode 9.4 adalah versi Python-nya.

Kode 9.4 `fibonacci.py`: keempat fungsi yang sama dalam Python, ditambah versi standar yang dibuat cepat oleh `functools.lru_cache`.

```python
"""fibonacci.py: rekursi, fungsi pembantu, dan Fibonacci berwaktu linear."""
from functools import lru_cache

calls = 0   # berapa kali fibonacci() telah dimasuki


def print_nums(x):
    """Menghitung mundur ke 0: kasus dasar, lalu panggilan rekursif pada masalah yang lebih
kecil."""
    if x == 0:
        print(0)
    else:
        print(x, end=", ")
        print_nums(x - 1)


def fibonacci(n):
    """Versi "standar": dua panggilan rekursif per langkah."""
    global calls
    calls += 1
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fibonacci(n - 1) + fibonacci(n - 2)


def fibonacci_tail(n, total, prev, count):
    """Versi rekursif ekor: parameter tambahan membawa hasil kerja."""
    if count >= n - 1:
        return total + prev
    return fibonacci_tail(n, total + prev, total, count + 1)


def fib(n):
    """Fungsi pembantu menyembunyikan parameter tambahan dari pemanggil."""
    if n == 0:
        return 0
    return fibonacci_tail(n, 1, 0, 1)
```

Kode 9.4 (Lanjutan) *fibonacci.py: keempat fungsi yang sama dalam Python, ditambah versi standar yang dibuat cepat oleh functools.lru_cache.*

```
def fibonacci_pair(n):
    """Waktu linear dengan mengembalikan dua nilai sekaligus: (F(n), F(n-1))."""
    if n <= 1:
        return (n, 0)
    f1, f2 = fibonacci_pair(n - 1)
    return (f1 + f2, f1)


@lru_cache(maxsize=None)
def fibonacci_cached(n):
    """Versi standar lagi, tetapi setiap jawaban diingat."""
    return n if n <= 1 else fibonacci_cached(n - 1) + fibonacci_cached(n - 2)


def main():
    global calls
    print_nums(10)

    for n in (10, 20, 30):
        calls = 0
        f = fibonacci(n)
        print(f"fibonacci({n}) = {f} after {calls} calls")
    print("fibonacci(10, 1, 0, 1) =", fibonacci_tail(10, 1, 0, 1))
    print(f"fib(10) = {fib(10)}, fib(1) = {fib(1)}, fib(0) = {fib(0)}")
    print("fibonacciPair(10)[0] =", fibonacci_pair(10)[0])
    print("fibonacciPair(90)[0] =", fibonacci_pair(90)[0])
    print("fibonacci_cached(90) =", fibonacci_cached(90))


if __name__ == "__main__":
    main()
```

```
10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0
fibonacci(10) = 55 after 177 calls
fibonacci(20) = 6765 after 21891 calls
fibonacci(30) = 832040 after 2692537 calls
fibonacci(10, 1, 0, 1) = 55
fib(10) = 55, fib(1) = 1, fib(0) = 0
fibonacciPair(10)[0] = 55
fibonacciPair(90)[0] = 2880067194370816120
fibonacci_cached(90) = 2880067194370816120
```

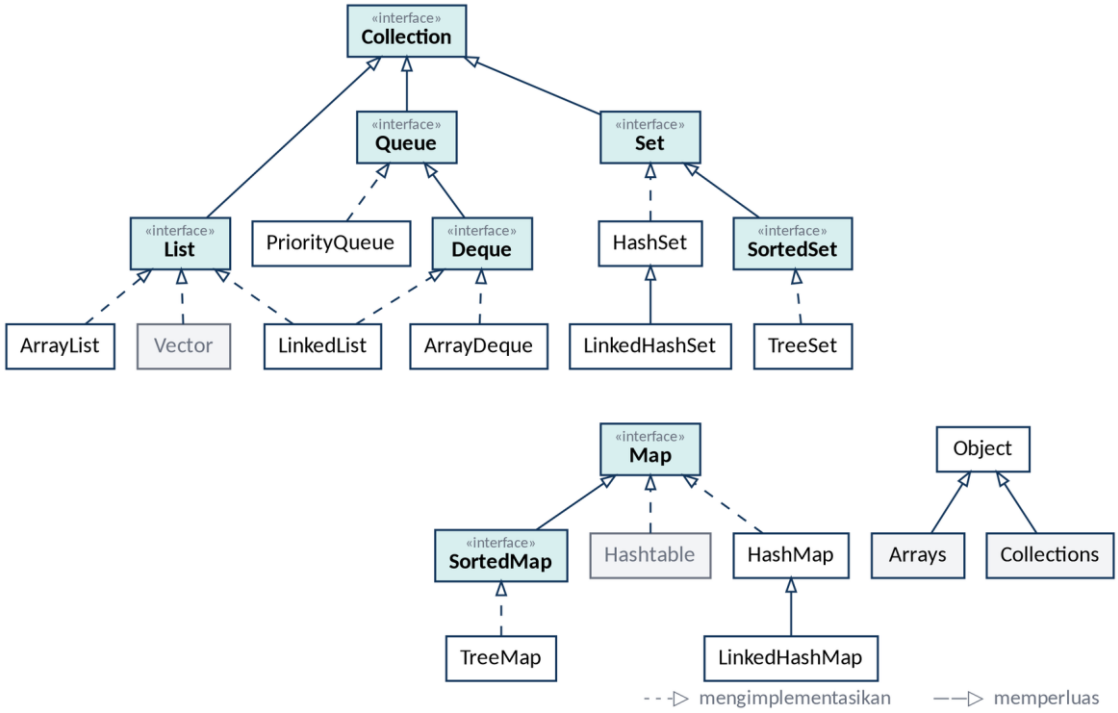
Python tidak punya *overloading* (Bab 6), sehingga fungsi rekursif ekor memerlukan namanya sendiri, `fibonacci_tail`; label yang dicetak dibuat sama dengan milik Java agar keluarannya dapat dibandingkan. Python mengembalikan dua nilai secara alami sebagai *tuple*, `(n, 0)`, yang dibongkar dengan `f1, f2 = fibonacci_pair(n - 1)`. Dekorator `@lru_cache` membuat sebuah fungsi *mengingat* setiap jawabannya di dalam kamus yang berkunci argumen, sehingga definisi naif dengan dua panggilan pun berjalan dalam waktu linear — sebuah koleksi yang menjawab "pernahkah saya melihat ini?" dalam satu langkah.

> ⚠ **PERINGATAN.** Baik Java maupun Python tidak menghapus panggilan ekor: setiap panggilan rekursif menempati satu bingkai di tumpukan panggilan sampai ia kembali, dan Python secara bawaan berhenti pada kedalaman sekitar 1000. Pakailah rekursi bila kedalamannya kecil, seperti pada pohon; untuk sejuta langkah, tulislah perulangan.

9.3 Collections Framework: Peta Wilayahnya

Koleksi (*collection*) adalah objek yang memegang objek-objek lain, yaitu *elemen*-elemennya, dan menawarkan operasi atas mereka sebagai satu kelompok: menambah satu, menghapus satu, menanyakan apakah suatu elemen ada, menghitungnya, mengunjungi semuanya. Larik adalah koleksi dari jenis yang paling primitif. **Collections Framework** milik Java, di dalam paket `java.util`, adalah sekumpulan antarmuka yang menamai *jenis-jenis* koleksi yang berguna, ditambah sekumpulan kelas yang mengimplementasikan antarmuka itu dengan cara yang berbeda-beda, ditambah dua kelas utilitas yang penuh dengan metode statis pembantu [5].

Gambar 9.2 adalah peta wilayahnya. Bacalah dari atas. Antarmuka `Collection` menjanjikan apa yang dapat dilakukan setiap kelompok elemen: `add`, `remove`, `contains`, `size`, `isEmpty`, `iterator`. Tiga antarmuka memperluasnya: `List` (barisan yang berposisi), `Queue` (elemen-elemen yang menunggu diambil menurut suatu urutan), dan `Set` (tanpa duplikat). `Deque` memperluas `Queue` dengan kerja di kedua ujung, dan `SortedSet` memperluas `Set` dengan urutan. Di bawahnya, dihubungkan oleh panah putus-putus yang berarti *mengimplementasikan*, terdapat kelas-kelas konkret yang Anda buat dengan `new`; panah utuh di antara dua kelas berarti *memperluas*, seperti `LinkedHashSet` memperluas `HashSet`. Di bawah koleksi-koleksi itu, `Map` berdiri sendiri: map bukanlah koleksi elemen melainkan koleksi asosiasi *kunci* → *nilai*, sehingga ia punya antarmuka akarnya sendiri, dengan `SortedMap` di bawahnya. Di kanan bawah ada dua kelas utilitas, `Arrays` dan `Collections`, yang tidak memperluas apa pun selain `Object` dan seluruhnya berisi metode statis seperti `Arrays.asList`, `Collections.sort`, dan `Collections.max`.



Gambar 9.2 Inti Collections Framework milik Java. Antarmuka ditandai «interface»; panah putus-putus berarti *mengimplementasikan* dan panah utuh berarti *memperluas*; kelas-kelas abu-abu adalah peninggalan sejarah. `Map` bukan `Collection`; `Arrays` dan `Collections` memuat metode statis pembantu.

`Vector` dan `Hashtable`, yang digambar abu-abu, adalah peninggalan sejarah, lebih tua daripada framework-nya, dan menyinkronkan setiap operasi, sehingga lebih lambat; kode baru memakai `ArrayList` dan `HashMap` (Bab 14 memperlihatkan perkakas untuk berbagi koleksi antar-thread). `Deque` (*double-ended queue*, antrean berujung dua) diimplementasikan oleh `ArrayDeque` dan `LinkedList`, sehingga `LinkedList` sekaligus sebuah `List` dan sebuah `Deque`. Dua penghalusan

sengaja tidak digambar agar gambar tetap mudah dibaca: `EnumMap`/`EnumSet` untuk kunci berupa enum, dan tiga antarmuka yang disisipkan Java 21 ke dalam hierarki, yaitu `SequencedCollection`, `SequencedSet`, dan `SequencedMap`, yang memberi setiap koleksi berurutan metode `getFirst`, `getLast`, `addFirst`, `addLast`, dan `reversed` [6]. `List`, `Deque`, `SortedSet`, dan `SortedMap` kini memperluasnya, sedangkan `LinkedHashSet` dan `LinkedHashMap` mengimplementasikannya.

9.3.1 Tiga keluarga: set, list, dan map

Hampir setiap pertanyaan tentang koleksi mana yang harus dipakai dimulai dengan memilih di antara tiga keluarga. **Tabel 9.1** menjajarkannya.

Tabel 9.1 Perbandingan tiga keluarga koleksi.

	Set	List	Map
Duplikat	tidak ada: menambah elemen yang sudah ada tidak mengubah apa pun	boleh: elemen yang sama dapat muncul berkali-kali	kunci unik; nilai boleh berulang
Urutan	tidak dipedulikan (tetapi `TreeSet` menjaga urutan terurut dan `LinkedHashSet` urutan penyisipan)	urutan elemen adalah intinya: setiap elemen punya posisi	tidak dipedulikan (tetapi `TreeMap` menjaga kunci terurut dan `LinkedHashMap` urutan penyisipan)
Operasi utama	`add`, `contains`, `remove`	`get(i)`, `set(i, x)`, `add`, `remove`	`get(key)`, `put(key, value)`
Gambaran sehari-hari	himpunan huruf dalam sebuah kata: “one” dan “neo” punya huruf yang sama	sebuah kalimat: “you and us us”	kamus: sebuah kata menuntun ke artinya
Python	`set`, `frozenset`	`list`, `tuple`	`dict`

Set menjawab *apakah elemen ini ada di dalam saya?*; list menyimpan segalanya, termasuk yang berulang, di tempat Anda menaruhnya; map menjawab *apa pasangan kunci ini?*, seperti kamus atau *associative array* (larik asosiatif). Antrean, keluarga keempat, adalah list dengan aturan tertentu untuk mengambil elemen (Subbab 9.7).

9.3.2 Memprogram terhadap antarmuka

Gambar itu membawa akibat praktis bagi cara kita mendeklarasikan variabel. Tulislah

```
List<String> myList = new ArrayList<String>();
```

alih-alih

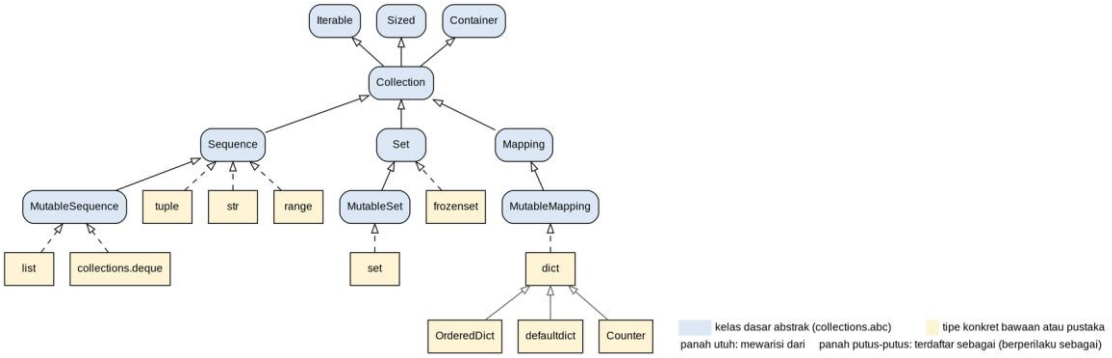
```
ArrayList<String> myList = new ArrayList<String>();
```

Sedapat mungkin antarmukalah yang dipakai sebagai tipe variabel, karena antarmukalah yang diandalkan bagian lain kode. Bila `myList` dideklarasikan sebagai `List`, kode yang memakainya hanya dapat memanggil operasi yang dimiliki setiap list. Kelak, bila pengukuran menunjukkan bahwa `LinkedList` lebih cocok, kita mengganti satu kata di ruas kanan, dan tidak ada bagian lain program yang dapat rusak, karena tidak ada bagian lain yang pernah tahu kelas apa itu. Bab 6 menyebut ini *memprogram terhadap antarmuka*; untuk koleksi, inilah kebiasaan sehari-hari para pemrogram Java. Hal yang sama berlaku untuk parameter: metode yang menerima `List<String>` menerima segala jenis list, sedangkan metode yang menuntut `ArrayList<String>` menolak jenis lain tanpa alasan. (Sejak Java 7, argumen tipe di ruas kanan boleh diserahkan kepada kompilator, `new ArrayList<>()`: bentuk *diamond*. Mulai sekarang kita memakainya.)

9.3.3 Koleksi di Python

Python punya lebih sedikit kelas koleksi, dan yang terpenting dibangun ke dalam bahasanya dengan sintaksis sendiri: `[1, 2]` membuat `list`, `(1, 2)` sebuah `tuple`, `{1, 2}` sebuah `set`, `{"WASH": 7000}` sebuah `dict`. Modul `collections` menambahkan `deque`, `OrderedDict`, `defaultdict`, `Counter`, dan `namedtuple`; modul `heapq` dan `bisect` menambahkan algoritma yang mengubah list biasa menjadi antrean prioritas atau larik terurut [7], [8].

Padanan antarmuka Java di Python adalah modul `collections.abc`, yang kelas-kelas dasar abstraknya menamai jenis-jenis koleksi: `Iterable` (dapat diulang), `Sized` (punya `len`), `Container` (mendukung `in`), dan gabungan ketiganya, `Collection`; di bawahnya `Sequence`, `Set`, dan `Mapping`, masing-masing dengan versi `Mutable`…. **Gambar 9.3** menggambarnya. Tipe-tipe bawaan tidak mewarisi kelas-kelas ini dengan cara biasa; mereka *didaftarkan* sebagai berperilaku seperti kelas itu, sehingga `isinstance([], collections.abc.MutableSequence)` bernilai `True`. Itulah *duck typing* yang diresmikan: kelas yang berperilaku seperti barisan *adalah* barisan.



Gambar 9.3 Tipe-tipe koleksi Python dan kelas dasar abstrak `collections.abc`, padanan antarmuka koleksi Java.

Tabel 9.2 memasangkan kedua bahasa. Simpanlah di dekat Anda sepanjang sisa bab ini.

Tabel 9.2 Koleksi-koleksi bab ini dalam Java dan dalam Python.

Kegunaan	Antarmuka Java	Kelas Java	Python
barisan berposisi	`List<E>`	`ArrayList`, `LinkedList`	`list`; `tuple` bila tidak pernah berubah
tanpa duplikat, uji keanggotaan cepat	`Set<E>`	`HashSet`, `LinkedHashSet`	`set`, `frozenset`
tanpa duplikat, terurut	`SortedSet<E>`, `NavigableSet<E>`	`TreeSet`	`sorted(s)` bila perlu; `bisect` pada list terurut
kunci → nilai	`Map<K,V>`	`HashMap`, `LinkedHashMap`	`dict` (menjaga urutan penyisipan sejak 3.7)
kunci → nilai, kunci terurut	`SortedMap<K,V>`, `NavigableMap<K,V>`	`TreeMap`	`dict(sorted(d.items()))` bila perlu
pertama masuk, pertama keluar	`Queue<E>`	`ArrayDeque`, `LinkedList`	`collections.deque`
terakhir masuk, pertama keluar	`Deque<E>`	`ArrayDeque`	`list` (`append`/`pop`)
terkecil dulu	`Queue<E>`	`PriorityQueue`	`heapq` pada sebuah `list`
menghitung	`Map<K,Integer>` dengan `merge`	`HashMap`, `TreeMap`	`collections.Counter`
kunci yang belum ada mendapat nilai bawaan	`computeIfAbsent`, `getOrDefault`	`Map` apa saja	`collections.defaultdict`, `dict.get`

9.4 List

9.4.1 Operasi sehari-hari

Kita mulai dari keluarga yang sudah kita kenal. **Kode 9.5** menjalankan operasi-operasi `List`, mula-mula pada beberapa kata, lalu pada posisi, dan akhirnya menyerahkan list yang sama kepada satu metode sebagai `ArrayList` dan sebagai `LinkedList`.

Kode 9.5 `ListDemo.java`: menambah, menguji, menghitung, operasi borongan, posisi, dan satu metode yang melayani dua kelas list.

```java
import java.util.ArrayList;
import java.util.Arrays;
import java.util.LinkedList;
import java.util.List;

public class ListDemo {
    public static void main(String[] args) {
        List<String> myList = new ArrayList<String>();   // antarmuka di sebelah kiri
        myList.add("you");
        myList.add("and");
        myList.add("us");
        System.out.println(myList);                        // [you, and, us]
        System.out.println(myList.contains("they"));       // false
        System.out.println(myList.add("us"));              // true: duplikat diizinkan
        System.out.println(myList.size());                 // 4

        String s = "";
        for (String w : myList) {
            s += w + " ";
        }
        System.out.println("Sentence = " + s);

        myList.addAll(Arrays.asList("love", "them"));
        System.out.println(myList);
        myList.removeAll(Arrays.asList("love", "them", "dude"));
        System.out.println(myList);
        myList.retainAll(Arrays.asList("and", "us", "love"));
        System.out.println(myList);

        // Posisi: setiap elemen list punya indeks, mulai dari 0.
        myList.add(0, "we");           // sisipkan di depan
        myList.set(1, "or");           // ganti indeks 1
        System.out.println(myList + " get(2) = " + myList.get(2)
                + " indexOf(\"us\") = " + myList.indexOf("us"));

        // Memprogram terhadap antarmuka: metode yang sama melayani kedua kelas.
        describe(new ArrayList<>(myList));
        describe(new LinkedList<>(myList));
    }

    static void describe(List<String> words) {
        words.remove(words.size() - 1);                   // buang elemen terakhir
        System.out.println(words.getClass().getSimpleName() + " " + words
                + " size " + words.size());
    }
}
```

```
[you, and, us]
false
true
4
Sentence = you and us us
[you, and, us, us, love, them]
[you, and, us, us]
[and, us, us]
[we, or, us, us] get(2) = us indexOf("us") = 2
ArrayList [we, or, us] size 3
LinkedList [we, or, us] size 3
```

Ikutilah keluarannya baris demi baris. Setelah tiga panggilan `add`, list tercetak sebagai `[you, and, us]`: `toString` sebuah list menampilkan elemen-elemennya secara berurutan di antara kurung siku. `contains("they")` bernilai `false`. `add("us")` mengembalikan `true` walaupun `"us"` sudah ada — list mengizinkan duplikat, dan `add` melaporkan bahwa list berubah. Ukurannya kini 4, dan

perulangan membangun kalimat `"you and us us "` (perhatikan spasi di akhir, yang ditambahkan setelah setiap kata termasuk yang terakhir).

Tiga baris berikutnya adalah **operasi borongan** (*bulk operations*), yang menerima satu koleksi utuh sebagai argumen. `addAll(Arrays.asList("love", "them"))` menambahkan kedua kata di ujung; `Arrays.asList` adalah cara cepat membuat list kecil berukuran tetap dari beberapa nilai. `removeAll(Arrays.asList("love", "them", "dude"))` menghapus setiap elemen yang muncul dalam argumen — `"dude"` tidak ada di dalam list, dan itu tidak berbahaya. `retainAll(Arrays.asList("and", "us", "love"))` melakukan kebalikannya: ia hanya menyimpan elemen yang muncul dalam argumen, sehingga `"you"` pergi dan kedua `"us"` tinggal: `[and, us, us]`. Dalam bahasa himpunan, `addAll` adalah gabungan, `removeAll` selisih, dan `retainAll` irisan, tetapi pada list ketiganya menghormati duplikat dan urutan.

Kemudian tibalah posisi. `add(0, "we")` menyisipkan di depan dan mendorong semua elemen lain satu tempat ke kanan; `set(1, "or")` mengganti elemen di indeks 1; `get(2)` membaca indeks 2; `indexOf("us")` menemukan indeks *pertama* yang berisi `"us"`, di sini 2. Posisi dimulai dari 0, seperti pada larik.

Terakhir, `describe(List<String> words)` ditulis terhadap antarmuka. Ia dipanggil sekali dengan `ArrayList` baru dan sekali dengan `LinkedList` baru, masing-masing salinan `myList` (setiap kelas koleksi punya konstruktor yang menyalin koleksi lain). Metode itu membuang elemen terakhir lalu mencetak nama kelas, isi list, dan ukurannya; kedua baris hanya berbeda pada nama kelas. Itulah memprogram terhadap antarmuka dalam praktik.

Kode 9.6 mengerjakan hal yang sama dalam Python.

Kode 9.6 `list_demo.py`: operasi yang sama dengan `list` Python, dan satu fungsi yang melayani `list` maupun `deque`.

```python
"""list_demo.py: operasi list sehari-hari, dan deque sebagai implementasi kedua."""
from collections import deque


def describe(words):
    words.pop()                                     # buang elemen terakhir
    print(type(words).__name__, list(words), "size", len(words))


def main():
    my_list: list[str] = []
    my_list.append("you")
    my_list.append("and")
    my_list.append("us")
    print(my_list)                      # ['you', 'and', 'us']
    print("they" in my_list)            # False
    my_list.append("us")                # append mengembalikan None; duplikat diizinkan
    print(len(my_list))                 # 4

    s = ""
    for w in my_list:
        s += w + " "
    print("Sentence = " + s)
    print("Sentence = " + " ".join(my_list))     # cara Pythonic, tanpa spasi di akhir

    my_list.extend(["love", "them"])
    print(my_list)
    my_list = [w for w in my_list if w not in {"love", "them", "dude"}]   # removeAll
    print(my_list)
    my_list = [w for w in my_list if w in {"and", "us", "love"}]          # retainAll
    print(my_list)

    # Posisi: setiap elemen list punya indeks, mulai dari 0.
    my_list.insert(0, "we")             # sisipkan di depan
    my_list[1] = "or"                   # ganti indeks 1
    print(my_list, "get(2) =", my_list[2], 'indexOf("us") =', my_list.index("us"))

    # Duck typing: fungsi yang sama melayani kedua kelas.
    describe(list(my_list))
    describe(deque(my_list))


if __name__ == "__main__":
    main()
```

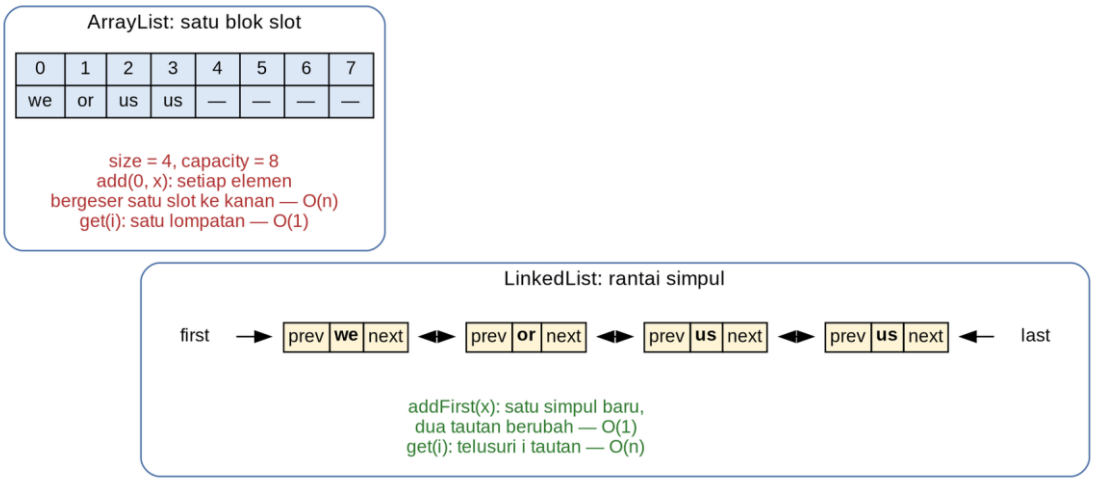
```
['you', 'and', 'us']
False
4
Sentence = you and us us
Sentence = you and us us
['you', 'and', 'us', 'us', 'love', 'them']
['you', 'and', 'us', 'us']
['and', 'us', 'us']
['we', 'or', 'us', 'us'] get(2) = us indexOf("us") = 2
list ['we', 'or', 'us'] size 3
deque ['we', 'or', 'us'] size 3
```

Kesepadanannya dekat. `append`, `insert`, pengindeksan, `index`, `len`, dan `in` mengerjakan tugas `add`, `add(i, x)`, `get`/`set`, `indexOf`, `size`, dan `contains`; `append` mengembalikan `None`, karena list Python tidak pernah menolak elemen. Python tidak punya `removeAll` maupun `retainAll`: cara Pythonic adalah **list comprehension**, `[w for w in my_list if w not in {"love", "them", "dude"}]`, yang membangun list *baru* berisi elemen-elemen yang lolos uji (tulis `my_list[:] = [...]` untuk mengubah objek aslinya di tempat). `" ".join(my_list)` membangun kalimat tanpa spasi di akhir. Petunjuk `my_list: list[str]` ditujukan bagi perkakas seperti mypy; Python

mengabaikannya saat berjalan (Subbab 9.8.7). Dan `describe` melayani `list` maupun `collections.deque` tanpa antarmuka yang dideklarasikan, karena keduanya punya `pop` dan `len`: *duck typing*, cara Python memprogram terhadap antarmuka.

9.4.2 ArrayList versus LinkedList

Kedua kelas mengimplementasikan `List`, sehingga menawarkan operasi yang sama. Cara keduanya menyimpan elemen sama sekali berbeda, dan itulah yang membuat operasi yang sama murah pada yang satu dan mahal pada yang lain. **Gambar 9.4** memperlihatkan list `[we, or, us, us]` dalam kedua bentuk.



Gambar 9.4 List yang sama disimpan dengan dua cara. `ArrayList` menyimpan referensi dalam satu blok slot yang bersebelahan, dengan kapasitas cadangan di ujung; `LinkedList` menyimpan rantai simpul, masing-masing tertaut ke tetangganya.

`ArrayList` adalah larik tumbuh dari Bab 5: satu blok slot yang bersebelahan, dan sebanyak `size` slot pertamanya terpakai. Karena slot-slotnya bersebelahan, alamat slot i dihitung dalam satu langkah, sehingga `get(i)` dan `set(i, x)` berwaktu konstan, $O(1)$. Menambah di ujung juga rata-rata $O(1)$ (pertumbuhan ulang yang sesekali terjadi dibayar oleh banyak penambahan yang murah, seperti diperlihatkan Bab 5). Namun menyisipkan atau menghapus di *depan* atau di tengah memaksa setiap elemen sesudahnya bergeser satu slot. Menyisipkan n elemen satu per satu di depan sebuah `ArrayList` yang semula kosong karena itu menggeser

$$0 + 1 + 2 + \cdots + (n-1) = \frac{n(n-1)}{2} \tag{9.3}$$

elemen seluruhnya, sekitar 5×10^9 pergeseran untuk $n = 100{,}000$.

`LinkedList` adalah rantai objek-objek simpul kecil, masing-masing memegang satu elemen dan dua referensi, `prev` dan `next`, ditambah referensi ke simpul `first` dan `last`. Menyisipkan di ujung mana pun membuat satu simpul dan mengubah dua referensi: $O(1)$, berapa pun panjang list-nya. Menghapus melalui iterator yang sudah berdiri di tempat yang tepat juga $O(1)$. Tetapi tidak ada cara melompat ke posisi i: `get(i)` harus menelusuri rantai dari ujung yang lebih dekat, hingga $n/2$ langkah, sehingga biayanya $O(n)$. Karena itu perulangan `for (int i = 0; i < list.size(); i++) list.get(i)` atas sebuah `LinkedList` berbiaya total $O(n^2)$ — cara terkenal untuk membuat program yang cepat menjadi lambat.

Dalam praktik `ArrayList` hampir selalu menang: slot-slotnya bersebelahan di memori, yang dibaca prosesor dengan sangat cepat, sedangkan setiap simpul `LinkedList` adalah objek terpisah

yang tersebar di memori. Untuk pekerjaan murah di *kedua ujung*, `ArrayDeque` (Subbab 9.7) pun mengalahkan `LinkedList`. `list` Python adalah sebuah `ArrayList`; `deque`-nya menangani kasus kedua ujung.

9.4.3 Mengunjungi setiap elemen, dan menghapus dengan aman

Ada tiga cara mengunjungi elemen-elemen list, dan satu jebakan terkenal. **Kode 9.7** memperlihatkan keempatnya.

Kode 9.7 `IterationDemo.java`: tiga cara mengunjungi list; menghapus di dalam perulangan for-each gagal, sedangkan `Iterator.remove` dan `removeIf` berhasil.

```
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

public class IterationDemo {
    public static void main(String[] args) {
        List<String> words = new ArrayList<>(List.of("you", "us", "us", "and", "us"));

        // 1. Tiga cara mengunjungi setiap elemen.
        for (int i = 0; i < words.size(); i++) {       // lewat indeks
            System.out.print(words.get(i) + " ");
        }
        System.out.println();
        for (String w : words) {                        // for-each (memakai Iterator)
            System.out.print(w + " ");
        }
        System.out.println();
        Iterator<String> it = words.iterator();         // Iterator secara manual
        while (it.hasNext()) {
            System.out.print(it.next() + " ");
        }
        System.out.println();

        // 2. Menghapus di dalam for-each: list berubah di bawah iterator.
        try {
            for (String w : words) {
                if (w.equals("us")) {
                    words.remove(w);
                }
            }
        } catch (java.util.ConcurrentModificationException e) {
            System.out.println("for-each + remove: " + e.getClass().getSimpleName());
        }
        System.out.println("words now: " + words);

        // 3. Menghapus melalui iterator itu sendiri aman.
        words = new ArrayList<>(List.of("you", "us", "us", "and", "us"));
        Iterator<String> it2 = words.iterator();
        while (it2.hasNext()) {
            if (it2.next().equals("us")) {
                it2.remove();
            }
        }
        System.out.println("iterator.remove: " + words);
```

Kode 9.7 (Lanjutan) *IterationDemo.java: tiga cara mengunjungi list; menghapus di dalam perulangan for-each gagal, sedangkan Iterator.remove dan removeIf berhasil.*

```
        // 4. Atau sebutkan apa yang dihapus, dan biarkan list yang berulang.
        words = new ArrayList<>(List.of("you", "us", "us", "and", "us"));
        words.removeIf(w -> w.equals("us"));
        System.out.println("removeIf: " + words);
    }
}
```

```
you us us and us
you us us and us
you us us and us
for-each + remove: ConcurrentModificationException
words now: [you, us, and, us]
iterator.remove: [you, and]
removeIf: [you, and]
```

Tiga baris pertama mencetak kata-kata yang sama dalam urutan yang sama. Perulangan berindeks memakai `get(i)`, yang baik untuk `ArrayList` tetapi lambat untuk `LinkedList`. **Perulangan for-each**, `for (String w : words)`, berlaku untuk setiap koleksi, karena setiap koleksi bersifat `Iterable`: ia punya metode `iterator()` yang mengembalikan sebuah **`Iterator`**, objek yang menelusuri koleksi satu elemen setiap kali. Perulangan ketiga mengerjakan secara manual apa yang dikerjakan for-each di balik layar: `hasNext()` menanyakan apakah masih ada elemen, dan `next()` mengembalikannya lalu maju. Iterator hanya bergerak maju — tidak ada `prev()` — dan ia membaca koleksi; ia tidak menyalinnya.

Blok kedua adalah jebakannya. Di dalam perulangan for-each kita memanggil `words.remove(w)` untuk setiap `"us"`. `"us"` yang pertama ditemukan dan dihapus. Pada putaran berikutnya, iterator tersembunyi menyadari bahwa list telah diubah di belakang punggungnya — setiap `ArrayList` menghitung perubahan strukturnya, dan iterator membandingkan hitungan itu dengan yang diingatnya — lalu melempar `ConcurrentModificationException`. List tertinggal setengah bersih, `[you, us, and, us]`. Namanya menyesatkan: tidak ada thread yang terlibat; "concurrent" berarti list diubah *selagi* sebuah iterasi berlangsung. Perilaku ini disebut **fail-fast** (gagal cepat), dan itu sebuah kebaikan: galat muncul seketika, dekat dengan penyebabnya.

Ada dua cara yang benar. Iterator sendiri punya metode `remove()` yang menghapus elemen yang terakhir dikembalikan `next()` sambil menjaga catatan iterator itu tetap mutakhir, sehingga blok ketiga menghapus ketiga `"us"` dengan aman. Yang paling sederhana, sejak Java 8 setiap koleksi punya `removeIf`, yang menerima sebuah *syarat* dan mengerjakan perulangannya di dalam. Syarat `w -> w.equals("us")` adalah **ekspresi lambda**: fungsi kecil yang ditulis di tempat, di sini "diberi `w`, jawablah apakah `w` sama dengan `"us"`". Bab 6 memakai satu lambda bersama `assertThrows`, dan Bab 10 menjelaskan lambda sepenuhnya; untuk sementara, bacalah panah itu sebagai "sedemikian sehingga".

Python punya jebakan yang sama, dalam bentuk yang lebih licik, seperti diperlihatkan **Kode 9.8**.

Kode 9.8 `iteration_demo.py`: penelusuran yang sama dalam Python. Menghapus dari list di dalam perulangan `for` diam-diam melewatkan elemen; mengubah ukuran kamus di dalam perulangan memunculkan `RuntimeError`.

```
"""iteration_demo.py: mengunjungi list, dan menghapus darinya dengan aman."""


def main():
    words = ["you", "us", "us", "and", "us"]

    # 1. Tiga cara mengunjungi setiap elemen.
    for i in range(len(words)):                 # lewat indeks
        print(words[i], end=" ")
    print()
    for w in words:                             # for (memakai iterator)
        print(w, end=" ")
    print()
    it = iter(words)                            # iterator secara manual
    while True:
        try:
            print(next(it), end=" ")
        except StopIteration:
            break
    print()

    # 2. Menghapus di dalam for: tanpa eksepsi, tetapi ada elemen yang terlewat!
    for w in words:
        if w == "us":
            words.remove(w)
    print("for + remove: no error")
    print("words now:", words)

    # 3. Berulang pada salinan, ubah yang asli.
    words = ["you", "us", "us", "and", "us"]
    for w in words[:]:
        if w == "us":
            words.remove(w)
    print("copy + remove:", words)

    # 4. Atau bangun list yang diinginkan dalam satu ekspresi.
    words = ["you", "us", "us", "and", "us"]
    words = [w for w in words if w != "us"]
    print("comprehension:", words)
```

Kode 9.8 (Lanjutan) *iteration_demo.py: penelusuran yang sama dalam Python. Menghapus dari list di dalam perulangan for diam-diam melewatkan elemen; mengubah ukuran kamus di dalam perulangan memunculkan RuntimeError.*

```
    # Berbeda dengan list, dict menyadari perubahan itu.
    stock = {"soap": 3, "bleach": 0}
    try:
        for item in stock:
            if stock[item] == 0:
                del stock[item]
    except RuntimeError as e:
        print("RuntimeError:", e)


if __name__ == "__main__":
    main()
```

```
you us us and us
you us us and us
you us us and us
for + remove: no error
words now: ['you', 'and', 'us']
copy + remove: ['you', 'and']
comprehension: ['you', 'and']
RuntimeError: dictionary changed size during iteration
```

Perulangan manual memakai fungsi bawaan `iter` dan `next`; ketika iterator habis, `next` memunculkan `StopIteration`, dan dengan cara itulah setiap perulangan `for` Python tahu kapan harus berhenti. Blok kedua menghapus dari list sambil berulang atasnya, dan Python **tidak** memunculkan galat — tetapi hasilnya, `['you', 'and', 'us']`, masih memuat sebuah `"us"`. Mari kita telusuri. List dimulai sebagai `[you, us, us, and, us]`. Iterator tersembunyi berdiri di indeks 1 (`"us"`), dan `remove` menghapusnya; list menutup celahnya, sehingga `"us"` kedua pindah ke indeks 1, dan iterator, yang maju ke indeks 2, tidak pernah melihatnya. Di indeks 3 perulangan bertemu `"us"` terakhir dan memanggil `words.remove("us")` — tetapi `remove` menghapus `"us"` *pertama* di dalam list, yaitu yang terlewat tadi, di indeks 1. Perulangan lalu melewati ujung list, dan `"us"` terakhir selamat. Bug yang memberi jawaban salah yang *masuk akal* adalah jenis yang paling buruk. Obatnya adalah berulang atas salinan, `words[:]`, sambil mengubah yang asli, atau — lebih baik — membangun list yang diinginkan dengan *comprehension*. Kamus dan himpunan bersifat fail-fast seperti koleksi Java: mengubah ukurannya selama perulangan memunculkan `RuntimeError: dictionary changed size during iteration`, seperti diperlihatkan blok terakhir.

> ✓ **WAWASAN KUNCI.** Jangan pernah mengubah *struktur* sebuah koleksi (menambah atau menghapus elemen) selagi sebuah perulangan menelusurinya, kecuali melalui penelusur itu sendiri (`Iterator.remove`). Kumpulkan perubahannya lalu terapkan sesudahnya, atau bangun koleksi baru. Mengganti elemen di tempat (`set(i, x)`, `words[i] = x`) tidak berbahaya, karena strukturnya tidak berubah.

9.5 Set (Himpunan)

9.5.1 Himpunan bilangan, dan himpunan nama

Set (himpunan) adalah koleksi tanpa duplikat. Pertanyaan utamanya adalah keanggotaan — *apakah elemen ini ada di dalam saya?* — dan ia menjawab pertanyaan itu jauh lebih cepat daripada list. **Kode 9.9** dimulai dengan `TreeSet` berisi bilangan bulat beserta aljabar himpunannya, lalu memasukkan enam kedatangan yang sama ke dalam tiga jenis himpunan.

Kode 9.9 `SetDemo.java`: `TreeSet` bilangan dengan gabungan, selisih, dan irisan; nama-nama yang sama dalam `HashSet`, `LinkedHashSet`, dan `TreeSet`; kueri elemen terdekat; dan himpunan huruf.

```
import java.util.Arrays;
import java.util.HashSet;
import java.util.LinkedHashSet;
import java.util.List;
import java.util.Set;
import java.util.TreeSet;

public class SetDemo {
    public static void main(String[] args) {
        Set<Integer> myNumbers = new TreeSet<Integer>();
        myNumbers.add(1);
        myNumbers.add(2);
        myNumbers.add(3);
        System.out.println(myNumbers);                    // [1, 2, 3]
        System.out.println(myNumbers.contains(7));        // false
        System.out.println(myNumbers.add(3));             // false: sudah ada
        System.out.println(myNumbers.size());             // 3

        int sum = 0;
        for (int n : myNumbers) {
            sum += n;
        }
        System.out.println("Sum = " + sum);               // Sum = 6

        myNumbers.addAll(Arrays.asList(1, 2, 3, 4, 5, 6, 7));
        System.out.println(myNumbers);                    // gabungan
        myNumbers.removeAll(Arrays.asList(4, 5, 6, 7, 8, 9, 10));
        System.out.println(myNumbers);                    // selisih
        myNumbers.retainAll(Arrays.asList(2, 3, 4, 5));
        System.out.println(myNumbers);                    // irisan
        System.out.println();

        // Kata-kata yang sama dalam tiga jenis himpunan.
        List<String> arrivals = List.of("Sari", "Budi", "Rina", "Agus", "Budi", "Sari");
        System.out.println("HashSet       " + new HashSet<>(arrivals));
        System.out.println("LinkedHashSet " + new LinkedHashSet<>(arrivals));
        System.out.println("TreeSet       " + new TreeSet<>(arrivals));

        // TreeSet tahu urutannya, sehingga dapat menjawab pertanyaan "terdekat".
        TreeSet<String> names = new TreeSet<>(arrivals);
        System.out.println("first " + names.first() + ", last " + names.last()
                + ", ceiling(\"C\") " + names.ceiling("C")
                + ", headSet(\"R\") " + names.headSet("R"));
```

Kode 9.9 (Lanjutan) *SetDemo.java: TreeSet bilangan dengan gabungan, selisih, dan irisan; nama-nama yang sama dalam HashSet, LinkedHashSet, dan TreeSet; kueri elemen terdekat; dan himpunan huruf.*

```
        // Urutan tidak penting bagi himpunan: huruf "one" dan "neo" adalah himpunan yang
sama.
        System.out.println("letters(one).equals(letters(neo)) = "
                + letters("one").equals(letters("neo")));
    }

    static Set<Character> letters(String word) {
        Set<Character> result = new TreeSet<>();
        for (char c : word.toCharArray()) {
            result.add(c);
        }
        return result;
    }
}
```

```
[1, 2, 3]
false
false
3
Sum = 6
[1, 2, 3, 4, 5, 6, 7]
[1, 2, 3]
[2, 3]

HashSet       [Sari, Budi, Agus, Rina]
LinkedHashSet [Sari, Budi, Rina, Agus]
TreeSet       [Agus, Budi, Rina, Sari]
first Agus, last Sari, ceiling("C") Rina, headSet("R") [Agus, Budi]
letters(one).equals(letters(neo)) = true
```

Blok pertama mencerminkan contoh list, dan perbedaannyalah yang penting. `add(3)` pada himpunan yang sudah memuat 3 mengembalikan `false`: himpunan memuat setiap elemen paling banyak satu kali. Operasi borongan kini menjadi aljabar himpunan sejati: `addAll` membentuk **gabungan** $\{1,2,3\} \cup \{1,\ldots,7\} = \{1,\ldots,7\}$, `removeAll` membentuk **selisih** $\{1,\ldots,7\} \setminus \{4,\ldots,10\} = \{1,2,3\}$, dan `retainAll` membentuk **irisan** $\{1,2,3\} \cap \{2,3,4,5\} = \{2,3\}$. `TreeSet` selalu tercetak dalam urutan menaik.

Blok kedua adalah empat baris paling mendidik di subbab ini. Enam kedatangan, dengan Budi dan Sari datang dua kali, menjadi empat nama berbeda di setiap himpunan — tetapi dalam tiga urutan yang berlainan:

- `HashSet` mencetak `[Sari, Budi, Agus, Rina]`, urutan yang tampak acak. Urutan itu tidak acak; itu urutan *ember hash*, yang dijelaskan di subsubbab berikut. Namun urutan itu tidak boleh diandalkan: ia dapat berubah ketika himpunannya tumbuh atau antarversi Java.
- `LinkedHashSet` mencetak `[Sari, Budi, Rina, Agus]`, urutan kedatangan pertama. Ia adalah himpunan hash yang juga merangkai elemen-elemennya dalam list tertaut menurut urutan penyisipan, dengan biaya dua referensi tambahan per elemen.
- `TreeSet` mencetak `[Agus, Budi, Rina, Sari]`, terurut. Ia menyimpan elemen-elemennya dalam pohon pencarian seimbang, yang juga memungkinkannya menjawab pertanyaan yang tidak dapat dijawab himpunan hash: `first()` dan `last()`, `ceiling("C")` (elemen terkecil yang $\geq$ `"C"`: tidak ada nama di himpunan itu yang berawalan C sampai Q, sehingga jawabannya `Rina`), dan `headSet("R")` (semua elemen yang $<$ `"R"`).

Baris terakhir memperlihatkan mengapa himpunan cocok untuk pertanyaan “huruf apa saja yang dipakai kata ini?”. Kata “one” dan “neo” memuat huruf yang sama dalam urutan berbeda, sehingga himpunan hurufnya sama. (Himpunan tidak cukup untuk mendeteksi *anagram*, kata yang memakai huruf yang sama dengan jumlah yang sama: “aab” dan “abb” punya himpunan huruf yang sama tetapi bukan anagram. Soal Latihan 9.4 membangun pencari anagram yang benar dengan sebuah map.)

Kode 9.10 adalah versi Python-nya.

Kode 9.10 `set_demo.py`: `set` Python dengan operator-operatornya, urutan-urutan yang diberikan tiga himpunan Java, pencarian biner pada list terurut, dan sebuah `frozenset`.

```
"""set_demo.py: set Python, dan cara memperoleh urutan yang diberikan tiga set Java."""
from bisect import bisect_left


def main():
    my_numbers: set[int] = set()
    my_numbers.add(1)
    my_numbers.add(2)
    my_numbers.add(3)
    print(my_numbers)                           # {1, 2, 3}
    print(7 in my_numbers)                      # False
    before = len(my_numbers)
    my_numbers.add(3)                           # add mengembalikan None ...
    print(len(my_numbers) > before)             # ... maka tanyakan apakah ukurannya
bertambah
    print(len(my_numbers))                      # 3

    total = 0
    for n in my_numbers:
        total += n
    print("Sum =", total)                       # atau cukup sum(my_numbers)

    my_numbers |= {1, 2, 3, 4, 5, 6, 7}         # gabungan
    print(my_numbers)
    my_numbers -= {4, 5, 6, 7, 8, 9, 10}        # selisih
    print(my_numbers)
    my_numbers &= {2, 3, 4, 5}                  # irisan
    print(my_numbers)
    print({1, 2, 3} ^ {2, 3, 4}, "(symmetric difference)")
    print()

    arrivals = ["Sari", "Budi", "Rina", "Agus", "Budi", "Sari"]
    distinct = set(arrivals)                    # urutan iterasi dapat berubah dari eksekusi
ke eksekusi
    print("set size   ", len(distinct))
    print("first-seen ", list(dict.fromkeys(arrivals)))   # seperti LinkedHashSet
    print("sorted     ", sorted(distinct))                 # seperti TreeSet

    # Pertanyaan "terdekat" pada list terurut, dengan pencarian biner.
    names = sorted(distinct)
    i = bisect_left(names, "C")
    print(f"first {names[0]}, last {names[-1]}, ceiling(\"C\") {names[i]}, "
          f"headSet(\"R\") {names[:bisect_left(names, 'R')]}")

    print("set('one') == set('neo') =", set("one") == set("neo"))
    frozen = frozenset({"WASH", "IRON"})        # himpunan imutabel: dapat menjadi kunci
dict
    print("frozenset is hashable:", hash(frozen) == hash(frozenset({"IRON", "WASH"})))
```

Kode 9.10 (Lanjutan) *set_demo.py: set Python dengan operator-operatornya, urutan-urutan yang diberikan tiga himpunan Java, pencarian biner pada list terurut, dan sebuah frozenset.*

```
if __name__ == "__main__":
    main()
```

```
{1, 2, 3}
False
False
3
Sum = 6
{1, 2, 3, 4, 5, 6, 7}
{1, 2, 3}
{2, 3}
{1, 4} (symmetric difference)

set size    4
first-seen  ['Sari', 'Budi', 'Rina', 'Agus']
sorted      ['Agus', 'Budi', 'Rina', 'Sari']
first Agus, last Sari, ceiling("C") Rina, headSet("R") ['Agus', 'Budi']
set('one') == set('neo') = True
frozenset is hashable: True
```

Python menulis aljabar himpunan dengan operator: `|` untuk gabungan, `-` untuk selisih, `&` untuk irisan, dan `^` untuk selisih simetris (elemen yang tepat berada di salah satu dari dua himpunan), masing-masing dengan bentuk di-tempat seperti `|=`. `add` mengembalikan `None`, sehingga listing membandingkan ukuran sebelum dan sesudah untuk mengetahui apakah elemennya baru.

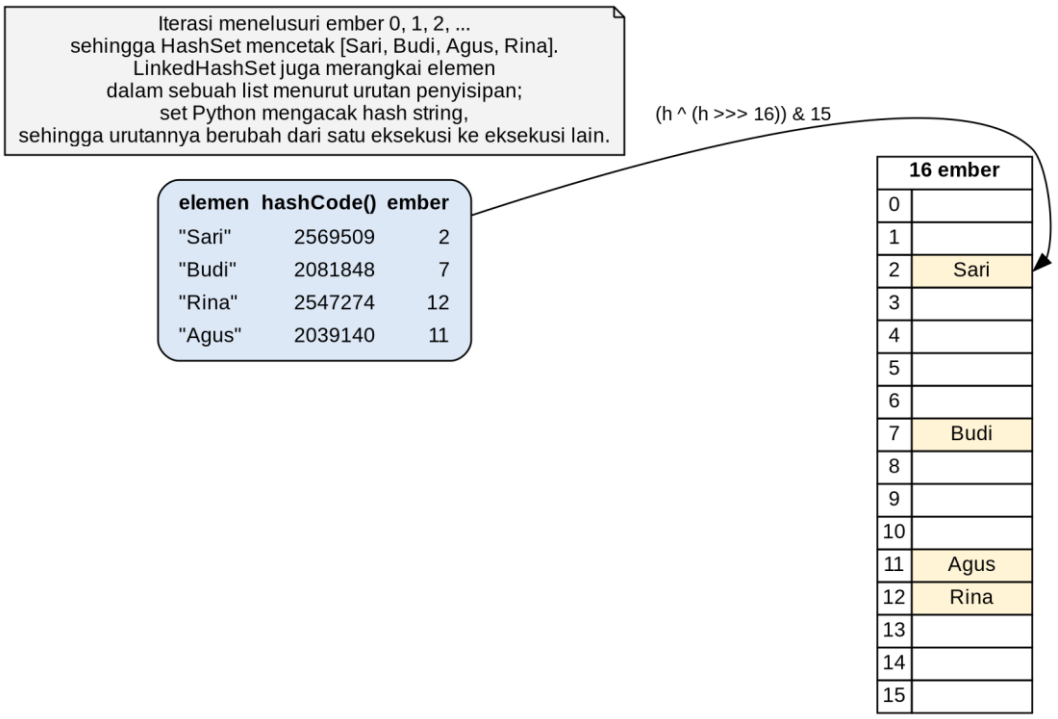
Perhatikan apa yang *tidak* dicetak listing: himpunan nama itu sendiri. `set` Python berisi string beriterasi dalam urutan yang berubah *dari satu eksekusi ke eksekusi berikutnya*, karena Python "menggarami" hash string dengan nilai acak saat mulai berjalan, sebagai pertahanan terhadap data yang sengaja dirancang agar bertabrakan. Bilangan bulat tidak digarami, sehingga `{1, 2, 3}` tercetak secara dapat diramalkan. Bila urutan penting, mintalah secara eksplisit: `sorted(s)` memberi urutan `TreeSet` Java, dan `list(dict.fromkeys(items))` memberi urutan kemunculan pertama ala `LinkedHashSet` (`dict` mengingat urutan penyisipan, dan `fromkeys` membuat kamus yang kuncinya adalah item-item itu). Untuk pertanyaan elemen terdekat, simpanlah list terurut dan pakailah modul `bisect`: `bisect_left(names, "C")` adalah indeks tempat `"C"` akan disisipkan agar list tetap terurut, yang ditemukan dengan pencarian biner, sehingga `names[i]` adalah *ceiling*-nya. Terakhir, `frozenset` adalah himpunan imutabel; karena tidak dapat diubah, hash-nya stabil dan ia dapat menjadi elemen himpunan lain atau kunci kamus.

9.5.2 Bagaimana HashSet menemukan elemen: hashing

Mengapa `contains` pada `HashSet` cepat? Karena himpunan itu tidak mencari. Ia *menghitung* di mana elemen itu seharusnya berada.

Setiap objek Java punya metode `hashCode()` yang mengembalikan `int`, dan Bab 6 serta Bab 7 menegaskan bahwa objek-objek yang sama harus mengembalikan kode hash yang sama. Untuk `String` s sepanjang k dengan karakter $s_0, \ldots, s_{k-1}$ kodenya adalah $s_0 \cdot 31^{k-1} + s_1 \cdot 31^{k-2} + \cdots + s_{k-1}$, dihitung dengan limpahan melingkar 32-bit. Untuk `"Sari"` nilainya $83 \cdot 31^3 + 97 \cdot 31^2 + 114 \cdot 31 + 105 = 2{,}472{,}653 + 93{,}217 + 3{,}534 + 105 = 2{,}569{,}509$.

`HashSet` (yang sebenarnya berisi `HashMap` di dalamnya, dengan elemen-elemennya sebagai kunci) menyimpan larik berisi m **ember** (*bucket*); himpunan baru dimulai dengan $m = 16$. Untuk menempatkan elemen berkode hash h, ia mencampurkan bit-bit tinggi ke bit-bit rendah lalu mengambil empat bit terendah, `(h ^ (h >>> 16)) & 15`, sebuah bilangan dari 0 sampai 15. Untuk `"Sari"` hasilnya ember 2. **Gambar 9.5** memperlihatkan keempat nama.



Gambar 9.5 `HashSet` berisi empat nama. Kode hash setiap nama memilih salah satu dari 16 ember, sehingga `contains` hanya melihat satu ember; iterasi menelusuri ember menurut urutan indeks, yang menjelaskan urutan tercetak `[Sari, Budi, Agus, Rina]`.

`contains("Rina")` menghitung embernya, 12, lalu membandingkan `"Rina"` hanya dengan isi ember 12, memakai `equals`. Ia tidak pernah melihat nama-nama lain. Iterasi menelusuri ember dari 0 sampai 15, dan persis itulah sebabnya **Kode 9.9** mencetak `[Sari, Budi, Agus, Rina]`: ember 2, 7, 11, 12.

Dua elemen berbeda dapat jatuh ke ember yang sama, sebuah **tabrakan** (*collision*); ember itu lalu memuat rantai pendek berisi beberapa elemen, dan `contains` membandingkan dengan masing-masing. Seberapa panjang rantainya? Bila n elemen tersebar ke dalam m ember, panjang rata-rata rantai sama dengan **faktor muatan** (*load factor*)

$$\alpha = \frac{n}{m} \tag{9.4}$$

dan, dengan anggapan fungsi hash menyebarkan elemen secara merata, pencarian yang gagal rata-rata memeriksa α elemen setelah embernya dihitung, dan pencarian yang berhasil sekitar $1 + \alpha/2$ [9]. Java menjaga α tetap kecil dengan paksa: ketika n melampaui $0.75\,m$, tabel menggandakan banyaknya ember dan menyebar ulang setiap elemen. Jadi $\alpha \le 0.75$ selalu berlaku, dan biaya harapan `add`, `contains`, dan `remove` adalah

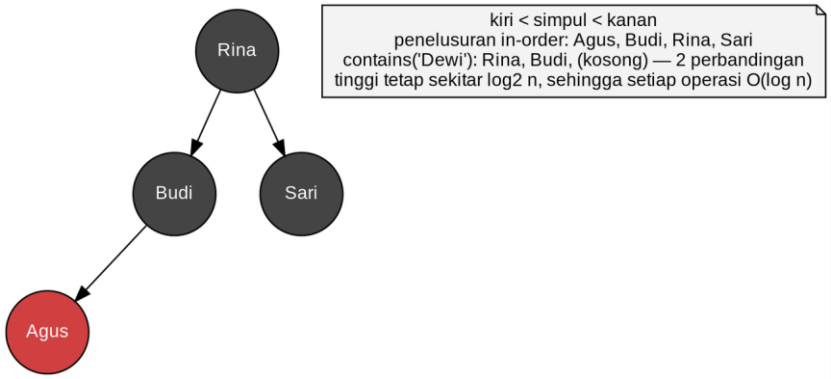
$$T_{\text{hash}}(n) = O(1 + \alpha) = O(1) \tag{9.5}$$

— konstan, berapa pun banyaknya elemen yang dimuat himpunan. Penggandaan itu berbiaya $O(n)$ saat terjadi, tetapi, seperti larik tumbuh di Bab 5, begitu jarang sehingga biaya rata-rata per penyisipan tetap konstan. (Sejak Java 8, ember yang memuat lebih dari delapan elemen bertabrakan diubah menjadi pohon kecil, yang membatasi kasus terburuk menjadi $O(\log n)$. `set` dan `dict` Python menyelesaikan tabrakan dengan *open addressing*, mencoba slot lain pada larik yang sama, dengan biaya harapan $O(1)$ yang sama.)

Semua ini bertumpu pada konsistensi `hashCode` dan `equals` — objek yang sama, kode hash yang sama — dan pada kode hash yang *tidak berubah* selama objek berada di dalam himpunan. Soal Latihan 9.8 memperlihatkan apa yang terjadi bila kode itu berubah.

9.5.3 Bagaimana TreeSet menjaga urutan

`TreeSet` menyimpan elemen-elemennya dalam sebuah **pohon pencarian biner**: setiap simpul punya paling banyak dua anak, setiap elemen di subpohon kiri sebuah simpul lebih kecil daripada elemen simpul itu, dan setiap elemen di subpohon kanannya lebih besar. **Gambar 9.6** memperlihatkan keempat nama.



Gambar 9.6 `TreeSet` berisi empat nama yang sama: pohon pencarian merah-hitam. Penelusuran in-order mengunjungi nama-nama menurut urutan; pencarian mengikuti satu lintasan dari akar.

Untuk mencari `"Dewi"`, bandingkan dengan akar `Rina`: Dewi datang sebelum Rina, jadi pergi ke kiri, ke `Budi`; Dewi datang sesudah Budi, jadi pergi ke kanan — di sana tidak ada apa-apa, sehingga `"Dewi"` tidak ada. Dua perbandingan. Menelusuri pohon *in-order* (subpohon kiri, simpul, subpohon kanan, penelusuran pada Soal 8.7) mengunjungi elemen-elemen secara terurut, dan begitulah `TreeSet` mencetak serta beriterasi.

Pohon pencarian hanya cepat bila *seimbang*: pohon yang dibangun dengan menyisipkan kunci-kunci yang sudah terurut ke dalam pohon pencarian biasa merosot menjadi rantai, dan mencarinya berbiaya $O(n)$, seperti diamati pada Penyelesaian 7.2. `TreeSet` dan `TreeMap` Java memakai **pohon merah-hitam** (*red-black tree*), yang simpul-simpulnya diwarnai sedemikian rupa sehingga tidak ada lintasan dari akar ke daun yang lebih dari dua kali panjang lintasan lainnya [10]. Ini menjamin bahwa pohon berisi n elemen punya tinggi

$$h \leq 2\log_2(n+1) \tag{9.6}$$

sehingga `add`, `contains`, dan `remove` mengunjungi $O(\log n)$ simpul [9]. Untuk sejuta elemen itu paling banyak sekitar 40 perbandingan, dibandingkan sekitar 500,000 untuk sebuah list. Himpunan pohon 10 sampai 20 kali lebih lambat daripada himpunan hash, dan sebagai gantinya ia menjaga urutan serta menjawab pertanyaan rentang (`first`, `ceiling`, `headSet`, `subSet`). Elemennya harus dapat dibandingkan: harus mengimplementasikan `Comparable`, atau himpunannya harus diberi sebuah `Comparator` saat dibuat (Subbab 9.9). Pustaka baku Python tidak punya himpunan berbasis pohon; `sorted()` dan `bisect` pada list mencukupi sebagian besar kebutuhan, dan paket pihak ketiga *sortedcontainers* mencukupi sisanya.

9.6 Map

9.6.1 Map dari kunci ke nilai

Map mengasosiasikan setiap *kunci* dengan satu *nilai*. Kunci bersifat unik, seperti kata-kata dalam kamus; nilai boleh berulang. Map adalah koleksi di balik setiap pertanyaan "cari saja": nomor pesanan → pesanan, kode layanan → harga, pelanggan → list pesanan, kata → banyaknya kemunculan.

Kode 9.11 dimulai dengan gaya yang dipakai banyak kode Java lama: *tipe mentah* (*raw types*), tanpa argumen tipe, sehingga setiap nilai keluar sebagai `Object` dan harus di-cast. Lalu ia mengerjakan tugas yang sama dengan generik dan berkenalan dengan metode-metode `Map` yang paling berguna.

Kode 9.11 `MapDemo.java`: menelusuri entri sebuah map, mula-mula dengan tipe mentah dan cast, lalu dengan generik; metode-metode utama `Map` pada daftar harga; menghitung dengan `merge`; dan tiga jenis map.

```
import java.util.HashMap;
import java.util.Iterator;
import java.util.LinkedHashMap;
import java.util.Map;
import java.util.Set;
import java.util.TreeMap;

public class MapDemo {
    @SuppressWarnings({"rawtypes", "unchecked"})
    static void oldStyle() {
        Map<Object, String> map = new HashMap<>();
        map.put(1, "you");
        map.put(2, "and");
        map.put(3, "us");

        Set info = map.entrySet();                    // tipe mentah: tanpa <...>
        Iterator iterator = info.iterator();
        while (iterator.hasNext()) {
            Map.Entry m = (Map.Entry) iterator.next(); // setiap langkah perlu cast
            int key = (Integer) m.getKey();
            String value = (String) m.getValue();
            System.out.println("key: " + key + " -> value: " + value);
        }
        System.out.println();
        map.remove(2);
        info = map.entrySet();
        iterator = info.iterator();
        while (iterator.hasNext()) {
            Map.Entry m = (Map.Entry) iterator.next();
            int key = (Integer) m.getKey();
            String value = (String) m.getValue();
            System.out.println("key: " + key + " -> value: " + value);
        }
        System.out.println();
    }

    public static void main(String[] args) {
        oldStyle();
```

Kode 9.11 (Lanjutan) *MapDemo.java: menelusuri entri sebuah map, mula-mula dengan tipe mentah dan cast, lalu dengan generik; metode-metode utama Map pada daftar harga; menghitung dengan merge; dan tiga jenis map.*

```
        // Penelusuran yang sama dengan generik: tanpa cast sama sekali.
        Map<Integer, String> map = new HashMap<>();
        map.put(1, "you");
        map.put(2, "and");
        map.put(3, "us");
        map.remove(2);
        for (Map.Entry<Integer, String> e : map.entrySet()) {
            System.out.println("key: " + e.getKey() + " -> value: " + e.getValue());
        }

        // Daftar harga: kode layanan -> rupiah per kilogram.
        Map<String, Integer> rates = new HashMap<>();
        System.out.println("put WASH: " + rates.put("WASH", 7000));       // null: kunci
baru
        rates.put("IRON", 5000);
        rates.put("WASH_IRON", 10000);
        System.out.println("put WASH: " + rates.put("WASH", 7500));       // 7000: nilai
lama
        System.out.println("get IRON = " + rates.get("IRON")
                + ", get SHOES = " + rates.get("SHOES")
                + ", getOrDefault SHOES = " + rates.getOrDefault("SHOES", 0));
        System.out.println("containsKey(\"IRON\") = " + rates.containsKey("IRON")
                + ", containsValue(5000) = " + rates.containsValue(5000));
        System.out.println("keys " + new TreeMap<>(rates).keySet()
                + ", values " + new TreeMap<>(rates).values());

        // Menghitung dengan merge: tambah 1, atau mulai dari 1 bila kuncinya baru.
        String[] tickets = {"WASH", "IRON", "WASH", "WASH_IRON", "WASH", "IRON"};
        Map<String, Integer> counts = new LinkedHashMap<>();
        for (String t : tickets) {
            counts.merge(t, 1, Integer::sum);
        }
        System.out.println("LinkedHashMap " + counts);
        System.out.println("TreeMap       " + new TreeMap<>(counts));
        map.forEach((k, v) -> System.out.println("forEach " + k + " = " + v));
    }
}
```

```
key: 1 -> value: you
key: 2 -> value: and
key: 3 -> value: us

key: 1 -> value: you
key: 3 -> value: us

key: 1 -> value: you
key: 3 -> value: us
put WASH: null
put WASH: 7000
get IRON = 5000, get SHOES = null, getOrDefault SHOES = 0
containsKey("IRON") = true, containsValue(5000) = true
keys [IRON, WASH, WASH_IRON], values [5000, 7500, 10000]
LinkedHashMap {WASH=3, IRON=2, WASH_IRON=1}
TreeMap       {IRON=2, WASH=3, WASH_IRON=1}
forEach 1 = you
forEach 3 = us
```

Map bukanlah koleksi, sehingga `oldStyle` menelusurinya melalui `entrySet()`, sebuah *view* (tampilan) map sebagai `Set` berisi objek `Map.Entry`, yang masing-masing memegang satu kunci dan satu nilai. Dengan tipe mentah setiap `next()` mengembalikan `Object`, sehingga kode harus melakukan cast ke `Map.Entry`, lalu kuncinya ke `Integer` dan nilainya ke `String` — tiga janji yang

tidak dapat diperiksa kompilator. `@SuppressWarnings({"rawtypes", "unchecked"})` membungkam peringatan untuk kode ini saja. Setelah `map.remove(2)` hanya kunci 1 dan 3 yang tersisa. (Kunci `Integer` yang kecil kebetulan keluar dari `HashMap` dalam urutan menaik, karena kode hash bilangan bulat kecil adalah bilangan itu sendiri; untuk kunci terurut, tetaplah meminta `TreeMap`.)

Versi generik di `main` sama sekali tidak memerlukan cast: `Map<Integer, String>` memberi tahu kompilator tipe-tipenya, `Map.Entry<Integer, String> e` menerima setiap entri, dan `e.getKey()` sudah berupa `Integer`. Subbab 9.8 menjelaskan caranya.

Daftar harga memperlihatkan metode-metode sehari-hari:

- `put(key, value)` menyimpan sebuah nilai dan **mengembalikan nilai sebelumnya** untuk kunci itu, atau `null` bila belum ada. `put("WASH", 7000)` yang pertama mencetak `null`; yang kedua, `put("WASH", 7500)`, menggantinya dan mengembalikan nilai lama 7000. Map tidak pernah memegang dua nilai untuk satu kunci.
- `get(key)` mengembalikan nilainya, atau `null` bila kunci tidak ada; `getOrDefault(key, d)` mengembalikan `d` alih-alih `null`.
- `containsKey` cepat (satu pencarian hash); `containsValue` harus melihat setiap nilai, $O(n)$.
- `keySet()` dan `values()` adalah view atas kunci dan nilai. Membungkus map dalam `TreeMap` membuat keduanya keluar terurut.

Menghitung adalah pemakaian map yang paling umum, dan `merge` mengerjakannya dalam satu baris: `counts.merge(t, 1, Integer::sum)` berarti "bila `t` belum ada, simpan 1; bila sudah, simpan nilai lama ditambah 1". `Integer::sum` adalah *referensi metode* (*method reference*), cara singkat menyerahkan metode yang sudah ada, `Integer.sum(a, b)`, sebagai aturan untuk memadukan nilai lama dan baru. `LinkedHashMap` mencetak hitungan menurut urutan kemunculan pertama kode-kodenya, `TreeMap` menurut urutan abjad kuncinya. Terakhir, `map.forEach((k, v) -> ...)` mengunjungi setiap entri dengan lambda berparameter dua.

Kode 9.12 memperlihatkan `dict` Python.

Kode 9.12 `map_demo.py`: `dict` Python, `get`-nya dengan nilai bawaan, `KeyError`, dan tiga cara menghitung: `get`, `Counter`, dan `defaultdict`.

```
"""map_demo.py: dict Python, padanan Map di Java."""
from collections import Counter, defaultdict


def main():
    d = {}
    d[1] = "you"
    d[2] = "and"
    d[3] = "us"
    for key, value in d.items():
        print(f"key: {key} -> value: {value}")
    print()
    del d[2]
    for key, value in d.items():
        print(f"key: {key} -> value: {value}")
    print()

    # Daftar harga: kode layanan -> rupiah per kilogram.
    rates: dict[str, int] = {}
    print("put WASH:", rates.get("WASH"))            # None: kunci baru
    rates["WASH"] = 7000
    rates["IRON"] = 5000
    rates["WASH_IRON"] = 10000
    old = rates.get("WASH")
    rates["WASH"] = 7500
    print("put WASH:", old)                          # 7000: nilai lama
    print(f"get IRON = {rates['IRON']}, get SHOES = {rates.get('SHOES')}, "
          f"getOrDefault SHOES = {rates.get('SHOES', 0)}")
    print(f'containsKey("IRON") = {"IRON" in rates}, '
          f"containsValue(5000) = {5000 in rates.values()}")
    print(f"keys {sorted(rates)}, values {[rates[k] for k in sorted(rates)]}")
    try:
        rates["SHOES"]
    except KeyError as e:
        print("rates['SHOES'] -> KeyError:", e)
```

Kode 9.12 (Lanjutan) *map_demo.py: dict Python, get-nya dengan nilai bawaan, KeyError, dan tiga cara menghitung: get, Counter, dan defaultdict.*

```
    # Menghitung: tiga cara.
    tickets = ["WASH", "IRON", "WASH", "WASH_IRON", "WASH", "IRON"]
    counts: dict[str, int] = {}
    for t in tickets:
        counts[t] = counts.get(t, 0) + 1
    print("dict        ", counts)                    # urutan penyisipan, seperti
LinkedHashMap
    print("sorted      ", dict(sorted(counts.items())))  # seperti TreeMap
    print("Counter     ", Counter(tickets).most_common(2))
    by_first_letter = defaultdict(list)              # kunci yang belum ada dimulai sebagai
[]
    for t in dict.fromkeys(tickets):
        by_first_letter[t[0]].append(t)
    print("defaultdict ", dict(by_first_letter))


if __name__ == "__main__":
    main()
```

```
key: 1 -> value: you
key: 2 -> value: and
key: 3 -> value: us

key: 1 -> value: you
key: 3 -> value: us

put WASH: None
put WASH: 7000
get IRON = 5000, get SHOES = None, getOrDefault SHOES = 0
containsKey("IRON") = True, containsValue(5000) = True
keys ['IRON', 'WASH', 'WASH_IRON'], values [5000, 7500, 10000]
rates['SHOES'] -> KeyError: 'SHOES'
dict         {'WASH': 3, 'IRON': 2, 'WASH_IRON': 1}
sorted       {'IRON': 2, 'WASH': 3, 'WASH_IRON': 1}
Counter      [('WASH', 3), ('IRON', 2)]
defaultdict  {'W': ['WASH', 'WASH_IRON'], 'I': ['IRON']}
```

`dict` ditulis dengan kurung kurawal dan titik dua, dan diindeks dengan kurung siku. `d.items()` memberi pasangan kunci–nilai, yang dibongkar perulangan `for` menjadi `key, value`, tanpa cast dan tanpa objek entri. Dua perilaku berbeda dari Java dan penting setiap hari:

- `d[key]` pada kunci yang tidak ada memunculkan **`KeyError`** alih-alih mengembalikan `None`. Pakailah `d.get(key)`, yang mengembalikan `None`, atau `d.get(key, default)`. Penugasan `d[key] = value` tidak mengembalikan apa pun, sehingga listing membaca nilai lama dengan `get` sebelum menggantinya.
- `dict` mengingat **urutan penyisipan** (dijamin sejak Python 3.7), sehingga secara bawaan ia berperilaku seperti `LinkedHashMap`. Untuk urutan terurut, `dict(sorted(counts.items()))` membangun kamus baru dari pasangan-pasangan yang diurutkan menurut kunci.

Untuk menghitung, `counts[t] = counts.get(t, 0) + 1` adalah terjemahan langsung `merge`. Kelas `collections.Counter` mengerjakan semuanya sekaligus dan menambahkan `most_common(k)`. Dan `collections.defaultdict(list)` adalah kamus yang membuat nilai yang belum ada dengan memanggil `list()` saat sebuah kunci pertama kali dipakai, sehingga `by_first_letter[t[0]].append(t)` tidak memerlukan uji “apakah kuncinya sudah ada?”. Itulah padanan Python bagi `computeIfAbsent(key, k -> new ArrayList<>())` milik Java, yang dipakai buku pesanan binatu di Subbab 9.11.

9.6.2 Membaca berkas ke dalam koleksi

Koleksi menjadi berguna begitu data datang dari luar. **Kode 9.13** membaca sebuah berkas teks kecil dua kali — sekali kata demi kata dengan `Scanner`, sekali baris demi baris dengan `BufferedReader` — menghitung kata-katanya dengan `TreeMap`, dan melaporkan tiga kata yang paling sering muncul. Berkas `words.txt` terletak di samping programnya:

```
Wash the towels and fold the towels
Iron the shirts
Wash the shirts and the sheets
Fold the sheets
```

Kode 9.13 `WordFile.java`: sebuah berkas dibaca ke dalam `List` kata dan `List` baris, kata-katanya dihitung dalam `TreeMap`, dan yang tersering ditemukan dengan mengurutkan entri-entrinya.

```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;
import java.util.Scanner;
import java.util.TreeMap;

public class WordFile {
    public static void main(String[] args) {
        String fileName = args.length > 0 ? args[0] : "words.txt";

        // Cara ke-1: Scanner memberi kita satu kata setiap kali.
        List<String> words = new ArrayList<>();
        try (Scanner fileScan = new Scanner(new File(fileName))) {
            while (fileScan.hasNext()) {
                words.add(fileScan.next().toLowerCase());
            }
        } catch (IOException e) {
            System.err.println(e);
            return;
        }
        System.out.println(words.size() + " words: " + words);

        // Cara ke-2: BufferedReader memberi kita satu baris setiap kali.
        List<String> stringList = new ArrayList<>();
        try (BufferedReader br = new BufferedReader(new FileReader(fileName))) {
            String currentLine;
            while ((currentLine = br.readLine()) != null) {
                stringList.add(currentLine);
            }
        } catch (IOException e) {
            System.err.println(e);
            return;
        }
        System.out.println("Now we want to print the list out:");
        for (int i = 0; i < stringList.size(); i++) {
            System.out.println((i + 1) + ": " + stringList.get(i));
        }
```

Kode 9.13 (Lanjutan) *WordFile.java: sebuah berkas dibaca ke dalam List kata dan List baris, kata-katanya dihitung dalam TreeMap, dan yang tersering ditemukan dengan mengurutkan entri-entrinya.*

```
        // Hitung setiap kata; TreeMap menyimpan kata-kata terurut abjad.
        Map<String, Integer> counts = new TreeMap<>();
        for (String w : words) {
            counts.put(w, counts.getOrDefault(w, 0) + 1);
        }
        System.out.println(counts.size() + " distinct: " + counts);

        // Tiga kata paling sering: terbanyak dulu, yang seri menurut abjad.
        List<Map.Entry<String, Integer>> entries = new ArrayList<>(counts.entrySet());
        entries.sort((a, b) -> {
            int byCount = Integer.compare(b.getValue(), a.getValue());   // yang lebih
besar dulu
            return byCount != 0 ? byCount : a.getKey().compareTo(b.getKey());
        });
        System.out.println("top 3: " + entries.subList(0, 3));
    }
}
```

```
19 words: [wash, the, towels, and, fold, the, towels, iron, the, shirts, wash, the, shirts,
and, the, sheets, fold, the, sheets]
Now we want to print the list out:
1: Wash the towels and fold the towels
2: Iron the shirts
3: Wash the shirts and the sheets
4: Fold the sheets
8 distinct: {and=2, fold=2, iron=1, sheets=2, shirts=2, the=6, towels=2, wash=2}
top 3: [the=6, and=2, fold=2]
```

Kedua pembaca berasal dari Bab 5 dan dibuka dengan `try`-with-resources. `next()` milik `Scanner` mengembalikan satu kata setiap kali, diubah ke huruf kecil agar "Wash" dan "wash" terhitung satu; `readLine()` milik `BufferedReader` mengembalikan satu baris setiap kali dan `null` di akhir berkas. (Kode yang menutup `Scanner` di dalam `finally` dengan `fileScan.close()` punya lubang: bila berkasnya tidak ada, `fileScan` masih `null`, dan blok `finally` melempar `NullPointerException` yang menyembunyikan galat yang sebenarnya.) `java WordFile nothere.txt` mencetak `java.io.FileNotFoundException: nothere.txt (No such file or directory)`.

Baris penghitung `counts.put(w, counts.getOrDefault(w, 0) + 1)` adalah bentuk panjang `merge`. `TreeMap` lalu mencetak kedelapan kata berbeda menurut abjad. Untuk menemukan kata yang paling sering, kita menyalin entri-entrinya ke dalam list dan mengurutkannya dengan comparator yang ditulis sebagai lambda: bandingkan hitungannya secara *terbalik* (`Integer.compare(b.getValue(), a.getValue())`, yang lebih besar dulu), dan, hanya bila hitungannya sama, kata-katanya menurut abjad. `subList(0, 3)` adalah view atas tiga yang pertama. Hasil seri di antara keenam kata yang muncul dua kali diputus menurut abjad, sehingga `and` dan `fold` yang menang.

Kode 9.14 lebih pendek, seperti biasanya penanganan berkas di Python.

Kode 9.14 `word_file.py`: laporan yang sama dalam Python, dengan kunci pengurutan yang mengurutkan menurut hitungan menurun dan kata menaik, serta `Counter.most_common`.

```python
"""word_file.py: membaca berkas ke dalam list, menghitung katanya, melaporkan yang
tersering."""
import sys
from collections import Counter


def main():
    file_name = sys.argv[1] if len(sys.argv) > 1 else "words.txt"
    try:
        with open(file_name, encoding="utf-8") as f:
            text = f.read()
    except OSError as e:
        print(e, file=sys.stderr)
        return

    words = [w.lower() for w in text.split()]        # split() memotong di setiap spasi
putih
    print(len(words), "words:", words)

    string_list = text.splitlines()                  # satu string per baris
    print("Now we want to print the list out:")
    for i, line in enumerate(string_list, start=1):
        print(f"{i}: {line}")

    counts: dict[str, int] = {}
    for w in words:
        counts[w] = counts.get(w, 0) + 1
    print(len(counts), "distinct:", dict(sorted(counts.items())))

    # Tiga kata paling sering: terbanyak dulu, yang seri menurut abjad.
    ranked = sorted(counts.items(), key=lambda kv: (-kv[1], kv[0]))
    print("top 3:", ranked[:3])
    print("Counter:", Counter(words).most_common(3))

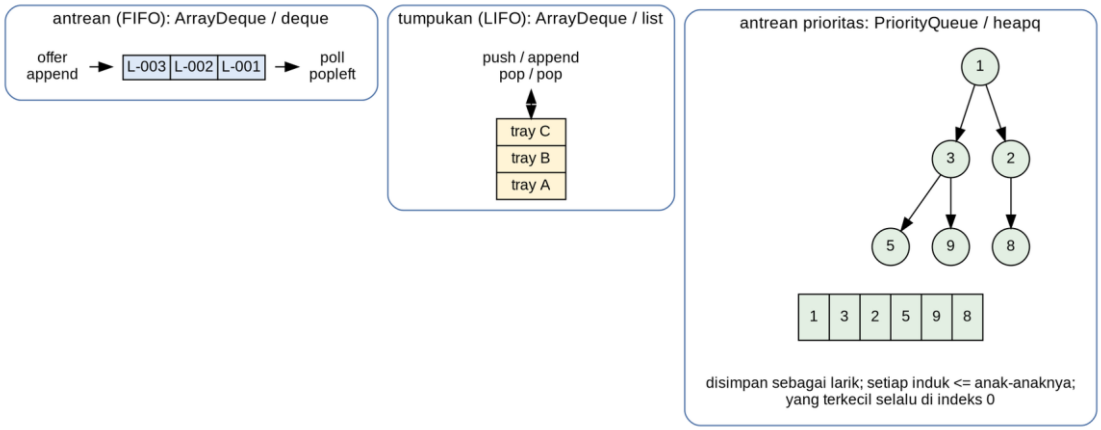

if __name__ == "__main__":
    main()
```

```
19 words: ['wash', 'the', 'towels', 'and', 'fold', 'the', 'towels', 'iron', 'the',
'shirts', 'wash', 'the', 'shirts', 'and', 'the', 'sheets', 'fold', 'the', 'sheets']
Now we want to print the list out:
1: Wash the towels and fold the towels
2: Iron the shirts
3: Wash the shirts and the sheets
4: Fold the sheets
8 distinct: {'and': 2, 'fold': 2, 'iron': 1, 'sheets': 2, 'shirts': 2, 'the': 6, 'towels':
2, 'wash': 2}
top 3: [('the', 6), ('and', 2), ('fold', 2)]
Counter: [('the', 6), ('wash', 2), ('towels', 2)]
```

`text.split()` tanpa argumen memotong di setiap deret spasi putih, termasuk pergantian baris, dan `text.splitlines()` memberi baris-baris tanpa karakter pergantian barisnya. Kunci pengurutan `lambda kv: (-kv[1], kv[0])` mengembalikan sebuah *tuple* untuk setiap pasangan: Python membandingkan tuple elemen demi elemen, sehingga pasangan-pasangan diurutkan menurut hitungan yang dinegatifkan (hitungan besar dulu) lalu menurut katanya. Menegatifkan bilangan adalah idiom Python untuk "bagian ini menurun". Baris terakhir memperlihatkan bahwa `Counter.most_common(3)` juga menemukan `the` lebih dulu, tetapi memutus hasil seri di antara kata-kata berhitungan dua menurut *kemunculan pertama* — `wash`, lalu `towels` — bukan menurut abjad. Kedua jawaban benar; keduanya menjawab pertanyaan yang berbeda, dan laporan yang menjanjikan "yang seri menurut abjad" harus mengurutkan secara eksplisit.

9.7 Antrean, Tumpukan, Deque, dan Antrean Prioritas

Sebagian koleksi lebih ditentukan oleh *elemen mana yang keluar berikutnya* daripada oleh cara menyimpan elemennya. **Gambar 9.7** memperlihatkan tiga aturannya.



Gambar 9.7 Tiga aturan untuk mengambil elemen. Antrean melepaskan elemen tertua (FIFO), tumpukan elemen terbaru (LIFO), dan antrean prioritas elemen terkecil, yang disimpan *binary heap* di indeks 0 lariknya.

- **Antrean** (*queue*) bersifat pertama masuk, pertama keluar (FIFO), seperti antrean di loket binatu: `offer` menambah di belakang, `poll` mengambil dari depan, `peek` melihat yang di depan tanpa mengambilnya.
- **Tumpukan** (*stack*) bersifat terakhir masuk, pertama keluar (LIFO), seperti setumpuk nampan: `push` menaruh di atas, `pop` mengambil dari atas.
- **Antrean prioritas** (*priority queue*) melepaskan elemen *terkecil* lebih dulu, apa pun urutan kedatangannya — terkecil menurut urutan alami atau menurut `Comparator` yang Anda berikan.

Antarmuka `Deque` milik Java (diucapkan "dek", *double-ended queue*) mengizinkan penyisipan dan penghapusan di kedua ujung, sehingga satu kelas, `ArrayDeque`, dapat berperan sebagai antrean maupun tumpukan. Ia adalah larik melingkar: depan dan belakangnya bergerak berkeliling di dalam satu blok slot yang tetap, sehingga kedua ujung berbiaya $O(1)$ tanpa menggeser apa pun. **Kode 9.15** memakai ketiga aturan itu.

Kode 9.15 `QueueDemo.java`: `ArrayDeque` sebagai antrean FIFO, sebagai tumpukan LIFO, dan sebagai deque; `PriorityQueue` dengan urutan alami dan dengan `Comparator.reverseOrder()`.

```
import java.util.ArrayDeque;
import java.util.Comparator;
import java.util.Deque;
import java.util.PriorityQueue;
import java.util.Queue;

public class QueueDemo {
    public static void main(String[] args) {
        // Antrean: pertama masuk, pertama keluar (FIFO).
        Queue<String> washQueue = new ArrayDeque<>();
        washQueue.offer("L-001");
        washQueue.offer("L-002");
        washQueue.offer("L-003");
        System.out.println("queue " + washQueue + ", peek " + washQueue.peek());
        System.out.println("poll " + washQueue.poll() + ", poll " + washQueue.poll()
                + ", left " + washQueue);

        // Tumpukan: terakhir masuk, pertama keluar (LIFO). Deque = antrean berujung dua.
        Deque<String> trays = new ArrayDeque<>();
        trays.push("tray A");
        trays.push("tray B");
        trays.push("tray C");
        System.out.println("stack " + trays + ", pop " + trays.pop() + ", pop " +
trays.pop());

        // Kedua ujung dari satu deque.
        Deque<Integer> d = new ArrayDeque<>();
        d.addLast(2);
        d.addLast(3);
        d.addFirst(1);
        System.out.println("deque " + d + ", pollFirst " + d.pollFirst()
                + ", pollLast " + d.pollLast());

        // Antrean prioritas: elemen terkecil keluar lebih dulu.
        Queue<Integer> pq = new PriorityQueue<>();
        for (int x : new int[] {5, 1, 8, 3, 9, 2}) {
            pq.offer(x);
        }
        System.out.println("heap array " + pq);          // TIDAK terurut: sebuah heap
        StringBuilder out = new StringBuilder();
        while (!pq.isEmpty()) {
            out.append(pq.poll()).append(' ');
        }
        System.out.println("poll order " + out.toString().trim());
```

Kode 9.15 (Lanjutan) *QueueDemo.java: ArrayDeque sebagai antrean FIFO, sebagai tumpukan LIFO, dan sebagai deque; PriorityQueue dengan urutan alami dan dengan Comparator.reverseOrder().*

```
        // Terbesar dulu, dengan Comparator.
        Queue<Integer> maxFirst = new PriorityQueue<>(Comparator.reverseOrder());
        for (int x : new int[] {5, 1, 8, 3, 9, 2}) {
            maxFirst.offer(x);
        }
        System.out.println("max first: " + maxFirst.poll() + " " + maxFirst.poll()
                + " " + maxFirst.poll());
        System.out.println("empty poll -> " + new ArrayDeque<String>().poll());
    }
}
```

```
queue [L-001, L-002, L-003], peek L-001
poll L-001, poll L-002, left [L-003]
stack [tray C, tray B, tray A], pop tray C, pop tray B
deque [1, 2, 3], pollFirst 1, pollLast 3
heap array [1, 3, 2, 5, 9, 8]
poll order 1 2 3 5 8 9
max first: 9 8 5
empty poll -> null
```

Antrean melepaskan `L-001` lebih dulu, lalu `L-002`: urutan kedatangan. Tumpukan tercetak `[tray C, tray B, tray A]` — `ArrayDeque` yang dipakai sebagai tumpukan tercetak dari atas ke bawah — dan mengeluarkan `tray C` lebih dulu. Deque menerima `addFirst` dan `addLast` serta memberi `pollFirst` dan `pollLast`. Setiap operasi `Queue` hadir dalam dua rasa: `offer`, `poll`, dan `peek` mengembalikan `false` atau `null` bila tidak dapat mengerjakan tugasnya, seperti diperlihatkan baris terakhir, sedangkan `add`, `remove`, dan `element` melempar eksepsi.

Antrean prioritas layak dilihat pelan-pelan. Setelah enam `offer`, ia tercetak `[1, 3, 2, 5, 9, 8]`: *tidak* terurut. `PriorityQueue` adalah sebuah **binary heap** (tumpukan biner), larik yang dibaca sebagai pohon biner lengkap dengan anak-anak indeks i berada di indeks $2i + 1$ dan $2i + 2$, dan setiap induk tidak lebih besar daripada anak-anaknya (**Gambar 9.7**, kanan). Aturan itu menaruh elemen terkecil di indeks 0 tetapi tidak berkata apa-apa tentang sisanya. `poll` mengambil indeks 0, memindahkan elemen terakhir ke puncak dan membiarkannya tenggelam ke tempatnya, yang memulihkan aturan itu dengan paling banyak satu langkah per tingkat. Heap berisi n elemen punya $\lfloor \log_2 n \rfloor + 1$ tingkat, sehingga `offer` dan `poll` berbiaya $O(\log n)$ dan `peek` berbiaya $O(1)$ [9]. Mengambil sampai kosong menghasilkan `1 2 3 5 8 9`: terurut, satu elemen setiap kali. Untuk mengambil yang *terbesar* lebih dulu, berikan `Comparator.reverseOrder()` kepada konstruktornya.

Kode 9.16 memperlihatkan perkakas Python: `collections.deque` untuk antrean dan deque, `list` biasa untuk tumpukan, dan modul `heapq`, yang menerapkan aturan heap pada list biasa.

Kode 9.16 `queue_demo.py`: `deque` sebagai antrean dan deque, `list` sebagai tumpukan, dan `heapq` sebagai antrean prioritas.

```
"""queue_demo.py: deque sebagai antrean dan tumpukan, heapq sebagai antrean prioritas."""
import heapq
from collections import deque


def main():
    # Antrean: pertama masuk, pertama keluar (FIFO).
    wash_queue = deque()
    wash_queue.append("L-001")
    wash_queue.append("L-002")
    wash_queue.append("L-003")
    print(f"queue {list(wash_queue)}, peek {wash_queue[0]}")
    first, second = wash_queue.popleft(), wash_queue.popleft()
    print(f"poll {first}, poll {second}, left {list(wash_queue)}")

    # Tumpukan: terakhir masuk, pertama keluar (LIFO). List biasa adalah tumpukan yang
lazim.
    trays = []
    trays.append("tray A")
    trays.append("tray B")
    trays.append("tray C")
    # f-string dibangun dari kiri ke kanan, sehingga list ditampilkan sebelum pop.
    print(f"stack {trays[::-1]}, pop {trays.pop()}, pop {trays.pop()}")

    # Kedua ujung dari satu deque.
    d = deque()
    d.append(2)
    d.append(3)
    d.appendleft(1)
    print(f"deque {list(d)}, pollFirst {d.popleft()}, pollLast {d.pop()}")

    # Antrean prioritas: elemen terkecil keluar lebih dulu.
    pq = []
    for x in [5, 1, 8, 3, 9, 2]:
        heapq.heappush(pq, x)
    print("heap array", pq)                              # TIDAK terurut: sebuah heap
    print("poll order", " ".join(str(heapq.heappop(pq)) for _ in range(len(pq))))
```

Kode 9.16 (Lanjutan) *queue_demo.py: deque sebagai antrean dan deque, list sebagai tumpukan, dan heapq sebagai antrean prioritas.*

```
    # Terbesar dulu: masukkan negatifnya.
    max_first = []
    for x in [5, 1, 8, 3, 9, 2]:
        heapq.heappush(max_first, -x)
    print("max first:", -heapq.heappop(max_first), -heapq.heappop(max_first),
          -heapq.heappop(max_first))
    try:
        deque().popleft()
    except IndexError as e:
        print("empty popleft -> IndexError:", e)


if __name__ == "__main__":
    main()
```

```
queue ['L-001', 'L-002', 'L-003'], peek L-001
poll L-001, poll L-002, left ['L-003']
stack ['tray C', 'tray B', 'tray A'], pop tray C, pop tray B
deque [1, 2, 3], pollFirst 1, pollLast 3
heap array [1, 3, 2, 5, 9, 8]
poll order 1 2 3 5 8 9
max first: 9 8 5
empty popleft -> IndexError: pop from an empty deque
```

`deque.popleft()` adalah penghapusan $O(1)$ dari depan; operasi yang sama pada `list`, `pop(0)`, menggeser setiap elemen yang tersisa dan berbiaya $O(n)$. Itulah kesalahan kinerja paling umum dalam program Python yang mengolah antrean. Larik heap-nya identik dengan milik Java, `[1, 3, 2, 5, 9, 8]`, karena kedua pustaka mengimplementasikan algoritma buku teks yang sama. `heapq` tidak punya pilihan terbalik; idiom untuk "terbesar dulu" adalah memasukkan bilangan yang dinegatifkan lalu menegatifkannya lagi saat keluar. `deque` kosong memunculkan `IndexError` alih-alih mengembalikan `None`. (Pencetakan tumpukan memakai f-string, yang bagian-bagiannya dievaluasi dari kiri ke kanan, dan menampilkan list secara terbalik agar puncaknya di depan seperti di Java. Dengan `print(trays, trays.pop(), ...)`, semua argumen dievaluasi *sebelum* pencetakan, dan list akan ditampilkan sesudah kedua pop.)

9.8 Generik dari Nol

Kita sudah menulis `List<String>`, `Map<Integer, String>`, dan `PriorityQueue<Integer>` puluhan kali. Sudah saatnya memahami kurung sudut itu.

> **DEFINISI. Kelas generik** (atau antarmuka generik) adalah kelas yang ditulis dengan satu atau lebih **parameter tipe**, nama pengganti untuk tipe, yang ditulis dalam kurung sudut sesudah nama kelas: `class GList<T>`. Pemakaian kelas itu menyediakan **argumen tipe** untuk setiap parameter, `GList<Integer>`, dan kompilator memeriksa setiap pemakaian seolah-olah `T` telah diganti dengan `Integer` di mana-mana. **Metode generik** mendeklarasikan parameter tipenya sendiri sebelum tipe kembaliannya: `static <T> T lastOf(List<T> list)`.

Dengan kata lain, generik memungkinkan kita menulis kelas yang tidak peduli jenis objek apa yang diolahnya, asalkan semuanya sejenis, dan membuat kompilator menahan setiap pemakai kelas itu pada jenis yang telah mereka pilih.

9.8.1 Kehidupan sebelum generik

Generik hadir di Java 5, pada tahun 2004 [11]. Sebelumnya, sebuah `List` memuat `Object`, dan **Kode 9.17** memperlihatkan apa artinya.

Kode 9.17 `RawTypes.java`: `List` mentah menerima sebuah `Integer` di antara nama-nama dan baru gagal ketika di-cast; `List<String>` generik tidak memerlukan cast dan akan menolak `Integer` itu saat kompilasi.

```
import java.util.ArrayList;
import java.util.List;

public class RawTypes {
    @SuppressWarnings({"rawtypes", "unchecked"})
    static void beforeGenerics() {
        List names = new ArrayList();      // List mentah: berisi Object
        names.add("Siti");
        names.add("Budi");
        names.add(42);                     // diterima tanpa protes
        for (int i = 0; i < names.size(); i++) {
            try {
                String name = (String) names.get(i);   // cast adalah satu-satunya
pemeriksaan kita
                System.out.println(name.toUpperCase());
            } catch (ClassCastException e) {
                System.out.println("index " + i + ": ClassCastException");
            }
        }
    }

    static void withGenerics() {
        List<String> names = new ArrayList<>();
        names.add("Siti");
        names.add("Budi");
        // names.add(42);                  // galat kompilasi: int tidak dapat dikonversi ke
String
        for (String name : names) {        // tidak perlu cast
            System.out.println(name.toUpperCase());
        }
    }

    public static void main(String[] args) {
        System.out.println("-- before generics --");
        beforeGenerics();
        System.out.println("-- with generics --");
        withGenerics();
    }
}
```

```
-- before generics --
SITI
BUDI
index 2: ClassCastException
-- with generics --
SITI
BUDI
```

Di `beforeGenerics`, list mentah menerima `"Siti"`, `"Budi"`, dan bilangan `42` tanpa sepatah kata pun, karena ketiganya adalah `Object`. Kerusakannya baru tampak ketika elemen-elemen dikeluarkan: setiap `get(i)` mengembalikan `Object`, cast `(String)` adalah klaim pemrogram bahwa isinya benar-benar string, dan untuk indeks 2 klaim itu salah, sehingga mesin virtual melempar `ClassCastException`. Dalam program sungguhan, `add(42)` dan cast yang gagal bisa berada di berkas yang berbeda, ditulis dengan selang berbulan-bulan, dan jejak tumpukan (*stack trace*) menunjuk ke cast, bukan ke kesalahannya.

Di `withGenerics`, deklarasi `List<String>` membuat kompilator yang memeriksa. Variabel for-each dapat langsung berupa `String`, tanpa cast, dan baris komentar `names.add(42)`, bila tanda `//`-nya Anda hapus, ditolak sebelum program sempat berjalan:

```
RawTypes.java:25: error: incompatible types: int cannot be converted to String
        names.add(42);
                  ^
Note: Some messages have been simplified; recompile with -Xdiags:verbose to get full output
1 error
```

Itulah seluruh tujuan generik: memindahkan satu keluarga galat dari saat eksekusi, tempat pengguna menjumpainya, ke saat kompilasi, tempat pemrogram menjumpainya.

List Python sejak dulu memuat apa saja. **Kode 9.18** memperlihatkan bahwa kesalahan yang sama tetap berjalan, dan bagaimana petunjuk tipe memungkinkan mypy menemukannya.

Kode 9.18 `raw_types.py`: list Python menerima apa saja; petunjuk `list[str]` memungkinkan mypy menolak `append(42)` sebelum program berjalan.

```python
"""raw_types.py: list Python menerima apa saja; petunjuk tipe membuat mypy dapat
memeriksanya."""


def without_hints():
    names = []                          # objek apa pun boleh masuk
    names.append("Siti")
    names.append("Budi")
    names.append(42)                    # diterima tanpa protes
    for i, name in enumerate(names):
        try:
            print(name.upper())
        except AttributeError:
            print(f"index {i}: AttributeError")


def with_hints() -> None:
    names: list[str] = []               # janji untuk mypy: hanya str yang masuk
    names.append("Siti")
    names.append("Budi")
    names.append(42)                    # mypy melaporkan baris ini; Python menjalankannya
    for name in names:
        print(name.upper() if isinstance(name, str) else f"{name!r} slipped in")


def main():
    print("-- without hints --")
    without_hints()
    print("-- with hints --")
    with_hints()


if __name__ == "__main__":
    main()
```

```
-- without hints --
SITI
BUDI
index 2: AttributeError
-- with hints --
SITI
BUDI
42 slipped in
```

Kedua fungsi berjalan, dan di keduanya bilangan bulat itu menyelinap masuk. mypy tidak berkata apa-apa tentang `without_hints`, yang list-nya tidak punya tipe elemen yang dideklarasikan, dan satu hal yang tepat tentang `with_hints`:

```
raw_types.py:20: error: Argument 1 to "append" of "list" has incompatible type "int";
expected "str"  [arg-type]
Found 1 error in 1 file (checked 1 source file)
```

Generik Java ditegakkan oleh kompilator yang harus dilalui setiap program; petunjuk tipe Python oleh perkakas terpisah yang Anda pilih untuk dijalankan.

9.8.2 Kelas generik: GList

Bab 8 membangun list imutabel berisi *bilangan bulat* dari `nil` dan `cons`. List berisi string akan memerlukan kelas kedua, identik kecuali kata `int`, dan list berisi pesanan kelas ketiga. **Kode 9.19** menulis list itu sekali saja, untuk sembarang tipe elemen `T`.

Kode 9.19 `GListDemo.java`: `GList<T>`, list imutabel generik dengan pembantu rekursif untuk `toString`, dipakai dengan `Integer` dan dengan `String`.

```
import java.util.ArrayList;
import java.util.List;

public class GListDemo {
    public static void main(String[] args) {
        GList<Integer> myGList = new GList<>();
        System.out.printf("myGList: %s%n", myGList);
        int head = 10;
        GList<Integer> myGList2 = new GList<>(head, myGList);
        System.out.printf("myGList2: %s%n", myGList2);
        head = 50;
        GList<Integer> myGList3 = new GList<>(head, myGList2);
        System.out.printf("myGList3: %s%n", myGList3);

        // Kelas yang sama, argumen tipe yang berbeda.
        GList<String> towels = new GList<>("white", new GList<>("blue", GList.nil()));
        String first = towels.head();                  // tanpa cast: head() mengembalikan
String
        System.out.println("towels: " + towels + ", head " + first
                + ", tail " + towels.tail() + ", size " + towels.size());
        try {
            GList.<String>nil().tail();
        } catch (IllegalStateException e) {
            System.out.println("IllegalStateException: " + e.getMessage());
        }
    }
}

// List imutabel umum (generik): T mewakili tipe kelas apa pun.
class GList<T> {
    private final boolean empty;
    private final T head;
    private final GList<T> tail;

    public GList() {                                   // list kosong
        empty = true;
        head = null;
        tail = null;
    }

    public GList(T head, GList<T> tail) {              // list tidak kosong
        this.empty = false;
        this.head = head;
        this.tail = tail;
    }
```

Kode 9.19 (Lanjutan) *GListDemo.java: GList<T>, list imutabel generik dengan pembantu rekursif untuk toString, dipakai dengan Integer dan dengan String.*

```
    public static <T> GList<T> nil() {                // metode generik
        return new GList<>();
    }

    public boolean empty() {
        return empty;
    }

    public T head() {
        if (empty) {
            throw new IllegalStateException("Trying to access head of an empty list");
        }
        return head;
    }

    public GList<T> tail() {
        if (empty) {
            throw new IllegalStateException("Trying to access tail of an empty list");
        }
        return tail;
    }

    public int size() {                               // rekursif: 1 + ukuran ekor
        return empty ? 0 : 1 + tail.size();
    }

    @Override
    public String toString() {
        return toStringHelper(this, new ArrayList<T>());
    }

    // Pembantu rekursif dengan parameter tambahan, yaitu akumulator.
    private String toStringHelper(GList<T> list, List<T> accumulator) {
        if (list == null) {
            throw new IllegalStateException("next element of list was null. "
                    + "Should either be another list or nil. List till now: " +
accumulator);
        } else if (list.empty()) {
            return accumulator.toString();
        } else {
            accumulator.add(list.head());
            return toStringHelper(list.tail(), accumulator);
        }
    }
}
```

```
myGList: []
myGList2: [10]
myGList3: [50, 10]
towels: [white, blue], head white, tail [blue], size 2
IllegalStateException: Trying to access tail of an empty list
```

Di dalam `class GList<T>`, parameter tipe `T` dipakai di mana pun sebuah tipe diharapkan: `head` bertipe `T`, `tail` bertipe `GList<T>`, `head()` mengembalikan `T`. Menurut kebiasaan, parameter tipe berupa satu huruf kapital — `T` untuk *type*, `E` untuk *element*, `K` dan `V` untuk *key* dan *value* — agar menonjol di antara nama-nama kelas.

Di `main`, `GList<Integer>` menyediakan argumen tipe `Integer`; setiap `new GList<>(head, ...)` menaruh sebuah `int` yang di-box di depan, menghasilkan `[]`, `[10]`, dan `[50, 10]`. *Kelas yang sama* lalu memuat `String`, dan `towels.head()` mengembalikan `String` tanpa cast apa pun.

Metode `toString` adalah pola rekursi-dengan-pembantu dari Subbab 9.2.3. Metode publiknya punya tanda tangan yang diperlukan setiap objek, `toString()`, dan menyerahkan pekerjaan kepada `toStringHelper(list, accumulator)` yang privat, yang parameter tambahannya adalah sebuah `ArrayList<T>` yang mengumpulkan elemen-elemen dalam perjalanan turun. Kasus dasarnya adalah list kosong, yang mengembalikan `toString` milik akumulator, `[50, 10]`. Pembantu itu juga berjaga terhadap rantai yang putus (ekor `null`) dengan pesan yang menunjukkan apa yang sudah terkumpul — berguna ketika men-debug list yang dirakit dengan tangan. `size()` adalah rekursi tanpa pembantu: satu ditambah ukuran ekornya.

Metode statis `nil()` adalah *metode generik* kita yang pertama. Metode statis tidak dapat memakai `T` milik kelas (anggota statis milik kelas, bukan milik `GList<Integer>` atau `GList<String>` tertentu), sehingga ia mendeklarasikan miliknya sendiri: `public static <T> GList<T> nil()`. Kompilator menyimpulkan `T` dari konteks, sehingga `new GList<>("blue", GList.nil())` mendapat `GList<String>`. Bila tidak ada konteks, argumen tipenya dapat diberikan secara eksplisit: `GList.<String>nil().tail()`.

> ⚠ **PERINGATAN.** Kode lama kadang mendeklarasikan metode semacam itu dengan tipe kembalian *mentah*, `public static GList nil()`, dan menulis `GList tail()` tanpa `<T>`. Kode itu lolos kompilasi, dengan peringatan, dan diam-diam mematikan pemeriksaan bagi setiap pemanggil: `GList` mentah dapat ditugaskan ke `GList<String>` apa pun isinya yang sebenarnya. Kompilasilah dengan `javac -Xlint:all`, dan perlakukan setiap peringatan "unchecked" atau "rawtypes" sebagai bug yang harus diperbaiki. Setiap listing di bab ini lolos kompilasi tanpa peringatan, kecuali di tempat tipe mentah diperlihatkan dengan sengaja, dan di sana peringatannya dibungkam secara eksplisit.

Kode 9.20 adalah versi Python-nya.

Kode 9.20 `glist_demo.py`: list generik dengan `TypeVar` dan `Generic[T]`, diperiksa oleh mypy.

```python
"""glist_demo.py: list imutabel generik dengan typing.Generic."""
from __future__ import annotations

from typing import Generic, TypeVar

T = TypeVar("T")                        # T mewakili tipe apa pun


class GList(Generic[T]):
    _head: T                            # dideklarasikan di sini, diisi hanya pada list tidak kosong
    _tail: GList[T]

    def __init__(self) -> None:         # list kosong
        self._empty = True

    @staticmethod
    def nil() -> GList[T]:
        return GList()

    @staticmethod
    def cons(head: T, tail: GList[T]) -> GList[T]:
        lst: GList[T] = GList()         # list tidak kosong
        lst._empty = False
        lst._head = head
        lst._tail = tail
        return lst

    def empty(self) -> bool:
        return self._empty

    def head(self) -> T:
        if self._empty:
            raise ValueError("Trying to access head of an empty list")
        return self._head

    def tail(self) -> GList[T]:
        if self._empty:
            raise ValueError("Trying to access tail of an empty list")
        return self._tail

    def size(self) -> int:              # rekursif: 1 + ukuran ekor
        return 0 if self._empty else 1 + self.tail().size()

    def __str__(self) -> str:
        return self._str_helper(self, [])
```

Kode 9.20 (Lanjutan) *glist_demo.py: list generik dengan TypeVar dan Generic[T], diperiksa oleh mypy.*

```
    def _str_helper(self, lst: GList[T], accumulator: list[T]) -> str:
        """Pembantu rekursif dengan parameter tambahan, yaitu akumulator."""
        if lst.empty():
            return "[" + ", ".join(str(x) for x in accumulator) + "]"
        accumulator.append(lst.head())
        return self._str_helper(lst.tail(), accumulator)


def main() -> None:
    my_glist: GList[int] = GList()
    print(f"myGList: {my_glist}")
    head = 10
    my_glist2 = GList.cons(head, my_glist)
    print(f"myGList2: {my_glist2}")
    head = 50
    my_glist3 = GList.cons(head, my_glist2)
    print(f"myGList3: {my_glist3}")

    towels: GList[str] = GList.cons("white", GList.cons("blue", GList.nil()))
    first: str = towels.head()
    print(f"towels: {towels}, head {first}, tail {towels.tail()}, size {towels.size()}")
    try:
        GList.nil().tail()
    except ValueError as e:
        print("ValueError:", e)


if __name__ == "__main__":
    main()
```

```
myGList: []
myGList2: [10]
myGList3: [50, 10]
towels: [white, blue], head white, tail [blue], size 2
ValueError: Trying to access tail of an empty list
```

Python menuliskan parameter tipe dalam dua langkah. `T = TypeVar("T")` membuat variabel tipe, dan `class GList(Generic[T])` menyatakan bahwa `GList` berparameter variabel itu. Sesudah itu, `GList[int]` dan `GList[str]` adalah tipe yang dipahami mypy, dan `towels.head()` diketahui bertipe `str`. (Python 3.12 menambahkan sintaksis yang lebih pendek, `class GList[T]:`, tanpa baris `TypeVar` [12]; listing ini memakai yang lama agar berjalan juga di Python 3.10 dan 3.11.) Dua rincian lagi: `from __future__ import annotations` memungkinkan sebuah metode menyebut `GList[T]` di dalam kelas yang masih sedang didefinisikan, dan anotasi tingkat kelas `_head: T` dan `_tail: GList[T]` mendeklarasikan atribut yang hanya diterima list tidak kosong, di dalam `cons`. Python tidak punya konstruktor yang di-overload, sehingga `nil()` dan `cons()` adalah dua cara membangun list, sama seperti `List` Java di Bab 8.

9.8.3 Metode generik, dan akhirnya sebuah Pair

Subbab 9.2.3 berakhir dengan sebuah keinginan: kelas yang memegang dua nilai dengan sembarang dua tipe, agar `fibonacciPair` dapat mengembalikan sesuatu yang mengatakan maksudnya. **Kode 9.21** mengabulkannya, bersama beberapa metode generik.

Kode 9.21 `GenericMethods.java`: kelas generik dengan dua parameter tipe, `Pair<A, B>`, dan empat metode generik, di antaranya `removeDuplicatesFromSortedList`.

```java
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

public class GenericMethods {
    // <A, B> sebelum tipe kembalian mendeklarasikan parameter tipe milik metode itu.
    static <A, B> Pair<B, A> swap(Pair<A, B> p) {
        return new Pair<>(p.getSecond(), p.getFirst());
    }

    static <T> T lastOf(List<T> list) {
        return list.get(list.size() - 1);
    }

    // {F(n), F(n-1)} lagi, kini sebagai Pair, bukan long[].
    static Pair<Long, Long> fibonacciPair(int n) {
        if (n <= 1) {
            return new Pair<>((long) n, 0L);
        }
        Pair<Long, Long> tmp = fibonacciPair(n - 1);
        return new Pair<>(tmp.getFirst() + tmp.getSecond(), tmp.getFirst());
    }

    // Menyimpan yang pertama dari setiap deret tetangga yang sama. Benar untuk masukan
TERURUT.
    static <T> List<T> removeDuplicatesFromSortedList(List<T> input) {
        List<T> result = new ArrayList<>();
        T previous = null;
        Iterator<T> iter = input.iterator();
        while (iter.hasNext()) {
            T current = iter.next();
            if (result.isEmpty() || !previous.equals(current)) {
                result.add(current);
            }
            previous = current;
        }
        return result;
    }

    public static void main(String[] args) {
        Pair<String, Integer> order = new Pair<>("L-001", 24500);
        Pair<Integer, String> swapped = swap(order);          // tipe disimpulkan
        System.out.println(order + " swapped is " + swapped);

        int price = order.getSecond();                        // di-unbox, tanpa cast
        System.out.println("price + 500 = " + (price + 500));
```

Kode 9.21 (Lanjutan) *GenericMethods.java: kelas generik dengan dua parameter tipe, Pair<A, B>, dan empat metode generik, di antaranya removeDuplicatesFromSortedList.*

```
        System.out.println("lastOf " + lastOf(List.of("wash", "dry", "iron")));
        System.out.println("lastOf " + GenericMethods.<Integer>lastOf(List.of(3, 1, 4)));

        System.out.println("fibonacciPair(10) = " + fibonacciPair(10));
        System.out.println("fibonacciPair(90).getFirst() = " +
fibonacciPair(90).getFirst());

        System.out.println(removeDuplicatesFromSortedList(
                List.of("cat", "cat", "dog", "monkey", "monkey")));
        System.out.println(removeDuplicatesFromSortedList(List.of(1, 1, 2, 3, 3, 3)));
        System.out.println(removeDuplicatesFromSortedList(new ArrayList<Double>()));
        System.out.println(removeDuplicatesFromSortedList(List.of(3, 1, 3)));   // tidak
terurut!
    }
}

// Kelas generik dengan dua parameter tipe.
final class Pair<A, B> {
    private final A first;
    private final B second;

    Pair(A first, B second) {
        this.first = first;
        this.second = second;
    }

    A getFirst() {
        return first;
    }

    B getSecond() {
        return second;
    }

    @Override
    public String toString() {
        return "(" + first + ", " + second + ")";
    }
}
```

```
(L-001, 24500) swapped is (24500, L-001)
price + 500 = 25000
lastOf iron
lastOf 4
fibonacciPair(10) = (55, 34)
fibonacciPair(90).getFirst() = 2880067194370816120
[cat, dog, monkey]
[1, 2, 3]
[]
[3, 1, 3]
```

`final class Pair<A, B>` punya dua parameter tipe, satu untuk setiap komponen. `new Pair<>("L-001", 24500)` adalah `Pair<String, Integer>`, dan `order.getSecond()` adalah `Integer` yang di-unbox ke variabel `int` bernama `price` tanpa cast.

Metode generik menulis parameter tipenya di antara pengubah (*modifier*) dan tipe kembalian. `static <A, B> Pair<B, A> swap(Pair<A, B> p)` terbaca: "untuk sembarang tipe `A` dan `B`, diberi pasangan sebuah `A` dan sebuah `B`, kembalikan pasangan sebuah `B` dan sebuah `A`". Pemanggil tidak pernah menyebut `A` atau `B`; kompilator **menyimpulkannya** dari argumen, sehingga `swap(order)` bertipe `Pair<Integer, String>`. `lastOf` berlaku untuk list bertipe apa pun dan mengembalikan tipe itu; `GenericMethods.<Integer>lastOf(...)` memperlihatkan bentuk eksplisit yang jarang

diperlukan. `fibonacciPair` kini mengembalikan `Pair<Long, Long>` — algoritma yang sama dengan **Kode 9.3**, tetapi hasilnya tercetak sebagai `(55, 34)` dan bagian-bagiannya bernama `getFirst()` dan `getSecond()`, bukan `[0]` dan `[1]`.

`removeDuplicatesFromSortedList` adalah metode generik yang berlaku untuk list berisi *apa saja* yang `equals`-nya bermakna. Ia menyimpan elemen pertama dari setiap deret tetangga yang sama. Pada list terurut, elemen-elemen yang sama bertetangga, sehingga cara ini menghapus semua duplikat dalam satu lintasan, $O(n)$: `[cat, cat, dog, monkey, monkey]` menjadi `[cat, dog, monkey]`. Panggilan terakhir memperlihatkan mengapa namanya menyebut *sorted* (terurut): pada `[3, 1, 3]` kedua angka 3 tidak bertetangga, dan tidak ada yang dihapus. Untuk masukan yang tidak terurut, perkakas yang tepat adalah himpunan — `new ArrayList<>(new LinkedHashSet<>(list))` menyimpan kemunculan pertama secara berurutan, juga dalam waktu harapan $O(n)$ — atau mengurutkan dulu, dalam $O(n \log n)$. Anda mungkin menjumpai versi lebih awal metode ini yang dimulai dengan `result.add(input.get(0))` dan menguji `previous != null`: versi itu melempar `IndexOutOfBoundsException` pada list kosong, dan salah bertingkah bila list memuat `null`. Versi dalam listing menguji `result.isEmpty()` sebagai gantinya dan menangani keduanya; panggilan dengan `ArrayList<Double>` kosong membuktikan hal yang pertama.

Kode 9.22 adalah padanannya dalam Python.

Kode 9.22 `generic_methods.py`: `Pair(Generic[A, B])` dan fungsi-fungsi generik yang bertipe `TypeVar`.

```python
"""generic_methods.py: kelas Pair generik dan fungsi generik dengan TypeVar."""
from __future__ import annotations

from typing import Generic, Sequence, TypeVar

A = TypeVar("A")
B = TypeVar("B")
T = TypeVar("T")


class Pair(Generic[A, B]):
    """Kelas generik dengan dua parameter tipe."""

    def __init__(self, first: A, second: B) -> None:
        self._first = first
        self._second = second

    @property
    def first(self) -> A:
        return self._first

    @property
    def second(self) -> B:
        return self._second

    def __str__(self) -> str:
        return f"({self._first}, {self._second})"


def swap(p: Pair[A, B]) -> Pair[B, A]:
    return Pair(p.second, p.first)


def last_of(seq: Sequence[T]) -> T:
    return seq[-1]


def fibonacci_pair(n: int) -> Pair[int, int]:
    """(F(n), F(n-1)) lagi, kini sebagai Pair, bukan tuple polos."""
    if n <= 1:
        return Pair(n, 0)
    tmp = fibonacci_pair(n - 1)
    return Pair(tmp.first + tmp.second, tmp.first)
```

Kode 9.22 (Lanjutan) *generic_methods.py: Pair(Generic[A, B]) dan fungsi-fungsi generik yang bertipe TypeVar.*

```
def remove_duplicates_from_sorted_list(items: Sequence[T]) -> list[T]:
    """Menyimpan yang pertama dari setiap deret tetangga yang sama. Benar untuk masukan
TERURUT."""
    result: list[T] = []
    for current in items:
        if not result or result[-1] != current:
            result.append(current)
    return result


def main() -> None:
    order: Pair[str, int] = Pair("L-001", 24500)
    swapped = swap(order)                          # mypy menyimpulkan Pair[int, str]
    print(f"{order} swapped is {swapped}")

    price: int = order.second
    print("price + 500 =", price + 500)

    print("lastOf", last_of(["wash", "dry", "iron"]))
    print("lastOf", last_of((3, 1, 4)))

    print("fibonacciPair(10) =", fibonacci_pair(10))
    print("fibonacciPair(90).getFirst() =", fibonacci_pair(90).first)

    print(remove_duplicates_from_sorted_list(["cat", "cat", "dog", "monkey", "monkey"]))
    print(remove_duplicates_from_sorted_list([1, 1, 2, 3, 3, 3]))
    print(remove_duplicates_from_sorted_list([]))
    print(remove_duplicates_from_sorted_list([3, 1, 3]))     # tidak terurut!


if __name__ == "__main__":
    main()
```

```
(L-001, 24500) swapped is (24500, L-001)
price + 500 = 25000
lastOf iron
lastOf 4
fibonacciPair(10) = (55, 34)
fibonacciPair(90).getFirst() = 2880067194370816120
['cat', 'dog', 'monkey']
[1, 2, 3]
[]
[3, 1, 3]
```

Fungsi generik di Python cukup memakai `TypeVar` dalam tanda tangannya: `def swap(p: Pair[A, B]) -> Pair[B, A]` mengatakan hal yang sama dengan kepala metode Java, dan mypy menyimpulkan `A` dan `B` pada setiap panggilan. `last_of(seq: Sequence[T]) -> T` menerima list, tuple, string, atau `range`, karena semuanya `Sequence`. (Untuk dua nilai yang tidak memerlukan nama, tuple biasa adalah kebiasaan Python; kelas `Pair` berguna bila komponen-komponennya layak diberi nama, atau bila tipenya perlu muncul di tanda tangan.)

9.8.4 Parameter tipe terbatas

Cobalah menambahkan pada `GList<T>` sebuah metode yang memeriksa apakah list itu terurut. Metode itu harus membandingkan tetangga, `a.compareTo(b)`, dan kompilator menolak: `T` bisa berupa kelas *apa saja*, dan tidak setiap kelas punya `compareTo`. Kelas-kelas Java pada umumnya tidak punya urutan alami — apa artinya sebuah `Scanner` datang sebelum `Scanner` lain? Kita perlu mengatakan "sembarang tipe `T`, *asalkan* dapat dibandingkan dengan `T` lain". Itulah **parameter tipe terbatas** (*bounded type parameter*):

```
class GList2<T extends Comparable<T>> { ... }
```

Bacalah `T extends Comparable<T>` sebagai "`T` harus tipe yang mengimplementasikan `Comparable<T>`". (Dalam sebuah batas, `extends` mencakup memperluas kelas maupun mengimplementasikan antarmuka.) Di dalam kelas, kompilator kini tahu bahwa setiap `T` punya metode `compareTo(T)`. Di luar, hanya tipe semacam itu yang diterima sebagai argumen tipe: `Integer`, `String`, `Double`, dan kelas-kelas `Comparable` buatan Anda sendiri, tetapi tidak `Object`. **Kode 9.23** adalah hasilnya.

Kode 9.23 `GList2Demo.java`: list yang tipe elemennya dibatasi oleh `Comparable`, dengan metode rekursif `sorted`, dan metode generik terbatas `max`.

```java
import java.util.List;

public class GList2Demo {
    // Metode generik terbatas: T harus bisa membandingkan dirinya dengan T lain.
    static <T extends Comparable<T>> T max(List<T> items) {
        T best = items.get(0);
        for (T x : items) {
            if (x.compareTo(best) > 0) {
                best = x;
            }
        }
        return best;
    }

    public static void main(String[] args) {
        GList2<Integer> myGList4 = new GList2<>();
        System.out.printf("myGList4: %s%n", myGList4);
        System.out.printf("myGList4 is sorted: %b%n", myGList4.sorted(myGList4));
        int head = 5;
        GList2<Integer> myGList5 = new GList2<>(head, myGList4);
        System.out.printf("myGList5: %s%n", myGList5);
        System.out.printf("myGList5 is sorted: %b%n", myGList5.sorted(myGList5));
        head = 10;
        GList2<Integer> myGList6 = new GList2<>(head, myGList5);
        System.out.printf("myGList6: %s%n", myGList6);
        System.out.printf("myGList6 is sorted: %b%n", myGList6.sorted(myGList6));
        head = 3;
        GList2<Integer> myGList7 = new GList2<>(head, myGList5);
        System.out.printf("myGList7: %s%n", myGList7);
        System.out.printf("myGList7 is sorted: %b%n", myGList7.sorted(myGList7));

        GList2<String> words = new GList2<>("iron", new GList2<>("wash", new GList2<>()));
        System.out.printf("words: %s is sorted: %b%n", words, words.sorted());

        System.out.println("max " + max(List.of(3, 9, 4)) + ", max "
                + max(List.of("wash", "dry", "iron")));
    }
}

// T dibatasi: hanya tipe yang mengimplementasikan Comparable<T> yang diizinkan.
class GList2<T extends Comparable<T>> {
    private final boolean empty;
    private final T head;
    private final GList2<T> tail;
```

Kode 9.23 (Lanjutan) *GList2Demo.java: list yang tipe elemennya dibatasi oleh Comparable, dengan metode rekursif sorted, dan metode generik terbatas max.*

```
    public GList2() {
        empty = true;
        head = null;
        tail = null;
    }

    public GList2(T head, GList2<T> tail) {
        this.empty = false;
        this.head = head;
        this.tail = tail;
    }

    public boolean empty() {
        return empty;
    }

    public T head() {
        if (empty) {
            throw new IllegalStateException("Trying to access head of an empty list");
        }
        return head;
    }

    public GList2<T> tail() {
        if (empty) {
            throw new IllegalStateException("Trying to access tail of an empty list");
        }
        return tail;
    }

    // Apakah list terurut tidak menurun? Bandingkan dua yang pertama, lalu rekursi.
    public boolean sorted(GList2<T> list) {
        if (list.empty() || list.tail().empty()) {
            return true;
        } else {
            T a = list.head();
            T b = list.tail().head();
            int compareResult = a.compareTo(b);   // boleh hanya karena adanya batas
            // 0 berarti keduanya sama
            // negatif (<0) berarti a datang sebelum b
            // positif (>0) berarti a datang sesudah b
            if (compareResult > 0) {
                return false;
            } else {
                return sorted(list.tail());
            }
```

Kode 9.23 (Lanjutan) *GList2Demo.java: list yang tipe elemennya dibatasi oleh Comparable, dengan metode rekursif sorted, dan metode generik terbatas max.*

```
        }
    }

    public boolean sorted() {                    // pembantu: tanpa argumen
        return sorted(this);
    }

    @Override
    public String toString() {
        StringBuilder sb = new StringBuilder("[");
        for (GList2<T> p = this; !p.empty; p = p.tail) {
            sb.append(p.head);
            if (!p.tail.empty) {
                sb.append(", ");
            }
        }
        return sb.append("]").toString();
    }
}
```

```
myGList4: []
myGList4 is sorted: true
myGList5: [5]
myGList5 is sorted: true
myGList6: [10, 5]
myGList6 is sorted: false
myGList7: [3, 5]
myGList7 is sorted: true
words: [iron, wash] is sorted: true
max 9, max wash
```

Metode `sorted(list)` bersifat rekursif dengan kedua bagian dari Subbab 9.2.2. Kasus dasar: list berisi nol atau satu elemen sudah terurut. Kasus rekursif: bandingkan dua elemen pertama; bila yang pertama datang sesudah yang kedua (`compareResult > 0`), list tidak terurut; bila tidak, list terurut tepat bila ekornya terurut. Komentar-komentarnya mengulang kontrak `compareTo` dari Bab 6: nol berarti sama, negatif berarti `a` datang sebelum `b`, positif berarti sesudahnya. Keluarannya sesuai: `[10, 5]` tidak terurut karena $10 > 5$, sedangkan `[3, 5]` terurut. Karena metode itu menerima list sebagai parameter, panggilannya berbunyi `myGList6.sorted(myGList6)`, yang janggal; listing menambahkan pembantu `sorted()` yang meneruskan `this`, sehingga pemakai cukup menulis `words.sorted()`. (Anda mungkin menjumpai metode ini dideklarasikan sebagai `public <U extends Comparable<U>> boolean sorted(GList2<T> list)`. `<U …>` itu mendeklarasikan parameter tipe kedua yang tidak pernah dipakai metodenya: tidak berbahaya, tetapi menyesatkan; batas pada `T`-lah yang membuat `compareTo` sah.)

Metode statis `max` memperlihatkan batas pada metode generik: `static <T extends Comparable<T>> T max(List<T> items)` berlaku untuk list bilangan bulat dan list string, dan menolak list berisi `Object` saat kompilasi.

Untuk melihat batas itu bekerja, tambahkan `GList2<Object> things = new GList2<>();` sebagai baris pertama `main`:

```
GList2Demo.java:16: error: type argument Object is not within bounds of type-variable T
        GList2<Object> things = new GList2<>();
               ^
  where T is a type-variable:
    T extends Comparable<T> declared in class GList2
GList2Demo.java:16: error: incompatible types: cannot infer type arguments for GList2<>
        GList2<Object> things = new GList2<>();
                                    ^
    reason: inference variable T has incompatible bounds
      equality constraints: Object
      upper bounds: Comparable<T>
  where T is a type-variable:
    T extends Comparable<T> declared in class GList2
2 errors
```

Python menyatakan batas dengan `TypeVar("T", bound=...)`. Batasnya di sini adalah sebuah **protokol** (Bab 6): `SupportsLessThan`, sembarang tipe yang punya metode `__lt__`, yang dipanggil oleh operator `<`. **Kode 9.24** memperlihatkannya.

Kode 9.24 `glist2_demo.py`: `TypeVar` yang dibatasi protokol `SupportsLessThan`; mypy menolak list berisi `object` biasa, dan Python gagal ketika mencoba membandingkannya.

```python
"""glist2_demo.py: variabel tipe terbatas - T harus mendukung operator <."""
from __future__ import annotations

from typing import Any, Generic, Protocol, Sequence, TypeVar


class SupportsLessThan(Protocol):
    def __lt__(self, other: Any, /) -> bool: ...


T = TypeVar("T", bound=SupportsLessThan)   # T dibatasi


def max_of(items: Sequence[T]) -> T:
    best = items[0]
    for x in items:
        if best < x:
            best = x
    return best


class GList2(Generic[T]):
    _head: T
    _tail: GList2[T]

    def __init__(self) -> None:
        self._empty = True

    @staticmethod
    def nil() -> GList2[T]:
        return GList2()

    @staticmethod
    def cons(head: T, tail: GList2[T]) -> GList2[T]:
        lst: GList2[T] = GList2()
        lst._empty, lst._head, lst._tail = False, head, tail
        return lst

    def empty(self) -> bool:
        return self._empty

    def head(self) -> T:
        if self._empty:
            raise ValueError("Trying to access head of an empty list")
        return self._head
```

Kode 9.24 (Lanjutan) *glist2_demo.py: TypeVar yang dibatasi protokol SupportsLessThan; mypy menolak list berisi object biasa, dan Python gagal ketika mencoba membandingkannya.*

```
    def tail(self) -> GList2[T]:
        if self._empty:
            raise ValueError("Trying to access tail of an empty list")
        return self._tail

    def sorted(self) -> bool:
        """Apakah list terurut tidak menurun? Bandingkan dua yang pertama, lalu rekursi."""
        if self.empty() or self.tail().empty():
            return True
        a, b = self.head(), self.tail().head()
        if b < a:                               # boleh hanya karena adanya batas
            return False
        return self.tail().sorted()

    def __str__(self) -> str:
        parts = []
        p = self
        while not p.empty():
            parts.append(str(p.head()))
            p = p.tail()
        return "[" + ", ".join(parts) + "]"


def main() -> None:
    my_glist4: GList2[int] = GList2.nil()
    print(f"myGList4: {my_glist4}")
    print(f"myGList4 is sorted: {my_glist4.sorted()}")
    my_glist5 = GList2.cons(5, my_glist4)
    print(f"myGList5: {my_glist5}")
    print(f"myGList5 is sorted: {my_glist5.sorted()}")
    my_glist6 = GList2.cons(10, my_glist5)
    print(f"myGList6: {my_glist6}")
    print(f"myGList6 is sorted: {my_glist6.sorted()}")
    my_glist7 = GList2.cons(3, my_glist5)
    print(f"myGList7: {my_glist7}")
    print(f"myGList7 is sorted: {my_glist7.sorted()}")

    words = GList2.cons("iron", GList2.cons("wash", GList2.nil()))
    print(f"words: {words} is sorted: {words.sorted()}")
    print(f"max {max_of([3, 9, 4])}, max {max_of(['wash', 'dry', 'iron'])}")
```

Kode 9.24 (Lanjutan) *glist2_demo.py: TypeVar yang dibatasi protokol SupportsLessThan; mypy menolak list berisi object biasa, dan Python gagal ketika mencoba membandingkannya.*

```
    things = GList2.cons(object(), GList2.cons(object(), GList2.nil()))   # mypy: galat
    try:
        things.sorted()
    except TypeError as e:
        print("TypeError:", e)


if __name__ == "__main__":
    main()
```

```
myGList4: []
myGList4 is sorted: True
myGList5: [5]
myGList5 is sorted: True
myGList6: [10, 5]
myGList6 is sorted: False
myGList7: [3, 5]
myGList7 is sorted: True
words: [iron, wash] is sorted: True
max 9, max wash
TypeError: '<' not supported between instances of 'object' and 'object'
```

`sorted` versi Python adalah metode milik list itu sendiri (`self.tail().sorted()`), sehingga tidak memerlukan pembantu. Baris-baris terakhir membangun list berisi dua `object()` biasa. Python menjalankan pembangunannya dengan senang hati dan baru gagal di dalam `sorted`, ketika `<` bertemu dua objek yang tidak dapat dibandingkan. mypy berkeberatan tepat di baris tempat list itu dibangun, yaitu tempat kesalahannya berada:

```
glist2_demo.py:88: error: Value of type variable "T" of "cons" of "GList2" cannot be
"object"  [type-var]
Found 1 error in 1 file (checked 1 source file)
```

9.8.5 Wildcard: mengapa List bukan List

Inilah teka-teki yang pasti dijumpai setiap pemrogram Java. `Cat` memperluas `Animal`, jadi `Cat` *adalah* `Animal`. Apakah karena itu `List<Cat>` adalah `List<Animal>`? Tampaknya jelas. Ternyata salah:

```
List<Cat> cats = new ArrayList<Cat>();
List<Animal> animals = cats;          // galat kompilasi
```

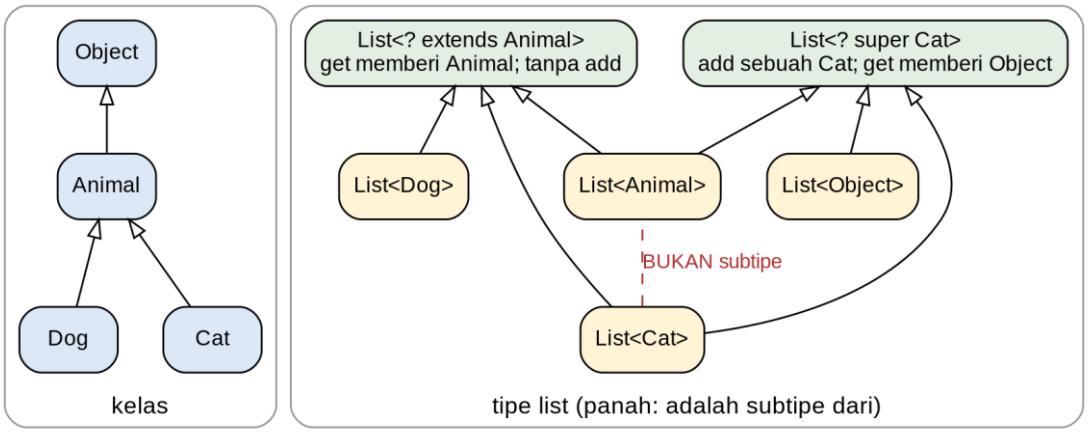
Andaikan Java mengizinkan baris kedua. `animals` dan `cats` lalu merujuk ke objek list yang *sama*, dan karena `animals` bertipe `List<Animal>`, baris berikut pun akan lolos kompilasi:

```
animals.add(new Dog("wondergirl"));   // Dog adalah Animal, jadi mengapa tidak?
```

Kini list yang dijanjikan `cats` hanya berisi kucing memuat seekor anjing, dan `Cat c = cats.get(2)` berikutnya gagal. Satu-satunya aturan yang aman adalah bahwa `List<Cat>` dan `List<Animal>` tipe yang tidak berhubungan, walaupun `Cat` dan `Animal` berhubungan. Secara umum, untuk sembarang tipe generik G:

$$Cat <: Animal \quad \not\Rightarrow \quad G\langle Cat\rangle <: G\langle Animal\rangle \tag{9.7}$$

dengan $<:$ dibaca "adalah subtipe dari". Tipe generik yang mematuhi Persamaan (9.7) disebut **invarian**. **Gambar 9.8** menggambarkan keadaannya, termasuk dua jalan keluar yang disediakan Java, yaitu **wildcard**.



Gambar 9.8 Hubungan subtipe di antara kelas (kiri) dan di antara tipe list (kanan). `List<Cat>` bukan subtipe `List<Animal>`, tetapi keduanya subtipe `List<? extends Animal>`, dan `List<Cat>`, `List<Animal>`, serta `List<Object>` semuanya subtipe `List<? super Cat>`.

Wildcard `?` berarti "suatu tipe yang tidak saya sebut namanya". Ia hadir dalam dua bentuk terbatas:

- `List<? extends Animal>` berarti "list berisi *suatu* subtipe `Animal`" — mungkin `List<Animal>`, mungkin `List<Cat>`, mungkin `List<Dog>`. Apa pun itu, setiap elemennya adalah `Animal`, sehingga **membaca** aman: `get` mengembalikan `Animal`. Tetapi **menambah** terlarang, karena kompilator tidak tahu subtipe mana yang sebenarnya dimuat list itu; bisa saja list kucing, dan elemen barunya bisa saja anjing. Satu-satunya nilai yang dapat ditambahkan adalah `null`.
- `List<? super Cat>` berarti "list berisi *suatu* supertipe `Cat`" — `List<Cat>`, `List<Animal>`, atau `List<Object>`. Apa pun itu, ia dapat memuat `Cat`, sehingga **menambah** `Cat` aman. Tetapi membaca hanya memberi `Object`, karena kompilator tidak tahu supertipe mana itu.

Jembatan keledai dari Joshua Bloch adalah **PECS: producer `extends`, consumer `super`** (produsen `extends`, konsumen `super`) [2]. Parameter yang darinya metode hanya *mengambil* nilai menghasilkan nilai, dan sebaiknya bertipe `? extends T`; parameter yang ke dalamnya metode hanya *menaruh* nilai mengonsumsi nilai, dan sebaiknya bertipe `? super T`. **Tabel 9.3** merangkumnya.

Tabel 9.3 Apa yang boleh dipegang setiap jenis deklarasi list, dan apa yang boleh Anda baca dan tulis melaluinya.

Deklarasi	Boleh memegang	`get` mengembalikan	`add` menerima	Dipakai untuk
`List<Animal>`	tepat `List<Animal>`	`Animal`	`Animal` (dan subkelasnya)	membaca dan menulis
`List<? extends Animal>`	`List<Animal>`, `List<Cat>`, `List<Dog>`	`Animal`	tidak ada (hanya `null`)	*produsen*: hanya dibaca
`List<? super Cat>`	`List<Cat>`, `List<Animal>`, `List<Object>`	`Object`	`Cat` (dan subkelasnya)	*konsumen*: hanya ditulisi
`List<?>`	list berisi apa saja	`Object`	tidak ada (hanya `null`)	ukuran, pengosongan, pencetakan

Kode 9.25 mengonkretkan semua ini dengan kelas `Animal`, `Cat`, dan `Dog`.

Kode 9.25 `WildcardDemo.java`: `Animal`, `Cat`, dan `Dog`; metode pembaca dengan `? extends Animal`; metode `addAll(src, dest)` dengan `? super T`; dan larik kovarian yang gagal saat eksekusi.

```
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

public class WildcardDemo {
    // Membaca dari src (produsen T) dan menulis ke dest (konsumen T).
    public static <T> void addAll(List<T> src, List<? super T> dest) {
        for (T element : src) {        // tambahkan setiap elemen src ke dest
            dest.add(element);
        }
    }

    // Menerima List<Animal>, List<Cat>, atau List<Dog>: hanya membaca.
    static String names(List<? extends Animal> animals) {
        StringBuilder sb = new StringBuilder();
        for (Animal a : animals) {
            sb.append(a.getName()).append(' ');
        }
        return sb.toString().trim();
    }

    public static void main(String[] args) {
        Cat cat1 = new Cat();
        cat1.setName("superman");
        Cat cat2 = new Cat();
        cat2.setName("spiderman");
        List<Cat> cats = new ArrayList<Cat>();
        cats.add(cat1);
        cats.add(cat2);
        Iterator<Cat> iterator = cats.iterator();
        while (iterator.hasNext()) {
            Cat c = iterator.next();                    // tanpa cast: iteratornya bertipe
            System.out.println(c.getName());
        }

        Dog dog1 = new Dog();
        dog1.setName("supergirl");
        Dog dog2 = new Dog();
        dog2.setName("spidergirl");
        List<Dog> dogs = new ArrayList<>();
        dogs.add(dog1);
        dogs.add(dog2);
        System.out.println("dogs: " + names(dogs));
```

Kode 9.25 (Lanjutan) *WildcardDemo.java: Animal, Cat, dan Dog; metode pembaca dengan ? extends Animal; metode addAll(src, dest) dengan ? super T; dan larik kovarian yang gagal saat eksekusi.*

```
        // List<Animal> zoo = cats;                    // galat kompilasi: lihat teks
        List<? extends Animal> animals = cats;        // view baca-saja sebagai Animal
        Animal animal1 = animals.get(0);
        System.out.println(animal1.getName() + " / " + animals.get(1).getName()
                + " / " + animal1);
        // animals.add(dog1);                         // galat kompilasi: lihat teks

        List<Animal> newAnimals = new ArrayList<>();
        addAll(cats, newAnimals);                     // T = Cat, dest is List<? super Cat>
        addAll(dogs, newAnimals);                     // T = Dog
        System.out.println("newAnimals: " + names(newAnimals));
        List<Object> things = new ArrayList<>();
        addAll(dogs, things);                         // Object juga supertipe Dog
        System.out.println("things: " + things);

        // Larik bersifat kovarian, sehingga kesalahan yang sama baru tertangkap saat
eksekusi.
        Animal[] pets = new Cat[2];
        try {
            pets[0] = dog1;
        } catch (ArrayStoreException e) {
            System.out.println("ArrayStoreException: " + e.getMessage());
        }
    }
}

class Animal {
    private String name;

    public Animal() {
    }

    public Animal(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
```

Kode 9.25 (Lanjutan) *WildcardDemo.java: Animal, Cat, dan Dog; metode pembaca dengan ? extends Animal; metode addAll(src, dest) dengan ? super T; dan larik kovarian yang gagal saat eksekusi.*

```
    @Override
    public String toString() {
        return "This is Animal class";
    }
}

class Cat extends Animal {
    public Cat() {
    }

    @Override
    public String toString() {
        return "This is Cat class";
    }
}

class Dog extends Animal {
    public Dog() {
    }

    public Dog(String name) {
        super(name);
    }

    @Override
    public String toString() {
        return "This is Dog class";
    }
}
```

```
superman
spiderman
dogs: supergirl spidergirl
superman / spiderman / This is Cat class
newAnimals: superman spiderman supergirl spidergirl
things: [This is Dog class, This is Dog class]
ArrayStoreException: Dog
```

`Iterator<Cat>` yang bertipe tidak memerlukan cast. Metode `names(List<? extends Animal>)` menerima list anjing di sini serta list kucing dan list campuran nanti: ia hanya membaca, sehingga `? extends` tepat sekali. Penugasan `List<? extends Animal> animals = cats` sah, dan `animals.get(0)` memberi `Animal` bernama `superman`; mencetaknya menjalankan `toString` milik `Cat`, *This is Cat class*, melalui *dynamic dispatch* (Bab 7).

Hapus `//` di depan kedua baris yang bertanda "galat kompilasi" dan `javac` menjelaskan kedua aturan itu dengan kata-katanya sendiri:

```
WildcardDemo.java:45: error: incompatible types: List<Cat> cannot be converted to
List<Animal>
        List<Animal> zoo = cats;
                           ^
WildcardDemo.java:50: error: incompatible types: Dog cannot be converted to CAP#1
        animals.add(dog1);
                    ^
  where CAP#1 is a fresh type-variable:
    CAP#1 extends Animal from capture of ? extends Animal
Note: Some messages have been simplified; recompile with -Xdiags:verbose to get full output
2 errors
```

Dalam pesan kedua, `CAP#1` adalah nama yang diberikan kompilator bagi tipe tak dikenal di balik `?` ini; yang diketahuinya hanyalah bahwa `CAP#1` memperluas `Animal`, dan sebuah `Dog` belum tentu termasuk di dalamnya.

Metode `addAll(List<T> src, List<? super T> dest)` menyalin setiap elemen `src` ke dalam `dest`. `src` adalah produsen dan bahkan boleh bertipe `List<? extends T>`; `dest` adalah konsumen, `? super T`. Dengan `T = Cat`, `dest` boleh berupa `List<Animal>`; dengan `T = Dog`, `List<Animal>` yang sama lagi, yang kini memuat dua kucing dan dua anjing; dan `List<Object>` pun boleh. `Collections.addAll` dan `Collections.copy` milik pustaka Java dideklarasikan persis dengan cara ini.

Blok terakhir menjelaskan mengapa para perancang Java begitu ketat dengan generik: mereka sudah pernah tersengat oleh **larik**, yang bersifat **kovarian** — `Cat[]` *adalah* `Animal[]`. Jadi `Animal[] pets = new Cat[2]` lolos kompilasi, dan `pets[0] = dog1` pun lolos. Larik itu mengingat saat eksekusi bahwa ia sebenarnya `Cat[]`, dan penyimpanannya gagal dengan `ArrayStoreException: Dog`. Generik justru menangkap kesalahan yang sama saat kompilasi.

Pemeriksa tipe Python menerapkan penalaran yang sama, dan **Kode 9.26** membiarkan kita menyaksikan apa yang terjadi bila tidak ada yang memeriksa.

Kode 9.26 `wildcard_demo.py`: `Sequence[Animal]` menerima list kucing (baca-saja, kovarian); `list[Animal]` tidak (invarian), dan bila aturan itu diabaikan, seekor anjing berakhir di antara kucing-kucing.

```python
"""wildcard_demo.py: varians di Python - list invarian, Sequence kovarian."""
from __future__ import annotations

from typing import Iterable, Sequence, TypeVar

T = TypeVar("T")


class Animal:
    def __init__(self, name: str = "") -> None:
        self.name = name

    def __str__(self) -> str:
        return "This is Animal class"


class Cat(Animal):
    def __str__(self) -> str:
        return "This is Cat class"


class Dog(Animal):
    def __str__(self) -> str:
        return "This is Dog class"


def add_all(src: Iterable[T], dest: list[T]) -> None:
    """Membaca dari src dan menambahkan ke dest."""
    for element in src:
        dest.append(element)


def names(animals: Sequence[Animal]) -> str:
    """Menerima list Animal, Cat, atau Dog: Sequence bersifat baca-saja."""
    return " ".join(a.name for a in animals)


def main() -> None:
    cats: list[Cat] = [Cat("superman"), Cat("spiderman")]
    for c in cats:
        print(c.name)
    dogs: list[Dog] = [Dog("supergirl"), Dog("spidergirl")]
    print("dogs:", names(dogs))
```

Kode 9.26 (Lanjutan) *wildcard_demo.py: Sequence[Animal] menerima list kucing (baca-saja, kovarian); list[Animal] tidak (invarian), dan bila aturan itu diabaikan, seekor anjing berakhir di antara kucing-kucing.*

```
    animals: Sequence[Animal] = cats              # aman: view baca-saja
    print(f"{animals[0].name} / {animals[1].name} / {animals[0]}")

    new_animals: list[Animal] = []
    add_all(cats, new_animals)
    add_all(dogs, new_animals)
    print("newAnimals:", names(new_animals))
    things: list[object] = []
    add_all(dogs, things)
    print("things:", [str(t) for t in things])

    # Yang dilarang Java saat kompilasi diizinkan Python saat eksekusi.
    zoo: list[Animal] = cats                      # mypy: galat (list invarian)
    zoo.append(Dog("wondergirl"))
    print("cats now holds:", [str(c) for c in cats])


if __name__ == "__main__":
    main()
```

```
superman
spiderman
dogs: supergirl spidergirl
superman / spiderman / This is Cat class
newAnimals: superman spiderman supergirl spidergirl
things: ['This is Dog class', 'This is Dog class']
cats now holds: ['This is Cat class', 'This is Cat class', 'This is Dog class']
```

Dalam sistem tipe Python, `list[T]` bersifat invarian, persis seperti `List<T>` Java, dan `Sequence[T]` — view *baca-saja* tanpa `append` — bersifat **kovarian**: `list[Cat]` boleh diberikan di tempat yang mengharapkan `Sequence[Animal]`. Jadi `Sequence[Animal]` memainkan peran `List<? extends Animal>`. Untuk sisi konsumen Python tidak memerlukan `? super`: fungsi `add_all(src: Iterable[T], dest: list[T])` menyelesaikan `T = Animal` untuk panggilan dengan tujuan `list[Animal]`, dan menerima `cats` sebagai `Iterable[Animal]` karena `Iterable` pun kovarian. Tiga baris terakhir `main` melakukan apa yang ditolak Java: `zoo: list[Animal] = cats`, lalu `zoo.append(Dog("wondergirl"))`. Python menjalankannya, dan keluarannya memperlihatkan akibatnya: list kucing kini memuat seekor anjing. mypy menunjuk baris yang bersalah, bahkan menyarankan perbaikannya:

```
wildcard_demo.py:57: error: Incompatible types in assignment (expression has type
"list[Cat]", variable has type "list[Animal]")  [assignment]
wildcard_demo.py:57: note: "list" is invariant -- see
https://mypy.readthedocs.io/en/stable/common_issues.html#variance
wildcard_demo.py:57: note: Consider using "Sequence" instead, which is covariant
Found 1 error in 1 file (checked 1 source file)
```

9.8.6 Type erasure: apa yang tersisa saat eksekusi

Bagaimana mesin virtual Java menjalankan `List<String>` dan `List<Integer>`? Ia tidak menjalankannya. Generik ditambahkan ke Java pada tahun 2004, dua belas tahun setelah mesin virtualnya dirancang, dan harus bekerja dengan berkas kelas dan pustaka yang sudah ada. Karena itu Java mengimplementasikannya dengan **type erasure** (penghapusan tipe): kompilator memeriksa setiap pemakaian argumen tipe, menyisipkan cast yang dahulu harus ditulis tangan pada kode mentah, lalu *menghapus* argumen-argumen tipe itu. Di dalam berkas kelas, `List<String>` hanyalah `List`, dan parameter tipe `T` menjadi batasnya, `Object` atau `Comparable` [1], [13]. **Kode 9.27** memperlihatkan akibat-akibatnya.

Kode 9.27 `ErasureDemo.java`: saat eksekusi `List<String>` dan `List<Integer>` adalah kelas yang sama; referensi mentah dapat menyelundupkan `Integer` ke dalam `List<String>`, dan `Collections.checkedList` menghentikannya di pintu.

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

public class ErasureDemo {
    @SuppressWarnings({"rawtypes", "unchecked"})
    static void sneakIn(List list) {       // parameter mentah: kompilator berhenti
memeriksa
        list.add(42);
    }

    public static void main(String[] args) {
        List<String> strings = new ArrayList<>();
        List<Integer> numbers = new ArrayList<>();
        System.out.println("same class at run time: "
                + (strings.getClass() == numbers.getClass()));
        System.out.println("class name: " + strings.getClass().getName());

        // Polusi heap: sebuah Integer di dalam List<String>.
        strings.add("Siti");
        sneakIn(strings);                  // di sini tidak ada yang gagal ...
        System.out.println("size " + strings.size() + ", contents " + strings);
        try {
            String s = strings.get(1);     // ... kegagalan muncul di sini, jauh dari
sebabnya
            System.out.println(s);
        } catch (ClassCastException e) {
            System.out.println("ClassCastException at get(1)");
        }

        // View yang diperiksa justru gagal tepat saat add yang salah.
        List<String> guarded = Collections.checkedList(new ArrayList<>(), String.class);
        guarded.add("Budi");
        try {
            sneakIn(guarded);
        } catch (ClassCastException e) {
            System.out.println("checkedList refused: " + e.getMessage());
        }
    }
}
```

```
same class at run time: true
class name: java.util.ArrayList
size 2, contents [Siti, 42]
ClassCastException at get(1)
checkedList refused: Attempt to insert class java.lang.Integer element into collection with
element type class java.lang.String
```

Dua baris pertama membuktikannya: saat eksekusi kedua list hanyalah `java.util.ArrayList`. Blok berikutnya memperlihatkan **polusi heap** (*heap pollution*). Metode `sneakIn` menerima `List` *mentah*, yang mematikan pemeriksaan di dalamnya, lalu menambahkan `42` ke dalam sesuatu yang diyakini pemanggilnya sebagai `List<String>`. Tidak ada yang gagal pada `add`, karena saat eksekusi tidak ada `String` di dalam tipe list itu untuk diperiksa. List itu kini tercetak sebagai `[Siti, 42]`. Kegagalannya muncul belakangan, pada `strings.get(1)`, tempat kompilator menyisipkan cast ke `String` atas nama kita. Itulah sebabnya tipe mentah dan peringatan "unchecked" patut dicurigai: keduanya satu-satunya cara untuk mematahkan jaminan **Kode 9.17**. Bila Anda harus menyerahkan koleksi kepada kode yang tidak Anda percayai, bungkuslah dengan `Collections.checkedList(list, String.class)`: view yang diperiksa itu *mengingat* tipe

elemennya dan menolak elemen yang salah tepat saat ditambahkan, seperti diperlihatkan baris terakhir.

Erasure juga menjelaskan empat hal yang tidak dapat Anda tulis di Java. Diberikan berkas `Limits.java`

```
import java.util.List;

class Limits<T> {
    T make() { return new T(); }
    T[] many() { return new T[10]; }
    boolean test(Object o) { return o instanceof List<String>; }
    List<int> numbers;
}
```

`javac` menjawab:

```
Limits.java:7: error: unexpected type
    List<int> numbers;
         ^
  required: reference
  found:    int
Limits.java:4: error: unexpected type
    T make() { return new T(); }
                        ^
  required: class
  found:    type parameter T
  where T is a type-variable:
    T extends Object declared in class Limits
Limits.java:5: error: generic array creation
    T[] many() { return new T[10]; }
                         ^
Limits.java:6: error: Object cannot be safely cast to List<String>
    boolean test(Object o) { return o instanceof List<String>; }
                                    ^
4 errors
```

Saat eksekusi tidak ada `T` untuk dibuat atau dijadikan larik (obatnya adalah menyerahkan sebuah pabrik atau `Class<T>`; `ArrayList` sendiri menyimpan `Object[]`), tidak ada `String` tersisa untuk diperiksa dengan `instanceof` (`instanceof List<?>` diizinkan), dan argumen tipe harus berupa tipe referensi, sehingga bilangan masuk ke `List<Integer>`, dengan *boxing* (Bab 5).

Python melangkah lebih jauh daripada erasure: argumen tipe *tidak pernah* menjadi bagian dari program yang berjalan. **Kode 9.28** memperlihatkan apa yang tersisa.

Kode 9.28 `erasure_demo.py`: `list[str]` hanyalah alias bagi `list`; tidak ada yang memeriksa argumen tipe saat eksekusi, tetapi petunjuk tipe tetap tersedia bagi perkakas.

```
"""erasure_demo.py: argumen tipe di Python untuk perkakas, bukan untuk program yang
berjalan."""
from typing import Generic, TypeVar, get_origin, get_type_hints

T = TypeVar("T")


class Box(Generic[T]):
    def __init__(self, item: T) -> None:
        self.item = item


class Shelf:
    boxes: list[Box[int]]


def main() -> None:
    strings: list[str] = []
    numbers: list[int] = []
    print("same class at run time:", type(strings) is type(numbers))
    print("class name:", type(strings).__name__)

    alias = list[str]
    print("list[str] is", repr(alias), "- its origin is", get_origin(alias))
    made = list[str]([1, 2])           # memanggil alias hanya membuat list biasa
    print("list[str]([1, 2]) =", made)
    try:
        isinstance(strings, list[str])
    except TypeError as e:
        print("TypeError:", e)

    b = Box[int]("not an int")         # tidak ada yang memeriksa argumennya
    print("Box[int] holds", repr(b.item), "- remembered:", getattr(b, "__orig_class__"))
    print("hints kept for tools:", get_type_hints(Shelf))


if __name__ == "__main__":
    main()
```

```
same class at run time: True
class name: list
list[str] is list[str] - its origin is <class 'list'>
list[str]([1, 2]) = [1, 2]
TypeError: isinstance() argument 2 cannot be a parameterized generic
Box[int] holds 'not an int' - remembered: __main__.Box[int]
hints kept for tools: {'boxes': list[__main__.Box[int]]}
```

`list[str]` adalah objek alias kecil yang *asal*-nya (*origin*) adalah `list`; memanggilnya membuat list biasa, bahkan berisi bilangan bulat. `isinstance(x, list[str])` ditolak dengan `TypeError`, dengan alasan yang sama seperti `instanceof List<String>` di Java. `Box[int]` dengan senang hati memuat sebuah string, walaupun objek itu mengingat, dalam atribut `__orig_class__`, alias mana yang membuatnya. Dan petunjuk yang ditulis dalam program disimpan sebagai data, untuk perkakas: `typing.get_type_hints(Shelf)` mengembalikannya. mypy, tentu saja, melaporkan setiap kesalahan yang disengaja dalam berkas ini (empat galat, pada baris 24, 27, dan 31).

9.8.7 Generik di Java dan Python: ringkasan

Tabel 9.4 mengumpulkan kosakata subbab ini dalam kedua bahasa.

Tabel 9.4 Pemrograman generik di Java dan di Python.

Gagasan	Java	Python (diperiksa mypy)
parameter tipe sebuah kelas	`class Box<T>`	`T = TypeVar("T")`, `class Box(Generic[T])`; sejak 3.12 `class Box[T]:`
argumen tipe	`Box<Integer>`	`Box[int]`
metode atau fungsi generik	`static <T> T first(List<T> xs)`	`def first(xs: Sequence[T]) -> T`
parameter terbatas	`<T extends Comparable<T>>`	`TypeVar("T", bound=SupportsLessThan)`
tipe koleksi bawaan	`List<String>`, `Map<String, Integer>`	`list[str]`, `dict[str, int]` (3.9+)
view baca-saja, kovarian	`List<? extends Animal>`	`Sequence[Animal]`, `Iterable[Animal]`
tulis-saja, kontravarian	`List<? super Cat>`	jarang diperlukan; `TypeVar` yang diselesaikan saat panggilan
tipe elemen tak dikenal	`List<?>`	`list[Any]`, `Sequence[object]`
yang dilihat program saat berjalan	dihapus: `List`	tidak ada: `list`
siapa yang menegakkannya	`javac`, selalu	mypy atau pemeriksa lain, bila Anda menjalankannya

9.9 Menata Urutan: Comparable dan Comparator

Sebuah list dapat diurutkan dengan `Collections.sort(list)` atau `list.sort(...)`, tetapi tidak setiap list dapat diurutkan: pengurutan harus dapat bertanya tentang sembarang dua elemen, "mana yang datang lebih dulu?". Ada tepat dua cara menjawabnya:

1. Kelas elemennya mengimplementasikan **`Comparable<T>`**, yang satu-satunya metodenya, `compareTo(T other)`, mendefinisikan *urutan alami* kelas itu (Bab 6). `String`, `Integer`, dan `LocalDate` semuanya melakukannya. Maka `Collections.sort(list)` tidak memerlukan apa-apa lagi.
2. Kita menulis sebuah **`Comparator<T>`**, objek terpisah yang metodenya `compare(a, b)` mendefinisikan *suatu* urutan, dipilih dari luar kelas, lalu menyerahkannya kepada pengurutan: `Collections.sort(list, comparator)`. Sebuah kelas punya satu urutan alami tetapi boleh punya comparator sebanyak apa pun: menurut umur, menurut nama, menurut kunjungan, terbesar dulu.

Kedua metode mengikuti kontrak yang sama: hasilnya negatif bila argumen pertama datang lebih dulu, nol bila keduanya sama dalam urutan ini, positif bila yang pertama datang sesudahnya [5]. Urutan yang benar juga harus *konsisten*: bila *a* datang sebelum *b* dan *b* sebelum *c*, maka *a* sebelum *c*; bila tidak, pengurutan dapat gagal dengan `IllegalArgumentException: Comparison method violates its general contract!`.

9.9.1 Iterator, duplikat, iterable, dan sebuah comparator

Kode 9.29 adalah tur lima bagian yang mengikat beberapa gagasan bab ini, diakhiri dengan sebuah `Comparator` untuk mahasiswa.

Kode 9.29 `MyCollection.java`: sebuah iterator, fakta bahwa iterator hanya membaca, menghapus duplikat dari list terurut, sebuah `Iterable` dalam perulangan for-each, dan mengurutkan mahasiswa dengan `AgeComparator`.

```
import java.util.ArrayList;
import java.util.Collection;
import java.util.Collections;
import java.util.Comparator;
import java.util.Iterator;
import java.util.List;

public class MyCollection {
    public static <T> List<T> removeDuplicatesFromSortedList(List<T> input) {
        List<T> result = new ArrayList<>();
        if (input.isEmpty()) {
            return result;
        }
        T previous = null;
        result.add(input.get(0));
        Iterator<T> iter = input.iterator();
        while (iter.hasNext()) {
            T current = iter.next();
            if (previous != null && !previous.equals(current)) {
                result.add(current);
            }
            previous = current;
        }
        return result;
    }

    public static void main(String[] args) {
        System.out.println("Part 1: Iterator");
        Collection<String> myCollection = new ArrayList<>();
        myCollection.add("cat");
        Iterator<String> iterator = myCollection.iterator();
        while (iterator.hasNext()) {
            System.out.println(iterator.next());
        }

        System.out.printf("%nPart 2: Iterator is only read not change%n");
        Iterator<String> iterator2 = myCollection.iterator();
        while (iterator2.hasNext()) {
            String element = iterator2.next();   // tidak ada iterator2.prev()
            element += " and dog";                // hanya mengubah salinan lokal
            System.out.println(element);
        }
        System.out.println("collection still holds " + myCollection);
```

Kode 9.29 (Lanjutan) *MyCollection.java: sebuah iterator, fakta bahwa iterator hanya membaca, menghapus duplikat dari list terurut, sebuah Iterable dalam perulangan for-each, dan mengurutkan mahasiswa dengan AgeComparator.*

```
        System.out.printf("%nPart 3: Removing duplicate values (sorted list only)%n");
        myCollection.add("cat");
        myCollection.add("dog");
        myCollection.add("monkey");
        myCollection.add("monkey");
        Iterator<String> iterator3 = myCollection.iterator();
        while (iterator3.hasNext()) {
            System.out.println(iterator3.next());
        }
        System.out.printf("%nAfter removing the duplicate values%n");
        Collection<String> myCollection2 =
                removeDuplicatesFromSortedList((List<String>) myCollection);
        Iterator<String> iterator4 = myCollection2.iterator();
        while (iterator4.hasNext()) {
            System.out.println(iterator4.next());
        }

        System.out.printf("%nPart 4: Iterable%n");
        Collection<Integer> it = new ArrayList<>();
        it.add(3);
        it.add(5);
        it.add(7);
        int sum = 0;
        for (Integer i : it) {                    // berlaku untuk apa pun yang Iterable
            sum += i;
        }
        System.out.println("sum = " + sum);

        System.out.printf("%nPart 5: Comparator%n");
        Collection<Student> students = new ArrayList<>();
        Student s = new Student(15);
        students.add(s);
        Student s1 = new Student(20);
        students.add(s1);
        Student s2 = new Student(12);
        students.add(s2);
        Iterator<Student> itStudents = students.iterator();
        while (itStudents.hasNext()) {
            System.out.println(itStudents.next().getAge());
        }
        AgeComparator age = new AgeComparator();
        System.out.printf("Age difference between s1 (age %d) and s2 (age %d) = %d%n",
                s1.getAge(), s2.getAge(), age.compare(s1, s2));
        System.out.println("After sort is called, the list of student becomes:");
        Collections.sort((List<Student>) students, new AgeComparator());
        Iterator<Student> itStudents2 = students.iterator();
```

Kode 9.29 (Lanjutan) *MyCollection.java: sebuah iterator, fakta bahwa iterator hanya membaca, menghapus duplikat dari list terurut, sebuah Iterable dalam perulangan for-each, dan mengurutkan mahasiswa dengan AgeComparator.*

```
        while (itStudents2.hasNext()) {
            System.out.println(itStudents2.next().getAge());
        }
    }
}

class Student {
    private final int age;

    Student(int age) {
        this.age = age;
    }

    public int getAge() {
        return age;
    }
}

// Memakai list -> urutkan semua elemen dengan Collections.sort().
// Untuk mengurutkan list objek: (1) kelasnya mengimplementasikan Comparable, atau
// (2) kita menulis kelas Comparator untuk objek yang ingin diurutkan.
class AgeComparator implements Comparator<Student> {
    @Override
    public int compare(Student s1, Student s2) {
        if (s1 == s2) {
            return 0;               // objek yang sama (atau keduanya null)
        } else if (s1 == null) {
            return -1;              // null datang lebih dulu
        } else if (s2 == null) {
            return 1;
        } else {
            return s1.getAge() - s2.getAge();   // yang lebih muda dulu
        }
    }
}
```

```
Part 1: Iterator
cat

Part 2: Iterator is only read not change
cat and dog
collection still holds [cat]

Part 3: Removing duplicate values (sorted list only)
cat
cat
dog
monkey
monkey

After removing the duplicate values
cat
dog
monkey

Part 4: Iterable
sum = 15

Part 5: Comparator
15
20
12
Age difference between s1 (age 20) and s2 (age 12) = 8
After sort is called, the list of student becomes:
12
15
20
```

Bagian 1 menelusuri koleksi satu elemen dengan `Iterator` eksplisit. Perhatikan tipe yang dideklarasikan, `Collection<String>`, antarmuka akar dalam **Gambar 9.2**: untuk bagian ini kita tidak memerlukan lebih dari sekelompok elemen.

Bagian 2 mengejutkan banyak pemula: perulangan mencetak `cat and dog`, tetapi koleksinya masih memuat `[cat]`. `element` menerima *referensi* ke `"cat"`; `+=` membangun string baru dan mengarahkan variabel lokal itu kepadanya, tanpa menyentuh referensi milik koleksi. Untuk mengubah elemen list, panggillah `set` (atau `ListIterator.set`; `ListIterator` juga dapat bergerak mundur, yang tidak dapat dilakukan `Iterator` biasa).

Bagian 3 menghapus duplikat dari `[cat, cat, dog, monkey, monkey]` dengan bentuk asli metode dari Subbab 9.8.3, kini dijaga oleh uji `isEmpty`. Cast `(List<String>) myCollection` diperlukan karena variabelnya dideklarasikan sebagai `Collection`; mendeklarasikan antarmuka yang benar-benar dibutuhkan kode, `List`, akan menghindarkannya.

Bagian 4 memperlihatkan bahwa perulangan for-each berlaku untuk apa pun yang `Iterable`: $3 + 5 + 7 = 15$.

Bagian 5 mengurutkan tiga mahasiswa menurut umur dengan `AgeComparator`. Perhatikan `compare`-nya. `s1 == s2` menangkap "objek yang sama" sekaligus "keduanya `null`". Dua uji berikutnya membuat `null` terurut paling depan. Baris terakhir, `s1.getAge() - s2.getAge()`, adalah jalan pintas klasik: selisihnya negatif bila `s1` lebih muda, nol bila sama, positif bila lebih tua. Dicetak untuk `s1` (umur 20) dan `s2` (umur 12), hasilnya $20 - 12 = 8$, positif, sehingga `s1` datang sesudah `s2`. Setelah `Collections.sort` umur-umurnya keluar `12 15 20`. Trik pengurangan ini aman untuk umur, tetapi Subbab 9.9.2 memperlihatkannya gagal untuk bilangan besar.

Kode 9.30 menceritakan kisah yang sama dalam Python.

Kode 9.30 `my_collection.py`: kelima bagian dalam Python; fungsi pembanding bergaya Java yang diadaptasi dengan `functools.cmp_to_key`, dan fungsi `key=` yang Pythonic.

```
"""my_collection.py: iterator, duplikat, iterable, dan comparator di Python."""
from functools import cmp_to_key


def remove_duplicates_from_sorted_list(items):
    result = []
    previous = None
    for current in items:
        if not result or previous != current:
            result.append(current)
        previous = current
    return result


class Student:
    def __init__(self, age):
        self._age = age

    def get_age(self):
        return self._age


def age_compare(s1, s2):
    """Fungsi pembanding bergaya Comparator.compare di Java."""
    if s1 is s2:
        return 0
    if s1 is None:
        return -1                       # None datang lebih dulu
    if s2 is None:
        return 1
    return s1.get_age() - s2.get_age()     # yang lebih muda dulu


def main():
    print("Part 1: Iterator")
    my_collection = ["cat"]
    iterator = iter(my_collection)
    for element in iterator:
        print(element)

    print("\nPart 2: Iterator is only read not change")
    for element in my_collection:
        element += " and dog"           # hanya mengikat ulang variabel perulangan
        print(element)
    print("collection still holds", my_collection)
```

Kode 9.30 (Lanjutan) *my_collection.py: kelima bagian dalam Python; fungsi pembanding bergaya Java yang diadaptasi dengan functools.cmp_to_key, dan fungsi key= yang Pythonic.*

```
    print("\nPart 3: Removing duplicate values (sorted list only)")
    my_collection += ["cat", "dog", "monkey", "monkey"]
    for element in my_collection:
        print(element)
    print("\nAfter removing the duplicate values")
    for element in remove_duplicates_from_sorted_list(my_collection):
        print(element)

    print("\nPart 4: Iterable")
    it = [3, 5, 7]
    total = 0
    for i in it:                        # berlaku untuk apa pun yang iterable
        total += i
    print("sum =", total)

    print("\nPart 5: Comparator")
    s, s1, s2 = Student(15), Student(20), Student(12)
    students = [s, s1, s2]
    for st in students:
        print(st.get_age())
    print(f"Age difference between s1 (age {s1.get_age()}) and s2 (age {s2.get_age()}) "
          f"= {age_compare(s1, s2)}")
    print("After sort is called, the list of student becomes:")
    students.sort(key=cmp_to_key(age_compare))     # comparator gaya lama
    for st in students:
        print(st.get_age())
    print("with key=:", [st.get_age() for st in sorted(students, key=Student.get_age)])


if __name__ == "__main__":
    main()
```

```
Part 1: Iterator
cat

Part 2: Iterator is only read not change
cat and dog
collection still holds ['cat']

Part 3: Removing duplicate values (sorted list only)
cat
cat
dog
monkey
monkey

After removing the duplicate values
cat
dog
monkey

Part 4: Iterable
sum = 15

Part 5: Comparator
15
20
12
Age difference between s1 (age 20) and s2 (age 12) = 8
After sort is called, the list of student becomes:
12
15
20
with key=: [12, 15, 20]
```

Bagian 2 berperilaku persis seperti di Java, dengan alasan yang sama: `element += " and dog"` mengikat ulang variabel perulangan, dan list-nya tidak berubah. Bagian 5 memuat perbedaan terpenting dalam seluruh subbab ini. Pengurutan Python tidak menerima fungsi pembanding berargumen dua; ia menerima **fungsi kunci** (*key function*) berargumen *satu*, dan mengurutkan elemen-elemen menurut nilai yang dikembalikan kunci itu. `sorted(students, key=Student.get_age)` berarti "urutkan menurut umur". Comparator bergaya Java masih dapat dipakai dengan membungkusnya, `key=cmp_to_key(age_compare)`, yang berguna saat memindahkan kode, tetapi kunci lebih mudah ditulis, lebih sulit disalahgunakan, dan lebih cepat, karena kunci setiap elemen dihitung sekali saja, bukan sekali per perbandingan.

9.9.2 Comparator untuk pemakaian sungguhan

Sejak Java 8, comparator jarang ditulis sebagai kelas. Antarmuka `Comparator` punya metode statis dan metode *default* yang *membangun* comparator dari field yang ingin dijadikan dasar pengurutan, lalu memadukannya. **Kode 9.31** mengurutkan lima pelanggan binatu dengan beberapa cara.

Kode 9.31 `SortingDemo.java`: urutan alami, `comparingInt(...).reversed()`, `comparing(...).thenComparing(...)`, comparator lambda, `TreeSet` dengan comparator, dan limpahan pada comparator pengurangan.

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.List;
import java.util.TreeSet;

public class SortingDemo {
    public static void main(String[] args) {
        List<Customer> customers = new ArrayList<>(List.of(
                new Customer("Rina", "Sidoarjo", 4),
                new Customer("Agus", "Surabaya", 7),
                new Customer("Sari", "Surabaya", 4),
                new Customer("Budi", "Gresik", 9),
                new Customer("Dewi", "Gresik", 4)));

        customers.sort(null);                           // urutan alami: compareTo
        System.out.println("by name      " + customers);

        customers.sort(Comparator.comparingInt(Customer::getVisits).reversed());
        System.out.println("most visits  " + customers);   // yang seri tetap berurutan
nama

        customers.sort(Comparator.comparing(Customer::getCity)
                .thenComparing(Customer::getName, Comparator.reverseOrder()));
        System.out.println("by city, then name Z-A:");
        for (Customer c : customers) {
            System.out.printf("  %-9s %s%n", c.getCity(), c);
        }

        // Comparator ditulis sebagai lambda: (a, b) -> int negatif, nol, atau positif.
        Comparator<Customer> byVisits = (a, b) -> Integer.compare(a.getVisits(),
b.getVisits());
        System.out.println("fewest visits " + Collections.min(customers, byVisits));

        // TreeSet dengan Comparator: "sama" menurut comparator berarti duplikat!
        TreeSet<Customer> oneAtEachCount = new TreeSet<>(
                Comparator.comparingInt(Customer::getVisits));
        oneAtEachCount.addAll(customers);
        System.out.println("TreeSet by visits " + oneAtEachCount);
```

Kode 9.31 (Lanjutan) *SortingDemo.java: urutan alami, comparingInt(...).reversed(), comparing(...).thenComparing(...), comparator lambda, TreeSet dengan comparator, dan limpahan pada comparator pengurangan.*

```
        // Mengapa pengurangan adalah comparator yang berbahaya.
        Comparator<Integer> bySubtraction = (a, b) -> a - b;
        int low = -2_000_000_000;
        int high = 2_000_000_000;
        System.out.println("subtraction says low vs high = " + bySubtraction.compare(low,
high)
                + ", Integer.compare says " + Integer.compare(low, high));
    }
}

class Customer implements Comparable<Customer> {
    private final String name;
    private final String city;
    private final int visits;

    Customer(String name, String city, int visits) {
        this.name = name;
        this.city = city;
        this.visits = visits;
    }

    String getName() {
        return name;
    }

    String getCity() {
        return city;
    }

    int getVisits() {
        return visits;
    }

    @Override
    public int compareTo(Customer other) {           // urutan alami: menurut nama
        return name.compareTo(other.name);
    }

    @Override
    public String toString() {
        return name + "(" + visits + ")";
    }
}
```

```
by name      [Agus(7), Budi(9), Dewi(4), Rina(4), Sari(4)]
most visits  [Budi(9), Agus(7), Dewi(4), Rina(4), Sari(4)]
by city, then name Z-A:
  Gresik    Dewi(4)
  Gresik    Budi(9)
  Sidoarjo  Rina(4)
  Surabaya  Sari(4)
  Surabaya  Agus(7)
fewest visits Dewi(4)
TreeSet by visits [Dewi(4), Agus(7), Budi(9)]
subtraction says low vs high = 294967296, Integer.compare says -1
```

`customers.sort(null)` mengurutkan menurut urutan alami, `compareTo`, yang membandingkan nama. `Comparator.comparingInt(Customer::getVisits)` membangun comparator yang membandingkan jumlah kunjungan, dan `.reversed()` membaliknya, sehingga pelanggan yang paling sering datang berada di depan. Sekarang perhatikan tiga pelanggan dengan empat kunjungan: `Dewi`, `Rina`, `Sari`, masih berurutan nama. Itu bukan kebetulan. `List.sort` milik Java

untuk objek bersifat **stabil** (*stable*): elemen-elemen yang dibandingkan sama mempertahankan urutan relatif sebelumnya. Kestabilan itulah yang memungkinkan "urutkan menurut kunjungan, dan di antara kunjungan yang sama pertahankan urutan abjad" dengan satu pengurutan sesudah pengurutan sebelumnya [14].

`Comparator.comparing(Customer::getCity).thenComparing(Customer::getName, Comparator.reverseOrder())` mengurutkan menurut kota dan, di dalam satu kota, menurut nama secara *menurun*: Gresik sebelum Sidoarjo sebelum Surabaya, dan di dalam Gresik `Dewi` sebelum `Budi`. Lambda `(a, b) -> Integer.compare(a.getVisits(), b.getVisits())` adalah comparator yang ditulis tangan; `Collections.min` mengembalikan pelanggan pertama dengan kunjungan paling sedikit.

Baris `TreeSet` memuat sebuah jebakan. `TreeSet` yang dibangun dengan comparator memakai *comparator itu* untuk menentukan kesamaan, bukan `equals`. Tiga pelanggan punya empat kunjungan, sehingga comparator menyebut mereka sama, dan himpunan itu hanya menyimpan yang pertama. Hanya tiga pelanggan yang bertahan: `[Dewi(4), Agus(7), Budi(9)]`. Pakailah comparator dengan himpunan atau map terurut hanya bila "sama menurut comparator" benar-benar berarti "elemen yang sama" — biasanya dengan menambahkan pemecah seri seperti `.thenComparing(Customer::getName)`.

Baris terakhir memperlihatkan mengapa pengurangan pada `AgeComparator` berbahaya secara umum. Untuk $a = -2{,}000{,}000{,}000$ dan $b = 2{,}000{,}000{,}000$, selisih sebenarnya $a - b = -4 \times 10^9$ tidak muat di dalam `int` (yang jangkauannya sekitar $\pm 2.1 \times 10^9$). Nilainya melingkar menjadi $-4{,}000{,}000{,}000 + 2^{32} = 294{,}967{,}296$, bilangan *positif*, sehingga comparator mengklaim bahwa -2 miliar lebih besar daripada $+2$ miliar. `Integer.compare(a, b)` membandingkan tanpa mengurangkan dan selalu mengembalikan tanda yang benar. Pakailah itu, atau `Comparator.comparingInt`, selalu.

Kode 9.32 adalah versi Python-nya.

Kode 9.32 `sorting_demo.py`: urutan alami dengan `__lt__` dan `total_ordering`, fungsi kunci, dua pengurutan stabil untuk arah campuran, dan bilangan bulat Python yang tak terbatas.

```
"""sorting_demo.py: pengurutan dengan fungsi kunci, dan mengapa pengurutan Python
stabil."""
from functools import total_ordering
from operator import attrgetter


@total_ordering
class Customer:
    def __init__(self, name, city, visits):
        self.name = name
        self.city = city
        self.visits = visits

    def __eq__(self, other):
        return isinstance(other, Customer) and self.name == other.name

    def __hash__(self):
        return hash(self.name)

    def __lt__(self, other):           # urutan alami: menurut nama
        return self.name < other.name

    def __repr__(self):
        return f"{self.name}({self.visits})"


def main():
    customers = [Customer("Rina", "Sidoarjo", 4), Customer("Agus", "Surabaya", 7),
                 Customer("Sari", "Surabaya", 4), Customer("Budi", "Gresik", 9),
                 Customer("Dewi", "Gresik", 4)]

    customers.sort()                                    # urutan alami: __lt__
    print("by name     ", customers)

    customers.sort(key=attrgetter("visits"), reverse=True)
    print("most visits ", customers)                    # yang seri tetap berurutan nama

    # Dua pengurutan stabil: kunci minor dulu, lalu kunci mayor.
    customers.sort(key=lambda c: c.name, reverse=True)
    customers.sort(key=lambda c: c.city)
    print("by city, then name Z-A:")
    for c in customers:
        print(f"  {c.city:<9} {c}")

    print("fewest visits", min(customers, key=lambda c: c.visits))
```

Kode 9.32 (Lanjutan) *sorting_demo.py: urutan alami dengan __lt__ dan total_ordering, fungsi kunci, dua pengurutan stabil untuk arah campuran, dan bilangan bulat Python yang tak terbatas.*

```
    # "Satu pelanggan per jumlah kunjungan", seperti TreeSet dengan Comparator.
    one_at_each_count = {}
    for c in customers:
        one_at_each_count.setdefault(c.visits, c)      # simpan yang pertama terlihat
    print("one per count", [one_at_each_count[k] for k in sorted(one_at_each_count)])

    low, high = -2_000_000_000, 2_000_000_000
    print("subtraction says low vs high =", low - high, "(Python ints never overflow)")


if __name__ == "__main__":
    main()
```

```
by name      [Agus(7), Budi(9), Dewi(4), Rina(4), Sari(4)]
most visits  [Budi(9), Agus(7), Dewi(4), Rina(4), Sari(4)]
by city, then name Z-A:
  Gresik    Dewi(4)
  Gresik    Budi(9)
  Sidoarjo  Rina(4)
  Surabaya  Sari(4)
  Surabaya  Agus(7)
fewest visits Dewi(4)
one per count [Dewi(4), Agus(7), Budi(9)]
subtraction says low vs high = -4000000000 (Python ints never overflow)
```

Urutan alami Python didefinisikan oleh `__lt__`; dekorator `@total_ordering` melengkapi `<=`, `>`, dan `>=` dari `__lt__` dan `__eq__`. `sort(key=attrgetter("visits"), reverse=True)` mengurutkan menurut kunjungan, terbesar dulu, dan pengurutan Python (Timsort) juga stabil, sehingga yang seri tetap berurutan nama [14]. Untuk arah campuran pada *string* — kota menaik, nama menurun — tidak ada bilangan yang dapat dinegatifkan. Penyelesaian Pythonic memanfaatkan kestabilan: urutkan dulu menurut kunci *minor* (nama, menurun), lalu menurut kunci *mayor* (kota, menaik). Pengurutan kedua mempertahankan urutan hasil pengurutan pertama di antara pelanggan sekota. Kamus yang dibangun dengan `setdefault` memainkan peran `TreeSet` dengan comparator — satu pelanggan per jumlah kunjungan, yang pertama terlihat — tetapi membuat aturan "satu per jumlah" terlihat jelas alih-alih menyembunyikannya di dalam comparator. Dan bilangan bulat Python tidak pernah melimpah, sehingga `low - high` cukup bernilai −4,000,000,000.

9.10 Memilih Koleksi yang Tepat

Kita kini mengenal selusin koleksi. Memilih di antaranya adalah soal menghitung berapa banyak pekerjaan yang dibutuhkan setiap operasi ketika koleksinya tumbuh, dalam notasi O yang dipakai di seluruh buku ini: $O(1)$ berarti banyaknya langkah terbatas berapa pun ukurannya, $O(\log n)$ banyak langkah yang bertambah satu setiap kali n berlipat dua, $O(n)$ banyak langkah yang sebanding dengan n. **Tabel 9.5** memberi biayanya untuk Java [5], [9].

Tabel 9.5 Biaya operasi-operasi utama kelas koleksi Java untuk n elemen.

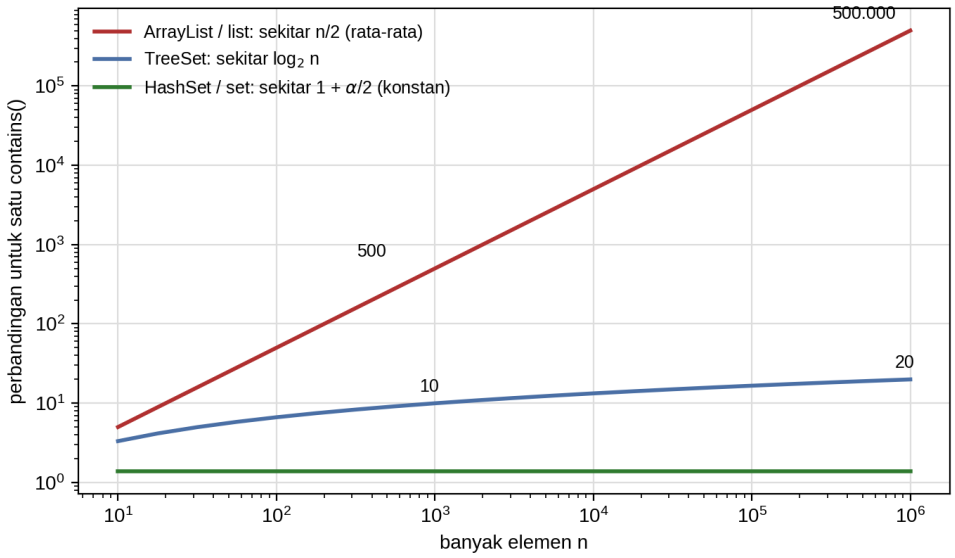
Kelas	add / put	remove(x)	contains / get(key)	get(i)	Urutan saat diiterasi
`ArrayList`	$O(1)$ di ujung (teramortisasi); $O(n)$ di tempat lain	$O(n)$	$O(n)$	$O(1)$	posisi
`LinkedList`	$O(1)$ di ujung; $O(n)$ untuk mencapai tengah	$O(n)$	$O(n)$	$O(n)$	posisi
`ArrayDeque`	$O(1)$ di kedua ujung	$O(1)$ di ujung	$O(n)$	—	depan ke belakang
`HashSet`, `HashMap`	$O(1)$ harapan	$O(1)$ harapan	$O(1)$ harapan	—	tidak ditentukan
`LinkedHashSet`, `LinkedHashMap`	$O(1)$ harapan	$O(1)$ harapan	$O(1)$ harapan	—	penyisipan
`TreeSet`, `TreeMap`	$O(\log n)$	$O(\log n)$	$O(\log n)$	—	terurut
`PriorityQueue`	$O(\log n)$	`poll`: $O(\log n)$; lainnya: $O(n)$	$O(n)$; `peek` $O(1)$	—	larik heap

Tabel 9.6 memberi hal yang sama untuk Python [7].

Tabel 9.6 Biaya operasi-operasi utama koleksi Python untuk n elemen.

Tipe	Tambah	Hapus	`x in c` / `d[k]`	`c[i]`	Catatan
`list`	`append` $O(1)$ teramortisasi; `insert(0, x)` $O(n)$	`pop()` $O(1)$; `pop(0)`, `remove(x)` $O(n)$	$O(n)$	$O(1)$	`ArrayList` Java
`tuple`	—	—	$O(n)$	$O(1)$	imutabel, dapat di-hash bila elemennya dapat
`collections.deque`	$O(1)$ di kedua ujung	$O(1)$ di ujung	$O(n)$	$O(n)$ di tengah	`ArrayDeque` Java
`set`, `frozenset`	$O(1)$ harapan	$O(1)$ harapan	$O(1)$ harapan	—	urutan berubah-ubah untuk string
`dict`	$O(1)$ harapan	$O(1)$ harapan	$O(1)$ harapan	—	urutan penyisipan terjaga
`heapq` pada `list`	`heappush` $O(\log n)$	`heappop` $O(\log n)$	$O(n)$	`h[0]` adalah minimum	`PriorityQueue` Java
`list` terurut + `bisect`	`insort` $O(n)$ (pencarian $O(\log n)$)	$O(n)$	$O(\log n)$ dengan `bisect`	$O(1)$	`TreeSet` Java untuk data yang lebih sering dibaca

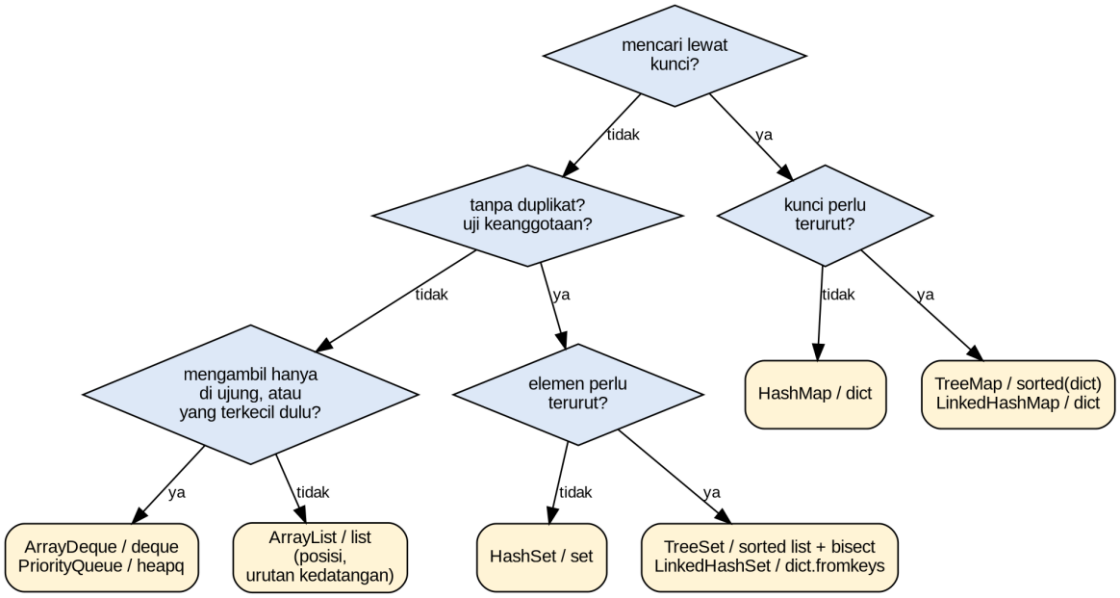
Perbedaan di antara kelas-kelas ini bukan sekadar teori. **Gambar 9.9** memplot banyaknya perbandingan yang dilakukan satu `contains` ketika koleksinya tumbuh dari sepuluh sampai sejuta elemen, pada sumbu logaritmik.



Gambar 9.9 Perbandingan yang dilakukan satu `contains` ketika n tumbuh (kedua sumbu logaritmik): sekitar $n/2$ untuk list, sekitar $\log_2 n$ untuk pohon seimbang, dan konstanta kecil untuk tabel hash dengan faktor muatan $\alpha \leq 0.75$.

Pada seribu elemen, list memerlukan sekitar 500 perbandingan, pohon sekitar 10, tabel hash satu atau dua. Pada sejuta, list memerlukan setengah juta, pohon 20, tabel hash tetap satu atau dua. Seandainya buku pesanan binatu menjawab “apakah L-217 ada di sistem?” dengan sebuah list, setiap pertanyaan atas sejuta pesanan akan semahal 25.000 pertanyaan yang dijawab himpunan pohon, atau beberapa ratus ribu pertanyaan yang dijawab himpunan hash.

Angka-angka seperti inilah alasan di balik kaidah praktis pada **Gambar 9.10**: ajukan empat pertanyaan secara berurutan — mencari lewat kunci? tanpa duplikat? hanya di ujung atau terkecil dulu? — dan jawabannya menyebut nama koleksinya; bila semua jawaban "tidak", pakailah list.



Gambar 9.10 Memilih koleksi dengan empat pertanyaan. Setiap daun menyebut kelas Java dan tipe Python yang menjawabnya.

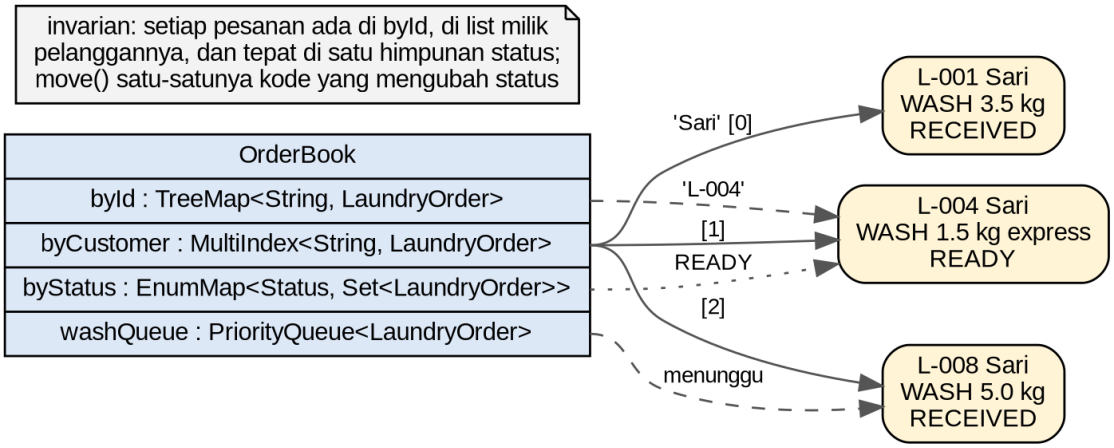
Ukurlah sebelum mengoptimalkan: untuk beberapa lusin elemen setiap koleksi cepat, dan kode yang paling jelaslah yang menang. Dan ketika sebuah program memerlukan *beberapa* jawaban ini untuk data yang sama — lewat nomor *dan* lewat pelanggan *dan* lewat status — penyelesaiannya adalah beberapa koleksi yang merujuk ke objek-objek yang sama, yang dijaga tetap sepakat oleh satu kelas. Itulah rancangan subbab berikut.

9.11 Profitina Laundry: Buku Pesanan dengan Beberapa Indeks

Pada awal bab, pemilik binatu mengajukan empat pertanyaan: *Di mana pesanan L-217? Apa saja yang dititipkan Bu Sari? Pesanan mana yang siap? Kantong mana yang masuk ke mesin berikutnya yang kosong?* Setiap pertanyaan kini punya koleksi yang menjawabnya dengan cepat:

- **cari lewat nomor pesanan**: map dari id ke pesanan, $O(\log n)$ dengan `TreeMap`, yang juga mendaftar pesanan menurut urutan id;
- **pesanan milik seorang pelanggan**: map dari nama pelanggan ke list pesanan pelanggan itu;
- **pesanan dalam suatu status**: satu himpunan per status — `RECEIVED`, `WASHING`, `READY`, `PICKED_UP` — di dalam sebuah `EnumMap`, map khusus untuk kunci enum yang menyimpan nilai-nilainya dalam larik berindeks posisi konstanta;
- **cucian berikutnya**: antrean prioritas yang menaruh pesanan kilat sebelum pesanan biasa dan, di dalam tiap kelompok, yang datang lebih awal lebih dulu.

Gambar 9.11 memperlihatkan rancangannya dengan tiga pesanan milik Sari. Hal pentingnya adalah bahwa keempat struktur itu tidak memegang *salinan* pesanan. Mereka memegang referensi ke objek-objek `LaundryOrder` yang *sama*, sehingga pesanan yang diubah melalui satu indeks berubah bagi semuanya.



Gambar 9.11 Buku pesanan: empat struktur, satu kumpulan objek `LaundryOrder`. Kelas ini menjaga sebuah invarian: setiap pesanan muncul di `byId`, di list milik pelanggannya, dan tepat di satu himpunan status.

Beberapa indeks membawa kewajiban baru: mereka tidak boleh berselisih. Bila sebuah pesanan menjadi `READY` tetapi masih berada di himpunan `WASHING`, jawaban atas "pesanan mana yang siap?" salah, diam-diam. Karena itu kelas ini menjaga sebuah **invarian**, pernyataan yang benar setelah setiap metode publik, dan memeriksanya dalam metode tersendiri:

$$\sum_{s \in Status} |byStatus[s]| = |byId| \quad \text{dan} \quad o \in byStatus[o.status] \text{ untuk setiap pesanan } o \tag{9.8}$$

Paruh pertama Persamaan (9.8) menyatakan bahwa himpunan-himpunan status bersama-sama memuat tepat sebanyak pesanan di dalam buku; paruh kedua menyatakan bahwa setiap pesanan berada di himpunan statusnya sendiri. Bersama-sama keduanya menyatakan bahwa setiap pesanan berada tepat di satu himpunan status. Rancangannya membuat invarian itu mudah dijaga: hanya satu metode privat, `move`, yang pernah mengubah status pesanan, dan metode itu memperbarui indeks dalam tarikan napas yang sama.

Harga mengikuti aturan Bab 5: WASH Rp7,000, IRON Rp5,000, dan WASH_IRON Rp10,000 per kilogram, dan pesanan kilat berharga separuhnya lagi. Untuk berat w dan tarif r,

$$p = \text{round}(w \cdot r) + e \cdot \left\lfloor \frac{\text{round}(w \cdot r)}{2} \right\rfloor, \qquad e = \begin{cases} 1 & \text{kilat} \\ 0 & \text{lainnya} \end{cases} \tag{9.9}$$

sehingga cucian kilat 1,5 kg milik Sari berharga $10{,}500 + 5{,}250 = 15{,}750$. Pesanan setrika saja tidak memerlukan mesin: pesanan itu tidak pernah masuk antrean cucian dan boleh langsung berpindah dari `RECEIVED` ke `READY`. **Kode 9.33** adalah seluruh programnya.

Kode 9.33 `LaundryOrderBook.java`: buku pesanan Profitina Laundry dengan `TreeMap`, `MultiIndex` generik, `EnumMap` berisi `LinkedHashSet`, dan `PriorityQueue`, ditambah laporan pendapatan yang diurutkan dengan comparator.

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.EnumMap;
import java.util.LinkedHashSet;
import java.util.List;
import java.util.Locale;
import java.util.Map;
import java.util.PriorityQueue;
import java.util.Set;
import java.util.TreeMap;

public class LaundryOrderBook {
    public static void main(String[] args) {
        OrderBook book = new OrderBook();
        book.receive(new LaundryOrder("L-001", "Sari", "WASH", 3.5, false));
        book.receive(new LaundryOrder("L-002", "Budi", "WASH_IRON", 5.0, true));
        book.receive(new LaundryOrder("L-003", "Rina", "IRON", 2.0, false));
        book.receive(new LaundryOrder("L-004", "Sari", "WASH", 1.5, true));
        book.receive(new LaundryOrder("L-005", "Agus", "WASH", 6.0, false));
        book.receive(new LaundryOrder("L-006", "Budi", "WASH", 4.0, false));
        book.receive(new LaundryOrder("L-007", "Dewi", "WASH_IRON", 2.5, true));
        book.receive(new LaundryOrder("L-008", "Sari", "WASH", 5.0, false));
        System.out.println("wash queue: " + book.washQueueInOrder());

        for (int i = 0; i < 3; i++) {
            System.out.println("start wash " + book.startNextWash().getId());
        }
        book.markReady("L-002");
        book.markReady("L-004");
        book.markReady("L-003");                // hanya setrika: tidak perlu dicuci
        book.pickUp("L-002");
        try {
            book.pickUp("L-005");
        } catch (IllegalStateException e) {
            System.out.println("refused: " + e.getMessage());
        }
        try {
            book.receive(new LaundryOrder("L-001", "Joko", "IRON", 1.0, false));
        } catch (IllegalArgumentException e) {
            System.out.println("refused: " + e.getMessage());
        }
```

Kode 9.33 (Lanjutan) *LaundryOrderBook.java: buku pesanan Profitina Laundry dengan TreeMap, MultiIndex generik, EnumMap berisi LinkedHashSet, dan PriorityQueue, ditambah laporan pendapatan yang diurutkan dengan comparator.*

```
        System.out.println("status counts " + book.statusCounts());
        System.out.println("Sari's orders " + book.ordersOf("Sari"));
        System.out.println("READY        " + book.withStatus(Status.READY));
        System.out.println("heaviest 3   " + book.heaviest(3));
        System.out.println("invariant holds: " + book.invariantHolds());
        book.printRevenueReport();
    }
}

enum Status { RECEIVED, WASHING, READY, PICKED_UP }

final class LaundryOrder {
    private static final Map<String, Integer> RATES = Map.of(
            "WASH", 7000, "IRON", 5000, "WASH_IRON", 10000);
    private static int arrivals = 0;          // memberi setiap pesanan nomor kedatangan

    private final String id;
    private final String customer;
    private final String service;
    private final double weightKg;
    private final boolean express;
    private final int arrival;
    private Status status = Status.RECEIVED;  // satu-satunya field yang berubah

    LaundryOrder(String id, String customer, String service, double weightKg, boolean
express) {
        if (!RATES.containsKey(service)) {
            throw new IllegalArgumentException("unknown service " + service);
        }
        this.id = id;
        this.customer = customer;
        this.service = service;
        this.weightKg = weightKg;
        this.express = express;
        this.arrival = ++arrivals;
    }

    String getId() { return id; }
    String getCustomer() { return customer; }
    String getService() { return service; }
    double getWeightKg() { return weightKg; }
    boolean isExpress() { return express; }
    int getArrival() { return arrival; }
    Status getStatus() { return status; }
    void setStatus(Status s) { status = s; }   // package-private: hanya OrderBook yang
memanggilnya
```

Kode 9.33 (Lanjutan) *LaundryOrderBook.java: buku pesanan Profitina Laundry dengan TreeMap, MultiIndex generik, EnumMap berisi LinkedHashSet, dan PriorityQueue, ditambah laporan pendapatan yang diurutkan dengan comparator.*

```
    boolean needsWash() {
        return !service.equals("IRON");
    }

    long price() {                              // aturan Bab 5: kilat menambah separuh
        long base = Math.round(weightKg * RATES.get(service));
        return express ? base + base / 2 : base;
    }

    @Override
    public boolean equals(Object o) {           // identitas pesanan = id-nya
        return o instanceof LaundryOrder other && id.equals(other.id);
    }

    @Override
    public int hashCode() {
        return id.hashCode();
    }

    @Override
    public String toString() {
        return id;
    }
}

// Indeks generik: satu kunci, banyak nilai, kunci disimpan terurut.
final class MultiIndex<K extends Comparable<K>, V> {
    private final Map<K, List<V>> map = new TreeMap<>();

    void add(K key, V value) {
        map.computeIfAbsent(key, k -> new ArrayList<>()).add(value);
    }

    List<V> get(K key) {
        return Collections.unmodifiableList(map.getOrDefault(key, List.of()));
    }

    Set<K> keys() {
        return Collections.unmodifiableSet(map.keySet());
    }
}
```

Kode 9.33 (Lanjutan) *LaundryOrderBook.java: buku pesanan Profitina Laundry dengan TreeMap, MultiIndex generik, EnumMap berisi LinkedHashSet, dan PriorityQueue, ditambah laporan pendapatan yang diurutkan dengan comparator.*

```
final class OrderBook {
    private final Map<String, LaundryOrder> byId = new TreeMap<>();
    private final MultiIndex<String, LaundryOrder> byCustomer = new MultiIndex<>();
    private final Map<Status, Set<LaundryOrder>> byStatus = new EnumMap<>(Status.class);
    // Kilat dulu; di dalam tiap kelompok, yang datang dulu dilayani dulu.
    private final PriorityQueue<LaundryOrder> washQueue = new PriorityQueue<>(
            Comparator.comparing((LaundryOrder o) -> !o.isExpress())
                      .thenComparingInt(LaundryOrder::getArrival));

    OrderBook() {
        for (Status s : Status.values()) {
            byStatus.put(s, new LinkedHashSet<>());
        }
    }

    void receive(LaundryOrder order) {
        if (byId.containsKey(order.getId())) {
            throw new IllegalArgumentException("duplicate id " + order.getId());
        }
        byId.put(order.getId(), order);
        byCustomer.add(order.getCustomer(), order);
        byStatus.get(Status.RECEIVED).add(order);
        if (order.needsWash()) {
            washQueue.offer(order);
        }
    }

    LaundryOrder startNextWash() {
        LaundryOrder next = washQueue.poll();
        if (next == null) {
            throw new IllegalStateException("no order is waiting to be washed");
        }
        move(next, Status.WASHING);
        return next;
    }

    void markReady(String id) {
        LaundryOrder o = find(id);
        boolean allowed = o.getStatus() == Status.WASHING
                || (o.getStatus() == Status.RECEIVED && !o.needsWash());
        if (!allowed) {
            throw new IllegalStateException(id + " cannot become READY from " +
o.getStatus());
        }
        move(o, Status.READY);
    }
```

Kode 9.33 (Lanjutan) *LaundryOrderBook.java: buku pesanan Profitina Laundry dengan TreeMap, MultiIndex generik, EnumMap berisi LinkedHashSet, dan PriorityQueue, ditambah laporan pendapatan yang diurutkan dengan comparator.*

```
    void pickUp(String id) {
        LaundryOrder o = find(id);
        if (o.getStatus() != Status.READY) {
            throw new IllegalStateException(id + " is " + o.getStatus() + ", not READY");
        }
        move(o, Status.PICKED_UP);
    }

    // Satu-satunya tempat indeks status berubah: hapus, ubah, tambahkan kembali.
    private void move(LaundryOrder o, Status to) {
        byStatus.get(o.getStatus()).remove(o);
        o.setStatus(to);
        byStatus.get(to).add(o);
    }

    private LaundryOrder find(String id) {
        LaundryOrder o = byId.get(id);
        if (o == null) {
            throw new IllegalArgumentException("no order " + id);
        }
        return o;
    }

    List<LaundryOrder> ordersOf(String customer) {
        return byCustomer.get(customer);
    }

    List<LaundryOrder> withStatus(Status s) {
        return new ArrayList<>(byStatus.get(s));
    }

    Map<Status, Integer> statusCounts() {
        Map<Status, Integer> counts = new EnumMap<>(Status.class);
        for (Map.Entry<Status, Set<LaundryOrder>> e : byStatus.entrySet()) {
            counts.put(e.getKey(), e.getValue().size());
        }
        return counts;
    }
```

Kode 9.33 (Lanjutan) *LaundryOrderBook.java: buku pesanan Profitina Laundry dengan TreeMap, MultiIndex generik, EnumMap berisi LinkedHashSet, dan PriorityQueue, ditambah laporan pendapatan yang diurutkan dengan comparator.*

```
    List<LaundryOrder> washQueueInOrder() {
        PriorityQueue<LaundryOrder> copy = new PriorityQueue<>(washQueue);
        List<LaundryOrder> result = new ArrayList<>();
        while (!copy.isEmpty()) {
            result.add(copy.poll());
        }
        return result;
    }

    List<LaundryOrder> heaviest(int k) {
        List<LaundryOrder> all = new ArrayList<>(byId.values());
        all.sort(Comparator.comparingDouble(LaundryOrder::getWeightKg).reversed()
                .thenComparing(LaundryOrder::getId));
        return all.subList(0, Math.min(k, all.size()));
    }

    // Setiap pesanan berada tepat di satu himpunan status.
    boolean invariantHolds() {
        int total = 0;
        for (Set<LaundryOrder> set : byStatus.values()) {
            total += set.size();
        }
        for (LaundryOrder o : byId.values()) {
            if (!byStatus.get(o.getStatus()).contains(o)) {
                return false;
            }
        }
        return total == byId.size();
    }

    void printRevenueReport() {
        Map<String, Long> perCustomer = new TreeMap<>();
        Map<String, long[]> perService = new TreeMap<>();    // {orders, rupiah}
        long total = 0;
        for (LaundryOrder o : byId.values()) {
            perCustomer.merge(o.getCustomer(), o.price(), Long::sum);
            long[] cell = perService.computeIfAbsent(o.getService(), k -> new long[2]);
            cell[0]++;
            cell[1] += o.price();
            total += o.price();
        }
        List<Map.Entry<String, Long>> ranking = new ArrayList<>(perCustomer.entrySet());
        ranking.sort(Map.Entry.<String, Long>comparingByValue().reversed()
                .thenComparing(Map.Entry.comparingByKey()));
```

Kode 9.33 (Lanjutan) *LaundryOrderBook.java: buku pesanan Profitina Laundry dengan TreeMap, MultiIndex generik, EnumMap berisi LinkedHashSet, dan PriorityQueue, ditambah laporan pendapatan yang diurutkan dengan comparator.*

```
        System.out.printf("%-22s %6s %12s%n", "REVENUE BY CUSTOMER", "orders", "amount");
        for (Map.Entry<String, Long> e : ranking) {
            System.out.printf("  %-20s %6d %12s%n", e.getKey(),
                    ordersOf(e.getKey()).size(), rupiah(e.getValue()));
        }
        System.out.printf("%-22s %6s %12s%n", "REVENUE BY SERVICE", "orders", "amount");
        for (Map.Entry<String, long[]> e : perService.entrySet()) {
            System.out.printf("  %-20s %6d %12s%n", e.getKey(),
                    e.getValue()[0], rupiah(e.getValue()[1]));
        }
        System.out.printf("  %-20s %6d %12s%n", "TOTAL", byId.size(), rupiah(total));
    }

    static String rupiah(long amount) {
        return String.format(Locale.US, "Rp%,d", amount);
    }
}
```

```
wash queue: [L-002, L-004, L-007, L-001, L-005, L-006, L-008]
start wash L-002
start wash L-004
start wash L-007
refused: L-005 is RECEIVED, not READY
refused: duplicate id L-001
status counts {RECEIVED=4, WASHING=1, READY=2, PICKED_UP=1}
Sari's orders [L-001, L-004, L-008]
READY        [L-004, L-003]
heaviest 3   [L-005, L-002, L-008]
invariant holds: true
REVENUE BY CUSTOMER    orders       amount
  Budi                      2    Rp103,000
  Sari                      3     Rp75,250
  Agus                      1     Rp42,000
  Dewi                      1     Rp37,500
  Rina                      1     Rp10,000
REVENUE BY SERVICE     orders       amount
  IRON                      1     Rp10,000
  WASH                      5    Rp145,250
  WASH_IRON                 2    Rp112,500
  TOTAL                     8    Rp267,750
```

Mari kita baca kelas demi kelas.

`Status` adalah enum dengan empat tahap kehidupan sebuah pesanan. Memakai enum alih-alih konstanta `int` (Subbab 9.2.1) berarti `byStatus` dapat berupa `EnumMap<Status, ...>` dan sebuah status tidak pernah dapat bernilai 7.

`LaundryOrder` memegang data sebuah pesanan. Semua field-nya `final` kecuali `status`, yang setter-nya package-private. Penghitung statis `arrivals` memberi nomor pesanan; antrean prioritas memakainya untuk memecah seri. `equals` dan `hashCode` hanya memakai id, dan itu sangat penting: bila hash-nya bergantung pada status, mengubah status akan mendamparkan pesanan di ember yang salah dalam `LinkedHashSet`-nya (Soal Latihan 9.8). `RATES` adalah `Map.of(...)`, map *imutabel*: `put` padanya melempar `UnsupportedOperationException`.

`MultiIndex<K extends Comparable<K>, V>` adalah kelas generik kecil buatan kita sendiri: satu kunci, banyak nilai, kunci disimpan dalam `TreeMap` (karena itulah ada batasnya). `add` memakai `computeIfAbsent(key, k -> new ArrayList<>())`, yang mengembalikan list milik kunci itu dan membuatnya saat kunci pertama kali dijumpai. `get` mengembalikan

`Collections.unmodifiableList(...)`, sebuah *view* baca-saja, sehingga tidak ada yang dapat merusak invarian melalui list yang kita serahkan; kunci yang tidak ada memberi `List.of()` yang kosong, bukan `null`. Karena generik, kelas ini tidak tahu apa-apa tentang binatu dan dapat mengindeks buku menurut judul tanpa perubahan.

`OrderBook` memiliki keempat struktur, semuanya `private final`. Konstruktornya memasukkan `LinkedHashSet` kosong ke dalam `EnumMap` untuk setiap status, sehingga `byStatus.get(s)` tidak pernah `null`. Comparator antreannya membandingkan `!o.isExpress()`; karena `false` terurut sebelum `true`, pesanan kilat berada di depan, dan `.thenComparingInt(LaundryOrder::getArrival)` memecah seri menurut kedatangan.

- `receive` menolak id kembar dengan eksepsi *sebelum* menyentuh struktur apa pun, sehingga pesanan yang ditolak tidak meninggalkan jejak. Lalu ia menambahkan pesanan ke setiap indeks — dan ke antrean cucian bila perlu dicuci.
- `startNextWash` mengambil dari antrean dan memindahkan pesanan ke `WASHING`.
- `markReady` dan `pickUp` memeriksa bahwa peralihannya diizinkan — pesanan baru boleh siap setelah dicuci, atau langsung bila hanya disetrika; pesanan baru boleh diambil bila sudah siap — dan melempar `IllegalStateException` bila tidak.
- `move` adalah satu-satunya tempat status berubah: hapus dari himpunan lama, ubah field-nya, tambahkan ke himpunan baru. Penghapusan harus terjadi *sebelum* field berubah, selagi pesanan masih dapat ditemukan di himpunan lamanya.
- `washQueueInOrder` mengambil dari *salinan* antrean; mencetak `PriorityQueue` itu sendiri akan menampilkan larik heap, bukan urutan pelayanan.
- `printRevenueReport` menjumlahkan harga per pelanggan dengan `merge` dan per layanan dengan `computeIfAbsent`, lalu memeringkat pelanggan dengan `Map.Entry.<String, Long>comparingByValue().reversed().thenComparing(Map.Entry.comparingByKey())`; `<String, Long>` yang eksplisit diperlukan karena `.reversed()` diterapkan sebelum kompilator sempat menyimpulkan tipenya.

Sekarang cocokkan keluarannya dengan `main`. Antrean cucian memuat tiga cucian kilat lebih dulu, `L-002, L-004, L-007`, lalu empat cucian biasa; `L-003` yang hanya disetrika tidak ada di dalamnya. Tiga cucian dimulai; `L-002` dan `L-004` selesai, `L-003` disetrika, dan `L-002` diambil. Buku menolak `L-005`, yang masih `RECEIVED`, dan menolak `L-001` yang kedua. Hitungan status, menurut urutan enum, berjumlah $4 + 1 + 2 + 1 = 8$, seperti dituntut Persamaan (9.8), dan `invariant holds` memeriksanya secara eksplisit. `READY` mendaftar `L-004` sebelum `L-003`, urutan keduanya menjadi siap, berkat `LinkedHashSet`.

Laporan pendapatan dapat diperiksa dengan tangan memakai Persamaan (9.9). Budi: $\text{round}(5.0 \times 10{,}000) = 50{,}000$, ditambah separuh untuk kilat, 75,000; dan $4.0 \times 7{,}000 = 28{,}000$; bersama-sama 103,000. Sari: $3.5 \times 7{,}000 = 24{,}500$, yang kilat 15,750, dan $5.0 \times 7{,}000 = 35{,}000$; bersama-sama 75,250. Agus 42,000, Dewi $25{,}000 + 12{,}500 = 37{,}500$, Rina $2.0 \times 5{,}000 = 10{,}000$. Totalnya 267,750, dan baris-baris per layanan berjumlah sama: $10{,}000 + 145{,}250 + 112{,}500 = 267{,}750$.

Kode 9.34 adalah versi Python-nya; keluarannya sama, kecuali cara Python menuliskan kamus dan nilai boolean.

Kode 9.34 `laundry_order_book.py`: buku pesanan yang sama dalam Python, dengan `MultiIndex` generik, kamus yang dipakai sebagai himpunan berurutan, dan `heapq` untuk antrean cucian.

```python
"""laundry_order_book.py: buku pesanan Profitina Laundry, diindeks dengan tiga cara."""
from __future__ import annotations

import heapq
import itertools
from enum import Enum
from typing import Any, Generic, Protocol, TypeVar


class Status(Enum):
    RECEIVED = 1
    WASHING = 2
    READY = 3
    PICKED_UP = 4


class LaundryOrder:
    RATES = {"WASH": 7000, "IRON": 5000, "WASH_IRON": 10000}
    _arrivals = itertools.count(1)            # memberi setiap pesanan nomor kedatangan

    def __init__(self, order_id: str, customer: str, service: str,
                 weight_kg: float, express: bool) -> None:
        if service not in self.RATES:
            raise ValueError(f"unknown service {service}")
        self.id = order_id
        self.customer = customer
        self.service = service
        self.weight_kg = weight_kg
        self.express = express
        self.arrival = next(self._arrivals)
        self.status = Status.RECEIVED         # satu-satunya atribut yang berubah

    def needs_wash(self) -> bool:
        return self.service != "IRON"

    def price(self) -> int:                   # aturan Bab 5: kilat menambah separuh
        base = round(self.weight_kg * self.RATES[self.service])
        return base + base // 2 if self.express else base

    def __eq__(self, other: object) -> bool:  # identitas pesanan = id-nya
        return isinstance(other, LaundryOrder) and self.id == other.id

    def __hash__(self) -> int:
        return hash(self.id)
```

Kode 9.34 (Lanjutan) *laundry_order_book.py: buku pesanan yang sama dalam Python, dengan MultiIndex generik, kamus yang dipakai sebagai himpunan berurutan, dan heapq untuk antrean cucian.*

```
    def __repr__(self) -> str:
        return self.id


class SupportsLessThan(Protocol):
    def __lt__(self, other: Any, /) -> bool: ...


K = TypeVar("K", bound=SupportsLessThan)
V = TypeVar("V")


class MultiIndex(Generic[K, V]):
    """Indeks generik: satu kunci, banyak nilai, kunci dilaporkan terurut."""

    def __init__(self) -> None:
        self._map: dict[K, list[V]] = {}

    def add(self, key: K, value: V) -> None:
        self._map.setdefault(key, []).append(value)

    def get(self, key: K) -> tuple[V, ...]:    # sebuah tuple: pemanggil tidak dapat
mengubahnya
        return tuple(self._map.get(key, ()))

    def keys(self) -> list[K]:
        return sorted(self._map)


class OrderBook:
    def __init__(self) -> None:
        self._by_id: dict[str, LaundryOrder] = {}
        self._by_customer: MultiIndex[str, LaundryOrder] = MultiIndex()
        # himpunan bergaya dict.fromkeys yang ingat urutan penyisipan: dict bernilai None
        self._by_status: dict[Status, dict[LaundryOrder, None]] = {s: {} for s in Status}
        # Kilat dulu; di dalam tiap kelompok, yang datang dulu dilayani dulu.
        self._wash_queue: list[tuple[bool, int, LaundryOrder]] = []

    def receive(self, order: LaundryOrder) -> None:
        if order.id in self._by_id:
            raise ValueError(f"duplicate id {order.id}")
        self._by_id[order.id] = order
        self._by_customer.add(order.customer, order)
        self._by_status[Status.RECEIVED][order] = None
        if order.needs_wash():
            heapq.heappush(self._wash_queue, (not order.express, order.arrival, order))
```

Kode 9.34 (Lanjutan) *laundry_order_book.py: buku pesanan yang sama dalam Python, dengan MultiIndex generik, kamus yang dipakai sebagai himpunan berurutan, dan heapq untuk antrean cucian.*

```
    def start_next_wash(self) -> LaundryOrder:
        if not self._wash_queue:
            raise RuntimeError("no order is waiting to be washed")
        _, _, order = heapq.heappop(self._wash_queue)
        self._move(order, Status.WASHING)
        return order

    def mark_ready(self, order_id: str) -> None:
        o = self._find(order_id)
        allowed = o.status is Status.WASHING or (
            o.status is Status.RECEIVED and not o.needs_wash())
        if not allowed:
            raise RuntimeError(f"{order_id} cannot become READY from {o.status.name}")
        self._move(o, Status.READY)

    def pick_up(self, order_id: str) -> None:
        o = self._find(order_id)
        if o.status is not Status.READY:
            raise RuntimeError(f"{order_id} is {o.status.name}, not READY")
        self._move(o, Status.PICKED_UP)

    def _move(self, o: LaundryOrder, to: Status) -> None:
        """Satu-satunya tempat indeks status berubah: hapus, ubah, tambahkan kembali."""
        del self._by_status[o.status][o]
        o.status = to
        self._by_status[to][o] = None

    def _find(self, order_id: str) -> LaundryOrder:
        try:
            return self._by_id[order_id]
        except KeyError:
            raise ValueError(f"no order {order_id}") from None

    def orders_of(self, customer: str) -> tuple[LaundryOrder, ...]:
        return self._by_customer.get(customer)

    def with_status(self, s: Status) -> list[LaundryOrder]:
        return list(self._by_status[s])

    def status_counts(self) -> dict[str, int]:
        return {s.name: len(orders) for s, orders in self._by_status.items()}

    def wash_queue_in_order(self) -> list[LaundryOrder]:
        return [entry[2] for entry in sorted(self._wash_queue)]
```

Kode 9.34 (Lanjutan) *laundry_order_book.py: buku pesanan yang sama dalam Python, dengan MultiIndex generik, kamus yang dipakai sebagai himpunan berurutan, dan heapq untuk antrean cucian.*

```
    def heaviest(self, k: int) -> list[LaundryOrder]:
        return sorted(self._by_id.values(), key=lambda o: (-o.weight_kg, o.id))[:k]

    def invariant_holds(self) -> bool:
        """Setiap pesanan berada tepat di satu himpunan status."""
        total = sum(len(orders) for orders in self._by_status.values())
        in_right_set = all(o in self._by_status[o.status] for o in self._by_id.values())
        return in_right_set and total == len(self._by_id)

    def print_revenue_report(self) -> None:
        per_customer: dict[str, int] = {}
        per_service: dict[str, list[int]] = {}            # [orders, rupiah]
        for o in self._by_id.values():
            per_customer[o.customer] = per_customer.get(o.customer, 0) + o.price()
            cell = per_service.setdefault(o.service, [0, 0])
            cell[0] += 1
            cell[1] += o.price()
        ranking = sorted(per_customer.items(), key=lambda kv: (-kv[1], kv[0]))

        print(f"{'REVENUE BY CUSTOMER':<22} {'orders':>6} {'amount':>12}")
        for name, amount in ranking:
            print(f"  {name:<20} {len(self.orders_of(name)):6d} {rupiah(amount):>12}")
        print(f"{'REVENUE BY SERVICE':<22} {'orders':>6} {'amount':>12}")
        for service in sorted(per_service):
            count, amount = per_service[service]
            print(f"  {service:<20} {count:6d} {rupiah(amount):>12}")
        total = sum(per_customer.values())
        print(f"  {'TOTAL':<20} {len(self._by_id):6d} {rupiah(total):>12}")


def rupiah(amount: int) -> str:
    return f"Rp{amount:,}"


def main() -> None:
    book = OrderBook()
    book.receive(LaundryOrder("L-001", "Sari", "WASH", 3.5, False))
    book.receive(LaundryOrder("L-002", "Budi", "WASH_IRON", 5.0, True))
    book.receive(LaundryOrder("L-003", "Rina", "IRON", 2.0, False))
    book.receive(LaundryOrder("L-004", "Sari", "WASH", 1.5, True))
    book.receive(LaundryOrder("L-005", "Agus", "WASH", 6.0, False))
    book.receive(LaundryOrder("L-006", "Budi", "WASH", 4.0, False))
    book.receive(LaundryOrder("L-007", "Dewi", "WASH_IRON", 2.5, True))
    book.receive(LaundryOrder("L-008", "Sari", "WASH", 5.0, False))
    print("wash queue:", book.wash_queue_in_order())
```

Kode 9.34 (Lanjutan) *laundry_order_book.py: buku pesanan yang sama dalam Python, dengan MultiIndex generik, kamus yang dipakai sebagai himpunan berurutan, dan heapq untuk antrean cucian.*

```
    for _ in range(3):
        print("start wash", book.start_next_wash().id)
    book.mark_ready("L-002")
    book.mark_ready("L-004")
    book.mark_ready("L-003")                # hanya setrika: tidak perlu dicuci
    book.pick_up("L-002")
    try:
        book.pick_up("L-005")
    except RuntimeError as e:
        print("refused:", e)
    try:
        book.receive(LaundryOrder("L-001", "Joko", "IRON", 1.0, False))
    except ValueError as e:
        print("refused:", e)

    print("status counts", book.status_counts())
    print("Sari's orders", list(book.orders_of("Sari")))
    print(f"READY        {book.with_status(Status.READY)}")
    print(f"heaviest 3   {book.heaviest(3)}")
    print("invariant holds:", book.invariant_holds())
    book.print_revenue_report()


if __name__ == "__main__":
    main()
```

```
wash queue: [L-002, L-004, L-007, L-001, L-005, L-006, L-008]
start wash L-002
start wash L-004
start wash L-007
refused: L-005 is RECEIVED, not READY
refused: duplicate id L-001
status counts {'RECEIVED': 4, 'WASHING': 1, 'READY': 2, 'PICKED_UP': 1}
Sari's orders [L-001, L-004, L-008]
READY        [L-004, L-003]
heaviest 3   [L-005, L-002, L-008]
invariant holds: True
REVENUE BY CUSTOMER    orders       amount
  Budi                      2    Rp103,000
  Sari                      3     Rp75,250
  Agus                      1     Rp42,000
  Dewi                      1     Rp37,500
  Rina                      1     Rp10,000
REVENUE BY SERVICE     orders       amount
  IRON                      1     Rp10,000
  WASH                      5    Rp145,250
  WASH_IRON                 2    Rp112,500
  TOTAL                     8    Rp267,750
```

Strukturnya sama kelas demi kelas; perbedaan yang menarik terletak pada cara setiap koleksi Java dituliskan.

- **Himpunan berurutan.** Python tidak punya `LinkedHashSet`, tetapi `dict` menjaga urutan penyisipan, sehingga kamus yang semua nilainya `None` adalah himpunan berurutan: `self._by_status[s][order] = None` menambah, `del self._by_status[s][order]` menghapus, dan `list(...)` memberi elemen-elemennya menurut urutan kedatangan. *Dictionary comprehension* `{s: {} for s in Status}` membuat satu untuk setiap status.
- **Antrean cucian.** `heapq` membandingkan item yang dimasukkan dengan `<`, dan `LaundryOrder` tidak punya `<`. Karena itu setiap entri berupa tuple `(not order.express, order.arrival, order)`: tuple dibandingkan elemen demi elemen, `False` terurut sebelum `True` sehingga pesanan

kilat di depan, dan nomor kedatangannya unik, sehingga perbandingan tidak pernah sampai ke objek pesanan itu sendiri. Trik tuple ini adalah cara baku memberi `heapq` prioritas buatan sendiri.

- **Indeks generik.** `MultiIndex(Generic[K, V])`, dengan `K` dibatasi protokol `SupportsLessThan` yang sama seperti di **Kode 9.24**, memakai `setdefault(key, [])`, padanan `computeIfAbsent`, dan mengembalikan *tuple* dari `get`, sehingga pemanggil menerima potret yang imutabel, bukan list milik indeks itu sendiri.
- **Nomor kedatangan.** `itertools.count(1)` adalah iterator tanpa akhir 1, 2, 3, …, yang dipakai bersama oleh semua pesanan sebagai atribut kelas, seperti field `static` di Java. mypy memeriksa seluruh berkas tanpa satu keluhan pun.

9.12 Profitina dalam Praktik

Profitina adalah platform visibilitas AI: ia memindai situs web sebuah merek dan mengukur seberapa siap situs itu untuk dikutip oleh mesin penjawab AI, disiplin yang disebut *generative engine optimization* (GEO). Produknya ditulis dalam Python, dan koleksi-koleksi bab ini adalah perkakas sehari-harinya, dipakai persis dalam peran yang diberikan buku pesanan kepada mereka.

Sebuah pemindaian mengunjungi banyak halaman sebuah situs, dan penjelajahan (*crawl*) tidak boleh mengambil halaman yang sama dua kali. Sebelum sebuah URL dimasukkan ke antrean, ia dibawa ke bentuk *kanonis* — satu ejaan untuk berbagai cara menulis alamat yang sama — dan sebuah **himpunan** URL kanonis menjawab "sudah pernah dilihat?" dalam waktu harapan konstan, pertanyaan pada Persamaan (9.5). Halaman-halaman yang menunggu diambil membentuk **antrean**; ketika sebagian halaman lebih penting daripada yang lain, antrean itu menjadi **antrean prioritas**, struktur dari Subbab 9.7. Hasilnya disimpan dalam **map** dari halaman ke hasil ukur dan dari sinyal ke skor, dan tren mingguan yang dilihat pemilik merek di dasbor adalah map dari minggu ke skor, dibaca secara terurut dan diekspor sebagai CSV. Umpan komunitasnya menyimpan penghitung "N baru" per kanal: map dari kanal ke hitungan, diperbarui dengan padanan `merge`.

Dalam basis kode sebesar itu, petunjuk tipe seperti `dict[str, list[Signal]]` menyatakan dalam satu baris apa yang dikembalikan sebuah fungsi, dan itulah tawaran **Tabel 9.4**; gerbang rilisnya sendiri adalah rangkaian lebih dari seribu uji otomatis yang harus lulus sebelum sebuah versi dikirim. Dan, seperti diperlihatkan buku pesanan, beberapa indeks atas objek-objek yang sama hanya tetap benar bila satu kelas memilikinya dan satu metode mengubahnya. Seperti biasa, hasil ukurnya adalah saran: platform menjelaskan apa yang ditemukannya, dan penggunalah yang memutuskan apa yang akan dilakukan.

9.13 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya

1. Mendeklarasikan kelas alih-alih antarmuka. `ArrayList<String> names = new ArrayList<>();` mengikat setiap metode yang menerima `names` pada satu implementasi. *Hindari:* deklarasikan variabel dan parameter dengan antarmuka, `List<String>`, `Map<String, Order>`; sebut nama kelas hanya sesudah `new` (Subbab 9.3.2).

2. Menghapus dari koleksi di dalam perulangan for-each. Java melempar `ConcurrentModificationException`; Python diam-diam melewatkan elemen (**Kode 9.8**). *Hindari:* pakailah `Iterator.remove`, `removeIf`, *comprehension*, atau perulangan atas salinan.

3. Memanggil `get(i)` dalam perulangan atas `LinkedList`, atau `pop(0)` dalam perulangan atas `list` Python. Setiap panggilan $O(n)$, sehingga perulangannya $O(n^2)$. *Hindari:* telusuri list tertaut dengan for-each; pakailah `ArrayDeque` atau `collections.deque` untuk antrean.

4. Memakai `list.contains` atau `x in list` untuk uji keanggotaan yang sering. Setiap uji memindai list. *Hindari:* simpan sebuah `HashSet` atau `set` di samping list, atau sebagai penggantinya (**Gambar 9.9**).

5. Mengandalkan urutan iterasi `HashSet`, `HashMap`, atau `set` Python. Urutannya bergantung pada kode hash dan ukuran tabel, dan untuk string Python berubah setiap kali program dijalankan. *Hindari:* pilih `LinkedHashSet`/`LinkedHashMap` (urutan penyisipan) atau `TreeSet`/`TreeMap` (terurut), atau urutkan secara eksplisit saat mencetak.

6. Mengubah field yang menjadi dasar `hashCode` atau `compareTo` selagi objeknya berada di himpunan hash, menjadi kunci map, atau berada di pohon. Objek itu terdampar di ember atau cabang yang salah; `contains` dan `remove` tidak lagi menemukannya (Soal Latihan 9.8). *Hindari:* dasarkan `equals`/`hashCode` pada field yang tidak pernah berubah, seperti id, dan jadikan `final`; di Python, berikan `__hash__` hanya kepada objek yang field-field ber-hash-nya imutabel.

7. Menulis comparator dengan pengurangan. `a - b` melimpah untuk nilai besar atau negatif dan melaporkan urutan yang salah (**Kode 9.31**). *Hindari:* `Integer.compare(a, b)`, `Comparator.comparingInt(...)`; di Python, fungsi `key=`.

8. Memakai comparator yang menyebut objek-objek berbeda "sama" di dalam `TreeSet` atau `TreeMap`. Pohon hanya menyimpan salah satunya. *Hindari:* tambahkan pemecah seri sampai "dibandingkan sama" berarti "elemen yang sama".

9. Memakai tipe mentah atau mengabaikan peringatan "unchecked". Keduanya mematikan pemeriksaan kompilator dan membuka jalan bagi polusi heap (**Kode 9.27**). *Hindari:* selalu tulis argumen tipe, kompilasilah dengan `-Xlint:all`, dan perbaiki setiap peringatan; bungkam satu peringatan hanya dengan komentar yang menjelaskan mengapa itu aman.

10. Mengharapkan `List<Cat>` merupakan `List<Animal>`. Bukan (Persamaan (9.7)). *Hindari:* metode yang hanya membaca sebaiknya menerima `List<? extends Animal>` (Python: `Sequence[Animal]`); metode yang hanya menambah sebaiknya menerima `List<? super Cat>`.

11. Mengira petunjuk tipe Python diperiksa saat program berjalan. Tidak; `list[int]` menerima string saat eksekusi (**Kode 9.28**). *Hindari:* jalankan mypy (atau pemeriksa lain) sebagai bagian dari pengujian, dan anotasilah fungsi-fungsi yang ingin diperiksa [3], [15].

12. `d[key]` pada kunci yang tidak ada di Python, `map.get(key).method()` pada kunci yang tidak ada di Java. Yang pertama memunculkan `KeyError`, yang kedua `NullPointerException`. *Hindari:* `d.get(key, default)`, `getOrDefault`, `computeIfAbsent`, `defaultdict`.

13. Membiarkan beberapa indeks atas objek-objek yang sama saling menyimpang. Pesanan di himpunan status yang salah memberi jawaban salah secara diam-diam. *Hindari:* satu kelas memiliki semua indeks, hanya satu metode yang mengubah field yang diindeks, dan pemeriksaan invarian seperti Persamaan (9.8) dijalankan dalam pengujian.

9.14 Ringkasan Bab

- Antarmuka boleh mendeklarasikan **konstanta**; konstanta itu otomatis `public static final`. Untuk himpunan tetap nilai-nilai yang berkaitan, **enum** lebih aman, baik di Java maupun Python. `Final` Python diperiksa oleh mypy, bukan saat eksekusi.

- Metode **rekursif** memerlukan kasus dasar yang dicapai setiap rantai panggilan dan panggilan rekursif pada masalah yang lebih kecil. Fibonacci naif melakukan $2F(n+1)-1$ panggilan (Persamaan (9.2)); membawa keadaan dalam parameter tambahan membuatnya linear, dan **metode pembantu** menyembunyikan parameter itu dari pemanggil.
- **Collections Framework** (**Gambar 9.2**) menamai jenis-jenis koleksi dengan antarmuka — `Collection`, `List`, `Set`, `Queue`, `Deque`, dan `Map` yang terpisah — dan mengimplementasikan masing-masing dengan beberapa cara. Padanan Python adalah `list`, `tuple`, `set`, `frozenset`, `dict` bawaan, ditambah `deque`, `Counter`, `defaultdict`, dan `heapq` (**Tabel 9.2**). Memprogramlah terhadap antarmuka.
- **List** menjaga posisi dan duplikat. `ArrayList` / `list` memberi akses lewat indeks dalam $O(1)$ dan penambahan di ujung yang murah; menyisipkan di depan berbiaya $O(n)$ (Persamaan (9.3)). Telusuri dengan for-each atau `Iterator`; hapus melalui iterator, `removeIf`, atau *comprehension*.
- **Set** tidak menyimpan duplikat. **Himpunan hash** menghitung ember sebuah elemen dari kode hash-nya dan menjawab `contains` dalam waktu harapan $O(1)$, karena faktor muatannya (Persamaan (9.4)) dijaga di bawah 0,75 (Persamaan (9.5)); **himpunan pohon** menjaga urutan terurut dalam pohon merah-hitam setinggi $O(\log n)$ (Persamaan (9.6)); **himpunan hash tertaut** menjaga urutan penyisipan.
- **Map** mengasosiasikan kunci unik dengan nilai: `put` mengembalikan nilai lama, `get` mengembalikan `null` (Python: `KeyError`, atau `get` dengan nilai bawaan), `merge`/`Counter` menghitung, `computeIfAbsent`/`defaultdict` mengelompokkan.
- **Antrean** bersifat FIFO, **tumpukan** LIFO, keduanya dilayani `ArrayDeque` / `deque`; **antrean prioritas** (`PriorityQueue`, `heapq`) adalah *binary heap* yang melepaskan elemen terkecil dalam $O(\log n)$.
- **Generik** memungkinkan satu kelas atau metode bekerja untuk sembarang tipe sementara kompilator memeriksa setiap pemakaian: parameter tipe `<T>`, metode generik `<T> T f(...)`, batas `<T extends Comparable<T>>`. Tipe generik bersifat **invarian** (Persamaan (9.7)); wildcard `? extends` (produsen, dibaca) dan `? super` (konsumen, ditulisi) memulihkan keluwesan — PECS (**Tabel 9.3**). Java **menghapus** argumen tipe saat eksekusi; Python tidak pernah memilikinya saat eksekusi, dan mypy memeriksa petunjuk `TypeVar`, `Generic`, dan `list[int]`-nya (**Tabel 9.4**).
- Urutan berasal dari `Comparable` (urutan alami) atau `Comparator` (urutan apa pun); bangun comparator dengan `comparing`, `thenComparing`, dan `reversed`; pengurutan objek Java dan pengurutan Python bersifat **stabil**. Python mengurutkan dengan **fungsi kunci**.
- **Pilihlah koleksi** menurut pertanyaan yang akan Anda ajukan (**Gambar 9.10**) dan biaya pada **Tabel 9.5** dan **Tabel 9.6**. Buku pesanan binatu memadukan `TreeMap`, `MultiIndex` generik, `EnumMap` berisi himpunan, dan `PriorityQueue` atas objek-objek yang sama, yang dijaga tetap konsisten oleh satu invarian (Persamaan (9.8)).

9.15 Soal Latihan

Soal Latihan 9.1 (memilih). Untuk setiap tugas, sebutkan kelas koleksi Java dan tipe Python yang akan Anda pakai, dan beri alasan dalam satu kalimat dengan memakai **Gambar 9.10** dan **Tabel 9.5**. (a) Tiket yang dicetak hari ini, menurut urutan pencetakan. (b) Himpunan pelanggan yang datang bulan ini, ditanyai "apakah ia sudah datang?" ratusan kali sehari. (c) Harga setiap kode layanan. (d) Pesanan yang menunggu satu-satunya meja setrika, dilayani menurut urutan kedatangan. (e) Nama-nama pelanggan menurut abjad, dengan pertanyaan "siapa yang tepat sesudah Rina?". (f) Tugas antar untuk kurir, selalu mengambil yang waktu janjinya paling awal. (g) Riwayat *undo* yang membatalkan tindakan paling akhir lebih dulu.

Soal Latihan 9.2 (menelusuri). Tanpa menjalankannya, ramalkan satu baris yang dicetak oleh

```
List<Integer> a = new ArrayList<>(List.of(4, 1, 4, 2));
Set<Integer> s = new TreeSet<>(a);
Map<Integer, Integer> m = new TreeMap<>();
for (int x : a) {
    m.merge(x, 1, Integer::sum);
}
a.remove(Integer.valueOf(4));
a.remove(1);
Deque<Integer> d = new ArrayDeque<>(a);
d.push(9);
d.offerLast(7);
System.out.println(a + " " + s + " " + m + " " + d + " " + d.pollLast() + d.pop());
```

Lalu tulislah langkah-langkah yang sama dalam Python dengan `list`, `set`, `Counter`, dan `deque`, dan ramalkan keluarannya.

Soal Latihan 9.3 (rekursi dengan pembantu). Tambahkan pada `GList<T>` di **Kode 9.19** metode `GList<T> reversed()` yang mengembalikan list baru dengan elemen-elemen dalam urutan terbalik. Tulislah secara rekursif, dengan pembantu privat yang membawa akumulator, sehingga metode itu melakukan $O(n)$ panggilan. Kerjakan hal yang sama dalam Python. Mengapa versi yang "jelas" — balik ekornya, lalu tambahkan kepalanya di ujung — lebih lambat untuk list yang dibangun dari sel-sel `cons`?

Soal Latihan 9.4 (anagram). Dua kata adalah *anagram* bila memakai huruf yang sama dengan jumlah yang sama: "listen", "silent", "enlist", dan "tinsel". Tulislah program yang mengelompokkan kata-kata `one, listen, neo, silent, loop, pool, polo, enlist, eon, aab, abb, tinsel` ke dalam kelompok-kelompok anagram dan mencetak setiap kelompok yang anggotanya lebih dari satu. Pakailah map yang kuncinya adalah huruf-huruf kata itu dalam urutan terurut. Jelaskan mengapa membandingkan *himpunan* huruf, seperti di **Kode 9.9**, akan keliru.

Soal Latihan 9.5 (tumpukan generik). Tulislah kelas generik `ArrayStack<T>` di Java, dan `ArrayStack(Generic[T])` di Python, dengan `push`, `pop`, `peek`, `isEmpty`/`is_empty`, dan `size`/`__len__`, yang ditopang list privat dengan *ujung* list sebagai puncaknya. `pop` dan `peek` pada tumpukan kosong harus melempar eksepsi yang jelas. Pakailah tumpukan itu dalam fungsi `balanced(text)` yang memeriksa apakah kurung `()`, `[]`, dan `{}` dalam sebuah teks berpasangan dan bersarang dengan benar, lalu ujilah pada `(a[b]{c})`, `(a[b)]`, `((`, dan `x}`.

Soal Latihan 9.6 (`Comparable<? super T>`). Diberikan

```
class Fruit implements Comparable<Fruit> {
    final int grams;
    Fruit(int grams) { this.grams = grams; }
    public int compareTo(Fruit other) { return Integer.compare(grams, other.grams); }
}
class Apple extends Fruit {
    Apple(int grams) { super(grams); }
}
```

jelaskan mengapa `max(apples)` gagal dikompilasi untuk `List<Apple> apples` bila `max` dideklarasikan seperti di **Kode 9.23**, `static <T extends Comparable<T>> T max(List<T> items)`. Ubahlah batasnya agar lolos kompilasi, dan jelaskan perubahan itu dengan PECS. (Inilah sebabnya pustaka mendeklarasikan `Collections.max` sebagai `<T extends Object & Comparable<? super T>> T max(Collection<? extends T> coll)`.)

Soal Latihan 9.7 (wildcard). Dengan `Cat` dan `Dog` memperluas `Animal`, serta deklarasi `List<Cat> cats`, `List<Animal> animals`, `List<Object> objects`, `List<? extends Animal> readers`, `List<? super Cat> writers`, sebutkan pernyataan mana yang lolos kompilasi, dan mengapa: (a) `readers`

`= cats;` (b) `readers = objects;` (c) `writers = animals;` (d) `writers = cats;` (e) `Animal a = readers.get(0);` (f) `readers.add(new Cat());` (g) `writers.add(new Cat());` (h) `Cat c = writers.get(0);` (i) `Object o = writers.get(0);` (j) `animals = cats;`.

Soal Latihan 9.8 (kode hash yang berubah). Kelas `Tag` punya field `code` yang dapat berubah, dan `equals` serta `hashCode`-nya memakai `code`. Ramalkan dua baris yang dicetak oleh

```
Set<Tag> set = new HashSet<>();
Tag t = new Tag("WASH");
set.add(t);
t.code = "IRON";
System.out.println(set.contains(t) + " " + set.contains(new Tag("WASH")) + " " +
set.size());
set.add(t);
System.out.println(set.size() + " " + set);
```

(`toString` mengembalikan `code`.) Jelaskan setiap nilai dengan **Gambar 9.5**. Apa yang dilakukan percobaan yang sama di Python, dan apa pelajarannya bagi `LaundryOrder` di **Kode 9.33**?

Soal Latihan 9.9 (menghitung biaya). (a) Dengan memakai Persamaan (9.3), berapa pergeseran elemen yang ditimbulkan penyisipan $n = 100{,}000$ elemen satu per satu di indeks 0 sebuah `ArrayList`? Berapa perubahan tautan yang dibutuhkan pekerjaan yang sama dengan `LinkedList` atau `addFirst` milik `ArrayDeque`? (b) Sebuah list berisi 10^6 id pelanggan memuat duplikat. Bandingkan banyaknya perbandingan untuk menghapus duplikat dengan perulangan bersarang ("untuk setiap elemen, apakah ia ada di antara elemen yang sudah disimpan?") dan dengan `HashSet`, dengan anggapan paling banyak ada 10^6 id berbeda. (c) Berapa tingkat yang dimiliki *binary heap* berisi 10^6 elemen, dan karena itu paling banyak berapa pertukaran yang dilakukan satu `poll`? (d) Menurut Persamaan (9.6), paling banyak berapa perbandingan yang dilakukan `contains` pada `TreeSet` berisi 10^6 elemen?

Soal Latihan 9.10 (cache LRU). Cache *least recently used* (LRU, paling lama tidak dipakai) memegang paling banyak c entri; ketika entri baru akan melampaui kapasitas, ia mengeluarkan entri yang paling lama tidak dibaca atau ditulis. Implementasikan `LruCache<K, V>` generik di Java dengan memperluas `LinkedHashMap` dalam *urutan akses* dan meng-override `removeEldestEntry`, serta `LruCache(Generic[K, V])` generik di Python dengan `collections.OrderedDict` dan `move_to_end`. Peragakan dengan kapasitas 3: put `L-001`, `L-002`, `L-003`, get `L-001`, put `L-004`, dan tunjukkan entri mana yang dikeluarkan. Bab 7 memperingatkan agar tidak memperluas kelas konkret; mengapa memperluas `LinkedHashMap` dapat diterima di sini?

Soal Latihan 9.11 (membatalkan pesanan). Tambahkan `cancel(id)` pada buku pesanan di **Kode 9.33** dan **Kode 9.34**. Hanya pesanan yang masih `RECEIVED` yang boleh dibatalkan; pesanan itu harus hilang dari *setiap* struktur, agar invarian Persamaan (9.8) tetap berlaku dan list milik Budi tidak lagi menampilkannya. Anda akan memerlukan metode `remove(key, value)` pada `MultiIndex`. Berapa biaya menghapus elemen sembarang dari `PriorityQueue` dan dari list `heapq`, dan alternatif apa ("penghapusan malas", *lazy deletion*) yang menghindari biaya itu?

Soal Latihan 9.12 (varians di Python). mypy menolak panggilan dalam berkas `weights.py`

```
class Order:
    def __init__(self, weight: float) -> None:
        self.weight = weight


class ExpressOrder(Order):
    pass


def total_weight(orders: list[Order]) -> float:
    return sum(o.weight for o in orders)


def main() -> None:
    express: list[ExpressOrder] = [ExpressOrder(2.5), ExpressOrder(4.0)]
    print(total_weight(express))


main()
```

walaupun program itu berjalan dan mencetak `6.5`. Jelaskan keberatan mypy dengan Persamaan (9.7), perbaiki tanda tangannya agar mypy menerima panggilan itu, dan berikan contoh badan fungsi yang membuat keberatan mypy beralasan.

9.16 Penyelesaian Soal Latihan

Penyelesaian 9.1. (a) `ArrayList` / `list`: posisi dan urutan kedatangan, penambahan di ujung. (b) `HashSet` / `set`: keanggotaan sering diuji, $O(1)$ harapan. (c) `HashMap` / `dict` (atau `EnumMap` bila kodenya berupa enum): pencarian lewat kunci. (d) `ArrayDeque` yang dipakai sebagai antrean / `collections.deque`: FIFO, kedua ujung $O(1)$. (e) `TreeSet` / `list` terurut dengan `bisect`: urutan terurut dan `higher("Rina")`, elemen terkecil yang sungguh-sungguh lebih besar; di Python `names[bisect_right(names, "Rina")]`. (f) `PriorityQueue` dengan comparator atas waktu janji / `heapq` dengan tuple `(time, seq, job)`. (g) `ArrayDeque` yang dipakai sebagai tumpukan (`push`, `pop`) / `list` dengan `append` dan `pop`. (Kelas lama Java `Stack` juga bisa, tetapi memperluas `Vector` dan tidak dianjurkan.)

Penyelesaian 9.2. Program Java mencetak

```
[1, 2] [1, 2, 4] {1=1, 2=1, 4=2} [9, 1, 2, 7] 79
```

`s` adalah himpunan terurut nilai-nilai berbeda; `m` menghitungnya (4 muncul dua kali). `a.remove(Integer.valueOf(4))` menghapus *nilai* 4, kemunculan pertamanya, menyisakan `[1, 4, 2]`; `a.remove(1)` menghapus *indeks* 1 (jebakan *boxing* dari Bab 5), menyisakan `[1, 2]`. Deque dibangun dari `[1, 2]`; `push` menambahkan 9 di depan, `offerLast` menambahkan 7 di belakang. Penyambungan string dievaluasi dari kiri ke kanan, sehingga `d` tercetak sebelum `pollLast()` mengembalikan 7 dan `pop()` mengembalikan 9; `7 + 9` tidak dijumlahkan, karena operan kirinya sudah berupa `String`, sehingga karakter `79` yang disambungkan. Versi Python

```
a = [4, 1, 4, 2]; s = sorted(set(a)); m = dict(sorted(Counter(a).items()))
a.remove(4); del a[1]
d = deque(a); d.appendleft(9); d.append(7)
print(a, s, m, list(d), d.pop(), d.popleft())
```

mencetak `[1, 2] [1, 2, 4] {1: 1, 2: 1, 4: 2} [9, 1, 2, 7] 7 9`. `list(d)` membuat potret sebelum kedua pop berjalan, karena argumen-argumen dievaluasi dari kiri ke kanan sebelum `print` mulai.

Penyelesaian 9.3. Di Java:

```
    public GList<T> reversed() {
        return reversedHelper(this, new GList<>());
    }

    private GList<T> reversedHelper(GList<T> rest, GList<T> accumulator) {
        if (rest.empty()) {
            return accumulator;                                    // kasus dasar
        }
        return reversedHelper(rest.tail(), new GList<>(rest.head(), accumulator));
    }
```

Setiap panggilan memindahkan satu elemen dari depan `rest` ke depan `accumulator`, seperti memindahkan piring dari satu tumpukan ke tumpukan lain; setelah n panggilan, akumulator memuat elemen-elemen dalam urutan terbalik. Di Python, `def reversed(self) -> GList[T]: return self._rev(self, GList.nil())` dengan `_rev(rest, acc)` mengembalikan `acc` bila `rest.empty()` dan `self._rev(rest.tail(), GList.cons(rest.head(), acc))` bila tidak. Versi yang "jelas" harus menambahkan di *ujung* ekor yang sudah dibalik, padahal list `cons` hanya dapat menambah di depan; menambah di ujung berarti menyalin seluruh list, $O(k)$ untuk list berisi k elemen, sehingga totalnya $1 + 2 + \cdots + (n-1) = \frac{n(n-1)}{2}$, jumlah yang sama dengan Persamaan (9.3): $O(n^2)$.

Penyelesaian 9.4. **Kode 9.35** dan **Kode 9.36** mengelompokkan kata-kata menurut huruf-hurufnya yang diurutkan.

Kode 9.35 `Anagrams.java`: kelompok-kelompok anagram dalam `TreeMap` yang berkunci huruf-huruf terurut setiap kata.

```java
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.Map;
import java.util.TreeMap;

public class Anagrams {
    // "neo", "one", dan "eon" semuanya berkunci "eno": huruf-hurufnya, diurutkan.
    static String key(String word) {
        char[] letters = word.toLowerCase().toCharArray();
        Arrays.sort(letters);
        return new String(letters);
    }

    static Map<String, List<String>> groups(List<String> words) {
        Map<String, List<String>> result = new TreeMap<>();
        for (String w : words) {
            result.computeIfAbsent(key(w), k -> new ArrayList<>()).add(w);
        }
        return result;
    }

    public static void main(String[] args) {
        List<String> words = List.of("one", "listen", "neo", "silent", "loop",
                "pool", "polo", "enlist", "eon", "aab", "abb", "tinsel");
        Map<String, List<String>> g = groups(words);
        for (Map.Entry<String, List<String>> e : g.entrySet()) {
            if (e.getValue().size() > 1) {
                System.out.println(e.getKey() + " -> " + e.getValue());
            }
        }
        System.out.println(g.size() + " groups from " + words.size() + " words");
    }
}
```

```
eilnst -> [listen, silent, enlist, tinsel]
eno -> [one, neo, eon]
loop -> [loop, pool, polo]
5 groups from 12 words
```

Kode 9.36 `anagrams.py`: pengelompokan yang sama dengan `defaultdict`.

```
"""anagrams.py: mengelompokkan kata-kata yang saling anagram."""
from collections import defaultdict


def key(word: str) -> str:
    """'neo', 'one', dan 'eon' semuanya berkunci 'eno': huruf-hurufnya, diurutkan."""
    return "".join(sorted(word.lower()))


def groups(words: list[str]) -> dict[str, list[str]]:
    result: defaultdict[str, list[str]] = defaultdict(list)
    for w in words:
        result[key(w)].append(w)
    return dict(sorted(result.items()))


def main() -> None:
    words = ["one", "listen", "neo", "silent", "loop", "pool", "polo", "enlist",
             "eon", "aab", "abb", "tinsel"]
    g = groups(words)
    for k, members in g.items():
        if len(members) > 1:
            print(f"{k} -> {members}")
    print(f"{len(g)} groups from {len(words)} words")
    print("set('aab') == set('abb'):", set("aab") == set("abb"), "- but they are not
anagrams")


if __name__ == "__main__":
    main()
```

```
eilnst -> ['listen', 'silent', 'enlist', 'tinsel']
eno -> ['one', 'neo', 'eon']
loop -> ['loop', 'pool', 'polo']
5 groups from 12 words
set('aab') == set('abb'): True - but they are not anagrams
```

`key("listen")` mengurutkan huruf-hurufnya menjadi `eilnst`; setiap anagram dari "listen" punya kunci yang sama, dan tidak ada kata lain yang punya, karena huruf-huruf terurut mencatat setiap huruf *beserta kelipatannya*. `computeIfAbsent` dan `defaultdict(list)` membuat setiap kelompok pada pemakaian pertama. Dua belas kata membentuk lima kelompok: tiga yang tercetak, dan dua kelompok tunggal `aab` dan `abb`. Himpunan huruf melupakan berapa kali setiap huruf muncul, sehingga `set("aab") == set("abb")`, seperti diperlihatkan baris terakhir Python, walaupun kedua kata itu bukan anagram.

Penyelesaian 9.5. **Kode 9.37** dan **Kode 9.38** mengimplementasikan tumpukan dan pemeriksa kurung.

Kode 9.37 `StackDemo.java`: `ArrayStack<T>` generik yang puncaknya adalah ujung sebuah `ArrayList` privat, dan pemeriksa kurung yang dibangun di atasnya.

```
import java.util.ArrayList;
import java.util.List;
import java.util.NoSuchElementException;

public class StackDemo {
    // Apakah setiap kurung ditutup pasangan yang tepat, dalam urutan yang tepat?
    static boolean balanced(String text) {
        ArrayStack<Character> open = new ArrayStack<>();
        for (char c : text.toCharArray()) {
            if (c == '(' || c == '[' || c == '{') {
                open.push(c);
            } else if (c == ')' || c == ']' || c == '}') {
                if (open.isEmpty()) {
                    return false;
                }
                char o = open.pop();
                if ((c == ')' && o != '(') || (c == ']' && o != '[') || (c == '}' && o !=
'{')) {
                    return false;
                }
            }
        }
        return open.isEmpty();
    }

    public static void main(String[] args) {
        ArrayStack<String> trays = new ArrayStack<>();
        trays.push("tray A");
        trays.push("tray B");
        trays.push("tray C");
        System.out.println(trays + " size " + trays.size() + ", peek " + trays.peek());
        System.out.println("pop " + trays.pop() + ", pop " + trays.pop() + ", left " +
trays);

        for (String s : new String[] {"(a[b]{c})", "(a[b)]", "((", "x}"}) {
            System.out.println(s + " balanced: " + balanced(s));
        }
        try {
            new ArrayStack<Integer>().pop();
        } catch (NoSuchElementException e) {
            System.out.println("NoSuchElementException: " + e.getMessage());
        }
    }
}
```

Kode 9.37 (Lanjutan) *StackDemo.java: ArrayStack<T> generik yang puncaknya adalah ujung sebuah ArrayList privat, dan pemeriksa kurung yang dibangun di atasnya.*

```
// Tumpukan generik. ArrayList-nya privat: pemanggil hanya melihat operasi tumpukan.
class ArrayStack<T> {
    private final List<T> items = new ArrayList<>();

    void push(T item) {
        items.add(item);                    // puncaknya adalah UJUNG list: O(1)
    }

    T pop() {
        T top = peek();
        items.remove(items.size() - 1);
        return top;
    }

    T peek() {
        if (items.isEmpty()) {
            throw new NoSuchElementException("stack is empty");
        }
        return items.get(items.size() - 1);
    }

    boolean isEmpty() {
        return items.isEmpty();
    }

    int size() {
        return items.size();
    }

    @Override
    public String toString() {
        return "bottom " + items + " top";
    }
}
```

```
bottom [tray A, tray B, tray C] top size 3, peek tray C
pop tray C, pop tray B, left bottom [tray A] top
(a[b]{c}) balanced: true
(a[b)] balanced: false
(( balanced: false
x} balanced: false
NoSuchElementException: stack is empty
```

Kode 9.38 `stack_demo.py`: tumpukan generik dalam Python, dan pemeriksa kurung yang sama dengan tabel pasangan.

```python
"""stack_demo.py: kelas tumpukan generik, dan pemeriksa kurung yang memakainya."""
from typing import Generic, TypeVar

T = TypeVar("T")


class ArrayStack(Generic[T]):
    """Tumpukan generik. List-nya privat: pemanggil hanya melihat operasi tumpukan."""

    def __init__(self) -> None:
        self._items: list[T] = []

    def push(self, item: T) -> None:
        self._items.append(item)            # puncaknya adalah UJUNG list: O(1)

    def pop(self) -> T:
        top = self.peek()
        self._items.pop()
        return top

    def peek(self) -> T:
        if not self._items:
            raise IndexError("stack is empty")
        return self._items[-1]

    def is_empty(self) -> bool:
        return not self._items

    def __len__(self) -> int:
        return len(self._items)

    def __str__(self) -> str:
        return f"bottom {self._items} top"


PARTNER = {")": "(", "]": "[", "}": "{"}
```

Kode 9.38 (Lanjutan) *stack_demo.py: tumpukan generik dalam Python, dan pemeriksa kurung yang sama dengan tabel pasangan.*

```
def balanced(text: str) -> bool:
    """Apakah setiap kurung ditutup pasangan yang tepat, dalam urutan yang tepat?"""
    open_brackets: ArrayStack[str] = ArrayStack()
    for c in text:
        if c in "([{":
            open_brackets.push(c)
        elif c in PARTNER:
            if open_brackets.is_empty() or open_brackets.pop() != PARTNER[c]:
                return False
    return open_brackets.is_empty()


def main() -> None:
    trays: ArrayStack[str] = ArrayStack()
    trays.push("tray A")
    trays.push("tray B")
    trays.push("tray C")
    print(f"{trays} size {len(trays)}, peek {trays.peek()}")
    print(f"pop {trays.pop()}, pop {trays.pop()}, left {trays}")

    for s in ["(a[b]{c})", "(a[b)]", "((", "x}"]:
        print(f"{s} balanced: {balanced(s)}")
    try:
        ArrayStack[int]().pop()
    except IndexError as e:
        print("IndexError:", e)


if __name__ == "__main__":
    main()
```

```
bottom ['tray A', 'tray B', 'tray C'] top size 3, peek tray C
pop tray C, pop tray B, left bottom ['tray A'] top
(a[b]{c}) balanced: True
(a[b)] balanced: False
(( balanced: False
x} balanced: False
IndexError: stack is empty
```

Puncaknya adalah *ujung* list, sehingga `push` dan `pop` berbiaya $O(1)$, dan list-nya privat, sehingga tidak ada pemanggil yang dapat menyisipkan di tengah: komposisi, bukan pewarisan (Bab 7). Pemeriksa itu memasukkan setiap kurung buka ke tumpukan; kurung tutup harus cocok dengan kurung yang paling akhir dibuka, yaitu tepat elemen di puncak. `(a[b)]` gagal karena `)` datang ketika `[` berada di puncak; `((` gagal karena dua kurung masih terbuka di akhir; `x}` gagal karena tumpukan kosong ketika `}` datang.

Penyelesaian 9.6. Untuk `List<Apple>`, kompilator harus memilih `T = Apple` (tipe elemen list itu) lalu memeriksa batas `Apple extends Comparable<Apple>`. Pemeriksaan itu gagal: `Apple` mewarisi `Comparable<Fruit>`, bukan `Comparable<Apple>`, dan tipe generik bersifat invarian. Dalam program uji `SuperBound.java` yang versi ketatnya dinamai `maxStrict`, `javac` berkata:

```
SuperBound.java:19: error: method maxStrict in class SuperBound cannot be applied to given
types;
        System.out.println(maxStrict(apples));
                           ^
  required: List<T>
  found:    List<Apple>
  reason: inference variable T has incompatible equality constraints Fruit,Apple
  where T is a type-variable:
    T extends Comparable<T> declared in method <T>maxStrict(List<T>)
1 error
```

Perbaikannya adalah `static <T extends Comparable<? super T>> T max(List<T> items)`: `T` harus dapat dibandingkan dengan *suatu supertipe* dirinya. `Apple` dapat dibandingkan dengan `Fruit`, supertipe `Apple`, sehingga batasnya terpenuhi, dan `max(apples)` mengembalikan apel 210 g. Dalam bahasa PECS, `Comparable` *mengonsumsi* nilai `T` (`compareTo`-nya menerima satu nilai sebagai argumen), sehingga argumen tipenya sebaiknya `? super T`.

Penyelesaian 9.7. (a) Lolos: `List<Cat>` adalah subtipe `List<? extends Animal>`. (b) Tidak lolos: `Object` bukan subtipe `Animal`. (c) Lolos: `Animal` adalah supertipe `Cat`. (d) Lolos: `Cat` adalah supertipe dirinya sendiri. (e) Lolos: apa pun isi `readers`, elemen-elemennya adalah `Animal`. (f) Tidak lolos: `readers` bisa saja `List<Dog>`. (g) Lolos: apa pun isi `writers`, ia dapat memuat `Cat`. (h) Tidak lolos: `writers` bisa saja `List<Object>`, sehingga `get` hanya mengembalikan `Object`; cast akan lolos kompilasi tetapi bisa gagal. (i) Lolos. (j) Tidak lolos: Persamaan (9.7).

Penyelesaian 9.8. Program itu mencetak

```
false false 1
2 [IRON, IRON]
```

Ketika `t` ditambahkan, kode hash-nya adalah kode hash `"WASH"`, sehingga ia disimpan di ember milik `"WASH"`. Setelah `t.code = "IRON"`, `contains(t)` menghitung ember milik `"IRON"`, melihat ke sana, dan tidak menemukan apa-apa: `false`. `contains(new Tag("WASH"))` melihat ke ember yang benar dan menemukan `t`, tetapi `equals` membandingkan `"WASH"` dengan kode `t` saat ini, `"IRON"`: `false`. Ukurannya tetap 1. `set.add(t)` lalu melihat ke ember `"IRON"`, tidak menemukan apa-apa, dan menambahkan `t` *untuk kedua kalinya*: "himpunan" itu kini memuat objek yang sama dua kali. `set` Python berperilaku identik (`False False 1`, lalu `2`). Pelajarannya: field yang dipakai `equals`/`hashCode` (dan `compareTo` di dalam pohon) tidak boleh berubah selama objeknya berada di dalam koleksi. Itulah sebabnya `LaundryOrder` mendasarkan keduanya pada id yang `final`, dan mengapa `status`-nya yang dapat berubah tidak ikut serta.

Penyelesaian 9.9. (a) $\frac{n(n-1)}{2} = \frac{100{,}000 \times 99{,}999}{2} = 4{,}999{,}950{,}000$ pergeseran, sekitar lima miliar. `LinkedList.addFirst` mengubah tiga referensi per penyisipan, seluruhnya sekitar $3n = 300{,}000$; `ArrayDeque.addFirst` menulis satu slot per penyisipan, sekitar n kali tulis ditambah penggandaan sesekali. (b) Perulangan bersarang membandingkan setiap elemen dari 10^6 elemen dengan hingga 10^6 elemen yang sudah disimpan, hingga sekitar $\frac{10^{12}}{2}$ perbandingan pada kasus terburuk; himpunan hash melakukan sekitar 10^6 langkah berbiaya harapan $O(1)$: selisih sekitar setengah juta kali. (c) $\lfloor \log_2 10^6 \rfloor + 1 = 19 + 1 = 20$ tingkat, sehingga satu `poll` melakukan paling banyak 19 pertukaran ketika elemen yang dipindahkan tenggelam. (d) $2\log_2(10^6 + 1) \approx 2 \times 19.93 \approx 39.9$, jadi paling banyak 39 tingkat di bawah akar, 40 perbandingan.

Penyelesaian 9.10. **Kode 9.39** dan **Kode 9.40** adalah kedua cache itu.

Kode 9.39 `LruCache.java`: cache LRU dalam selusin baris, dengan memperluas `LinkedHashMap` dalam urutan akses.

```java
import java.util.LinkedHashMap;
import java.util.Map;

// Cache yang melupakan entri yang paling lama tidak dipakai (LRU) saat penuh.
public class LruCache<K, V> extends LinkedHashMap<K, V> {
    private static final long serialVersionUID = 1L;
    private final int capacity;

    public LruCache(int capacity) {
        super(16, 0.75f, true);            // true = urutan menurut AKSES, bukan penyisipan
        this.capacity = capacity;
    }

    @Override
    protected boolean removeEldestEntry(Map.Entry<K, V> eldest) {
        return size() > capacity;          // dipanggil oleh put: keluarkan bila melebihi
kapasitas
    }

    public static void main(String[] args) {
        LruCache<String, String> receipts = new LruCache<>(3);
        receipts.put("L-001", "Sari 24,500");
        receipts.put("L-002", "Budi 75,000");
        receipts.put("L-003", "Rina 10,000");
        System.out.println(receipts.keySet());
        receipts.get("L-001");             // L-001 menjadi yang terakhir dipakai
        System.out.println("after get L-001  " + receipts.keySet());
        receipts.put("L-004", "Sari 15,750");
        System.out.println("after put L-004  " + receipts.keySet() + " (L-002 evicted)");
        System.out.println("get L-002 -> " + receipts.get("L-002"));
    }
}
```

```
[L-001, L-002, L-003]
after get L-001  [L-002, L-003, L-001]
after put L-004  [L-003, L-001, L-004] (L-002 evicted)
get L-002 -> null
```

Kode 9.40 `lru_cache.py`: cache yang sama dengan `OrderedDict` dan `move_to_end`.

```python
"""lru_cache.py: cache yang melupakan entri yang paling lama tidak dipakai (LRU) saat
penuh."""
from collections import OrderedDict
from typing import Generic, Hashable, Optional, TypeVar

K = TypeVar("K", bound=Hashable)
V = TypeVar("V")


class LruCache(Generic[K, V]):
    def __init__(self, capacity: int) -> None:
        self._capacity = capacity
        self._data: OrderedDict[K, V] = OrderedDict()

    def get(self, key: K) -> Optional[V]:
        if key not in self._data:
            return None
        self._data.move_to_end(key)          # kini yang terakhir dipakai
        return self._data[key]

    def put(self, key: K, value: V) -> None:
        self._data[key] = value
        self._data.move_to_end(key)
        if len(self._data) > self._capacity:
            self._data.popitem(last=False)   # keluarkan yang paling lama tidak dipakai

    def keys(self) -> list[K]:
        return list(self._data)


def main() -> None:
    receipts: LruCache[str, str] = LruCache(3)
    receipts.put("L-001", "Sari 24,500")
    receipts.put("L-002", "Budi 75,000")
    receipts.put("L-003", "Rina 10,000")
    print(receipts.keys())
    receipts.get("L-001")
    print("after get L-001 ", receipts.keys())
    receipts.put("L-004", "Sari 15,750")
    print("after put L-004 ", receipts.keys(), "(L-002 evicted)")
    print("get L-002 ->", receipts.get("L-002"))


if __name__ == "__main__":
    main()
```

```
['L-001', 'L-002', 'L-003']
after get L-001  ['L-002', 'L-003', 'L-001']
after put L-004  ['L-003', 'L-001', 'L-004'] (L-002 evicted)
get L-002 -> None
```

Konstruktor `LinkedHashMap(16, 0.75f, true)` menetapkan kapasitas awal, faktor muatan dari Persamaan (9.4), dan, dengan `true`, *urutan akses*: setiap `get` atau `put` memindahkan entri ke ujung list tertaut internal, sehingga kepala list selalu yang paling lama tidak dipakai. Setelah setiap `put`, `LinkedHashMap` memanggil `removeEldestEntry(head)`, dan override kita menjawab "ya, hapuslah" bila ukurannya melampaui kapasitas. Jejaknya: setelah tiga put urutannya `L-001, L-002, L-003`; `get("L-001")` memindahkannya ke ujung; `put("L-004")` membuat ukurannya 4, sehingga kepala list, `L-002`, dikeluarkan. Memperluas kelas konkret dapat diterima di sini karena `LinkedHashMap` memang *dirancang* untuk itu: `removeEldestEntry` adalah kait (*hook*) `protected` yang terdokumentasi dan satu-satunya tujuannya adalah di-override, dan dokumentasinya memperlihatkan persis pemakaian ini [5]. Masalah kelas dasar yang rapuh dari Bab 7 muncul ketika subkelas bergantung pada bagian dalam yang tidak terdokumentasi. Di Python,

`OrderedDict.move_to_end(key)` menandai sebuah entri sebagai yang terbaru dan `popitem(last=False)` membuang yang tertua. (Untuk sebuah *fungsi*, `functools.lru_cache` dari **Kode 9.4** mengerjakan semua ini untuk Anda.)

Penyelesaian 9.11. Tambahkan pada `MultiIndex` di Java

```
void remove(K key, V value) {
    List<V> list = map.get(key);
    if (list != null && list.remove(value) && list.isEmpty()) {
        map.remove(key);                  // tidak ada list kosong yang tertinggal
    }
}
```

dan pada `OrderBook`

```
void cancel(String id) {
    LaundryOrder o = find(id);
    if (o.getStatus() != Status.RECEIVED) {
        throw new IllegalStateException(id + " is " + o.getStatus() + " and cannot be
cancelled");
    }
    washQueue.remove(o);                  // O(n): heap tidak dapat menemukan o lebih
cepat
    byStatus.get(Status.RECEIVED).remove(o);
    byCustomer.remove(o.getCustomer(), o);
    byId.remove(id);
}
```

Di Python, `MultiIndex.remove` mengerjakan hal yang sama dengan `values.remove(value)` dan `del self._map[key]`, dan `cancel` membangun ulang heap tanpa pesanan itu: `self._wash_queue = [e for e in self._wash_queue if e[2] is not o]`, lalu `heapq.heapify(self._wash_queue)`, diikuti ketiga penghapusan. Menambahkan `book.cancel("L-006")` dan percobaan membatalkan `L-004` sesudah penolakan-penolakan di `main` mencetak, dalam kedua bahasa,

```
refused: L-004 is READY and cannot be cancelled
after cancel: queue [L-001, L-005, L-008], Budi [L-002]
```

dan invariannya tetap berlaku, dengan `RECEIVED` turun menjadi 3. `PriorityQueue.remove(o)` harus lebih dulu *menemukan* `o`, dan heap hanya terurut di sepanjang setiap lintasan dari akar, sehingga pencariannya linear, $O(n)$, diikuti perbaikan $O(\log n)$; `heapify` juga $O(n)$. *Penghapusan malas* menghindari pencarian itu: simpan himpunan id yang dibatalkan, biarkan entrinya di dalam heap, dan ketika `startNextWash` mengambil entri yang id-nya ada di himpunan itu, buang lalu ambil lagi. Setiap entri yang dibatalkan dengan demikian dihapus tepat sekali, dengan biaya $O(\log n)$, ketika mencapai puncak.

Penyelesaian 9.12. mypy melaporkan

```
weights.py:16: error: Argument 1 to "total_weight" has incompatible type
"list[ExpressOrder]"; expected "list[Order]"  [arg-type]
weights.py:16: note: "list" is invariant -- see
https://mypy.readthedocs.io/en/stable/common_issues.html#variance
weights.py:16: note: Consider using "Sequence" instead, which is covariant
Found 1 error in 1 file (checked 1 source file)
```

`ExpressOrder` adalah subtipe `Order`, tetapi menurut Persamaan (9.7) `list[ExpressOrder]` bukan subtipe `list[Order]`. Fungsi itu hanya membaca, jadi deklarasikan `def total_weight(orders: Sequence[Order]) -> float` (dari `collections.abc` atau `typing`); `Sequence` bersifat kovarian dan mypy menerima panggilannya. Keberatan itu beralasan untuk badan fungsi seperti `orders.append(Order(1.0))`: dengan `list[Order]` badan itu lolos pemeriksaan di dalam fungsi,

dan seandainya `list[ExpressOrder]` diterima, sebuah `Order` biasa akan berakhir di dalam list yang menjanjikan hanya pesanan kilat — anjing di antara kucing-kucing pada **Kode 9.26**.

Daftar Pustaka

[1] J. Gosling, B. Joy, G. Steele, G. Bracha, A. Buckley, D. Smith, and G. Bierman, *The Java Language Specification, Java SE 21 Edition*. Oracle America, 2023. [Online]. Available: https://docs.oracle.com/javase/specs/jls/se21/html/

[2] J. Bloch, *Effective Java*, 3rd ed. Boston, MA, USA: Addison-Wesley, 2018.

[3] The mypy project, *mypy documentation*: "Generics" and "Common issues and solutions: variance". [Online]. Available: https://mypy.readthedocs.io/

[4] M. T. Goodrich, R. Tamassia, and M. H. Goldwasser, *Data Structures and Algorithms in Java*, 6th ed. Hoboken, NJ, USA: Wiley, 2014.

[5] Oracle, *Java Platform, Standard Edition 21 API Specification*, package `java.util`, and "The Collections Framework" overview. [Online]. Available: https://docs.oracle.com/en/java/javase/21/docs/api/

[6] S. Marks, "JEP 431: Sequenced Collections," OpenJDK, 2023 (delivered in JDK 21). [Online]. Available: https://openjdk.org/jeps/431

[7] Python Software Foundation, *The Python Standard Library*, version 3.14: `collections`, `collections.abc`, `heapq`, `bisect`, `typing`; and "Time complexity" of built-in operations. [Online]. Available: https://docs.python.org/3/library/

[8] L. Ramalho, *Fluent Python*, 2nd ed. Sebastopol, CA, USA: O'Reilly Media, 2022.

[9] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 4th ed. Cambridge, MA, USA: MIT Press, 2022.

[10] L. J. Guibas and R. Sedgewick, "A dichromatic framework for balanced trees," in *Proc. 19th Annu. Symp. Foundations of Computer Science (FOCS)*, Ann Arbor, MI, USA, 1978, pp. 8–21.

[11] G. Bracha, "Generics in the Java programming language," Sun Microsystems tutorial, 2004. [Online]. Available: https://www.oracle.com/technetwork/java/javase/generics-tutorial-159168.pdf

[12] E. Traut, "PEP 695 – Type parameter syntax," Python Enhancement Proposal 695, 2022. [Online]. Available: https://peps.python.org/pep-0695/

[13] M. Naftalin and P. Wadler, *Java Generics and Collections*. Sebastopol, CA, USA: O'Reilly Media, 2006.

[14] T. Peters, "listsort.txt: description of Timsort," CPython source distribution, 2002. [Online]. Available: https://github.com/python/cpython/blob/main/Objects/listsort.txt

[15] G. van Rossum, J. Lehtosalo, and Ł. Langa, "PEP 484 – Type hints," Python Enhancement Proposal 484, 2014. [Online]. Available: https://peps.python.org/pep-0484/

Bab 10

Antarmuka Grafis dan Penanganan Peristiwa

Pada akhir bab ini Anda akan sanggup membangun jendela sungguhan, di Java dengan Swing dan di Python dengan Tkinter; menjelaskan lapisan-lapisan di bawahnya (AWT, Swing, Tcl/Tk); menata komponen dengan *layout manager* serta dengan `pack` dan `grid`; menggambar bentuk Anda sendiri di `paintComponent` dan di sebuah `Canvas`; menjelaskan dari nol apa itu *gelung peristiwa* (*event loop*); membuat program menanggapi klik, tombol papan ketik, kotak centang, dan gerakan tetikus melalui *listener* dan *callback*; menulis listener itu sebagai kelas dalam, kelas anonim, lambda, dan referensi metode di Java, serta sebagai *bound method*, *closure*, lambda, dan objek `partial` di Python, sambil tahu persis apa yang diingat masing-masing; menaati satu aturan yang dipaksakan setiap toolkit GUI terhadap thread; dan menguji jendela di mesin yang sama sekali tidak memiliki layar. Semuanya bertemu dalam jendela pencatatan pesanan yang lengkap untuk Profitina Laundry, ditulis dalam Swing dan dalam Tkinter, dengan formulir, validasi, harga yang langsung diperbarui, daftar pesanan, dan total, serta dengan uji mandiri yang membuktikan bahwa kedua versi berperilaku identik.

10.1 Di Mana Kita Berhenti

Setiap program dalam buku ini sejauh ini memiliki bentuk yang sama. Ia mulai dari puncak `main`, mengerjakan tugasnya dalam urutan yang ditulis pemrogram, mencetak beberapa baris, lalu berhenti. Bab 5 memungkinkan program membaca masukan, tetapi bahkan di sana programlah yang memegang kendali: ia mencetak sebuah *prompt*, *menunggu* pengguna mengetik satu baris, lalu melanjutkan. Bab 7 memberi Profitina Laundry satu keluarga kelas layanan yang harganya dihitung secara polimorfis, dan Bab 9 menyimpan banyak objek di dalam koleksi serta membuat koleksi itu aman-tipe dengan generik.

Namun, staf di meja laundry tidak memakai terminal. Mereka memakai jendela: sebuah formulir dengan kotak untuk mengetik nama pelanggan dan berat cucian, daftar layanan yang dapat dibuka ke bawah, kotak centang untuk layanan kilat, tombol bertuliskan *Add order*, dan daftar yang menampilkan pesanan hari itu dengan total di bagian bawah. Mereka mengeklik di mana pun mereka suka, dalam urutan apa pun yang mereka suka. Mereka bisa mengetik berat sebelum nama. Mereka bisa mengganti layanan tiga kali sebelum menambahkan pesanan. Mereka bisa menghapus pesanan yang keliru ditambahkan. Program tidak dapat mencetak *prompt* lalu menunggu satu jawaban tertentu, karena ia sama sekali tidak tahu apa yang akan dilakukan pengguna berikutnya.

Satu kenyataan itu — *penggunalah, bukan program, yang menentukan apa yang terjadi berikutnya* — mengubah cara sebuah program disusun, dan itulah pokok bahasan bab ini. Kita akan berkenalan dengan kosakata baru (komponen, kontainer, tata letak, peristiwa, listener, callback, gelung peristiwa), mempelajari dua toolkit nyata yang mewujudkannya (Swing untuk Java, Tkinter untuk Python), dan di sepanjang jalan menemukan tiga fitur bahasa yang membuat kode berbasis peristiwa enak ditulis: kelas dalam, kelas anonim, dan lambda di Java, serta *closure* di

Python. Fitur-fitur ini berguna jauh melampaui urusan jendela; Anda akan bertemu lagi dengannya di Bab 13 (iterator) dan Bab 14 (thread).

> **WAWASAN KUNCI:** Program konsol *memanggil* pengguna ("ketikkan beratnya sekarang"). Program GUI *dipanggil oleh* pengguna: ia menyiapkan jendela, berkata "bila tombol ini diklik, jalankan metode itu", lalu menyerahkan kendali kepada sebuah gelung yang menunggu sesuatu terjadi. Segala hal dalam bab ini bersumber dari pembalikan itu.

10.2 Dari Konsol ke Jendela: Siapa yang Memegang Kendali?

Mari kita buat pembalikan itu konkret. Beginilah versi konsol dari pencatatan pesanan laundry akan berjalan, ditulis sebagai percakapan yang dilakukannya dengan pengguna:

```
Customer name: Siti Aminah
Phone: 0812-3456-7890
Service (WF/IR/WDI): WF
Weight in kg: 4,2
Express (y/n): n
Added ORD-0001, Rp31,500. Another order (y/n)?
```

Program mengajukan lima pertanyaan dalam urutan yang tetap. Bila Siti berubah pikiran soal layanan kilat setelah beratnya diketik, tidak ada jalan kembali: pertanyaan itu sudah lewat. Bila beratnya salah ketik, program harus memulai seluruh pesanan dari awal atau memuat kode khusus untuk menanyakan ulang satu pertanyaan itu. Urutan percakapan tertanam di dalam kode, sebagai urutan pernyataan.

Sekarang bayangkan tugas yang sama sebagai jendela, yang diperlihatkan **Gambar 10.18** di Subbab 10.12 persis seperti tampak di layar. Kelima pertanyaan terlihat sekaligus, masing-masing di kotaknya sendiri. Pengguna mengisinya dalam urutan apa pun, membetulkan yang mana pun kapan pun, dan melihat harga berubah seketika berat atau layanan berubah. Baru ketika pengguna mengeklik *Add order* program memeriksa seluruh formulir, lalu menerima pesanan atau menjelaskan, dengan warna merah, apa yang keliru.

Program yang menggerakkan jendela seperti itu sama sekali tidak memuat percakapan. Sebaliknya, ia memuat banyak potongan kode kecil, masing-masing menjawab satu pertanyaan berbentuk "apa yang harus terjadi ketika *ini* terjadi?":

- ketika teks di kotak berat berubah, hitung ulang label harga;
- ketika layanan lain dipilih, hitung ulang label harga;
- ketika kotak kilat dicentang atau dilepas, hitung ulang label harga;
- ketika *Add order* diklik, validasi formulir, lalu tambahkan pesanan atau tampilkan galat;
- ketika sebuah baris daftar dipilih, aktifkan tombol *Remove selected*;
- ketika *Remove selected* diklik, hapus pesanan itu dan perbarui total.

Setiap kejadian ini adalah sebuah **peristiwa** (*event*), dan setiap potongan kode kecil itu adalah **penangan peristiwa** (*event handler*). Program utama hanya membangun jendela, menghubungkan setiap penangan dengan peristiwa yang ditanganinya, lalu melepaskan kendali kepada toolkit, yang berjalan terus (sampai jendela ditutup), menunggu peristiwa dan memanggil penangan yang tepat untuk masing-masing. Gaya program seperti ini disebut **berbasis peristiwa** (*event-driven*), dan pembalikan yang dikandungnya — pustaka yang memanggil kode Anda, bukan kode Anda yang memanggil pustaka — dikenal sebagai **pembalikan kendali** (*inversion of control*) [1].

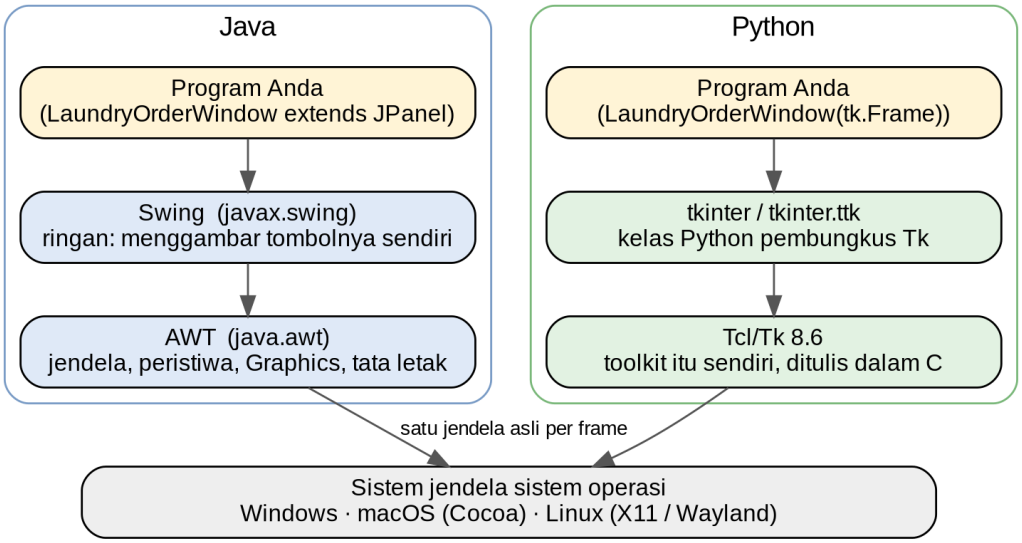
> **DEFINISI. Antarmuka pengguna grafis** (*graphical user interface*, GUI) menyajikan masukan dan keluaran program sebagai unsur visual — jendela, tombol, kotak teks, daftar, gambar — yang dioperasikan pengguna dengan tetikus, papan ketik, atau layar sentuh. **Peristiwa** (*event*) adalah apa pun yang terjadi dan mungkin ingin ditanggapi program: klik, penekanan tombol papan ketik, gerakan tetikus, jendela yang diubah ukurannya atau tersingkap, pewaktu yang berbunyi. **Penangan peristiwa** (disebut **listener** di Java dan **callback** di Python) adalah metode atau fungsi yang dipanggil toolkit ketika peristiwa tertentu terjadi. Program yang disusun sebagai sekumpulan penangan yang menunggu peristiwa adalah program **berbasis peristiwa**.

Tiga akibat langsung menyusul, dan kita akan membahas masing-masing dengan layak nanti di bab ini.

1. **Penangan harus singkat.** Selama satu penangan berjalan, tidak ada peristiwa lain yang dapat ditangani: jendela bahkan tidak dapat menggambar ulang dirinya. Penangan yang makan waktu tiga detik membekukan jendela selama tiga detik. Subbab 10.11 menunjukkan cara memindahkan pekerjaan lambat ke tempat lain.
2. **Keadaan tinggal di dalam objek, bukan di posisi dalam kode.** Program konsol tahu bahwa "beratnya sudah diketik" karena eksekusi sudah melewati baris yang membacanya. Program GUI mengetahuinya hanya karena ada objek yang mengingatnya — teks sebuah field teks, atau field kelas Anda sendiri. Itulah sebabnya GUI dan objek begitu serasi, dan mengapa setiap jendela dalam bab ini adalah sebuah kelas.
3. **Jendela harus dideskripsikan, bukan dikisahkan.** Alih-alih urutan *prompt*, program mendeklarasikan sebuah pohon komponen dan cara penataannya. Subbab 10.5 dan 10.8 membahas deskripsi itu.

10.3 Tiga Toolkit: AWT, Swing, dan Tkinter

Toolkit GUI adalah pustaka berisi balok-balok visual siap pakai — jendela, tombol, field teks, daftar — beserta mesin yang mengantarkan peristiwa kepadanya dan menggambarnya di layar. Pada prinsipnya Anda bisa menggambar setiap tombol piksel demi piksel sendiri, tetapi tidak ada yang melakukannya: toolkit memberi Anda tombol yang sudah tahu cara tampak tertekan, cara menunjukkan fokus papan ketik, dan cara memberi tahu Anda bahwa ia diklik. Bab ini memakai dua toolkit yang dibawa oleh bahasanya sendiri, sehingga setiap program berjalan di instalasi baru tanpa perlu mengunduh apa pun lagi. **Gambar 10.1** memperlihatkan bagaimana lapisan-lapisannya bertumpuk.



Gambar 10.1 Lapisan di bawah sebuah jendela. Di Java, panel Anda dibangun dari komponen Swing, yang bertumpu pada AWT, yang meminta sebuah jendela asli kepada sistem operasi. Di Python, `tkinter` adalah lapisan tipis kelas Python di atas toolkit Tcl/Tk.

10.3.1 AWT: toolkit yang pertama

Pustaka GUI pertama Java adalah **Abstract Window Toolkit**, paket `java.awt`, yang berasal dari JDK 1.0. Di AWT setiap tombol adalah tombol sungguhan milik sistem operasi, sebuah *native peer*: tombol Windows di Windows, tombol Mac di Mac. Tampilannya wajar, tetapi portabilitasnya terganggu. Sistem yang berbeda menawarkan widget yang berbeda dengan perilaku yang berbeda, sehingga AWT hanya dapat menawarkan apa yang dimiliki bersama oleh semua sistem, dan program tetap berperilaku sedikit berbeda dari satu sistem ke sistem lain. Komponen yang bergantung pada sumber daya asli seperti ini disebut **heavyweight** (berbobot berat).

AWT tidak usang. Ia tetap menyediakan fondasi tempat segala hal lain berdiri: jendela yang sungguh-sungguh dibuka sistem operasi, kelas-kelas yang mendeskripsikan peristiwa (`ActionEvent`, `MouseEvent`, ...) dan antarmuka listener yang menerimanya (paket `java.awt.event`), kelas `Graphics` untuk menggambar, warna dan fon, serta *layout manager*. Itulah sebabnya hampir setiap program Swing dimulai dengan mengimpor dari `java.awt` sekaligus `javax.swing`.

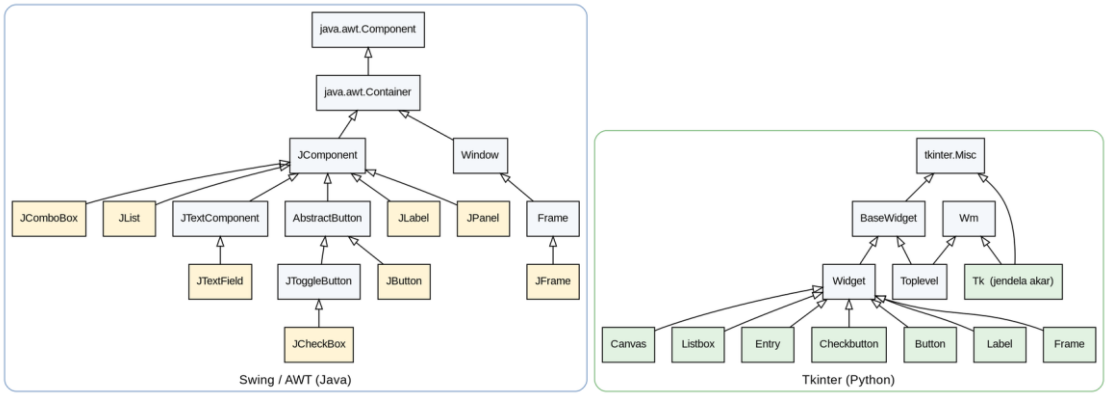
10.3.2 Swing: komponen yang menggambar dirinya sendiri

Pada akhir 1996 Sun Microsystems memulai **Java Foundation Classes** (JFC), yang bagian GUI-nya dirilis bersama Java 2 (JDK 1.2) sebagai paket `javax.swing`. *Swing* bukan singkatan; itu nama sandi proyeknya, dan nama itu melekat. Komponen Swing bersifat **lightweight** (berbobot ringan): sebuah `JButton` sama sekali bukan tombol asli. Swing meminta satu jendela asli per frame kepada sistem operasi, lalu *melukis* sendiri setiap tombol, label, dan daftar di dalamnya, dengan kode Java dan kelas `Graphics`. Hasilnya tampak dan berperilaku sama di mana pun, dan penampilannya bahkan dapat diganti dengan menukar *look and feel*, dengan harga sedikit kecepatan yang tidak dirasakan siapa pun di mesin masa kini [1].

Karena program biasanya mengimpor `java.awt` sekaligus `javax.swing`, dan keduanya memiliki kelas untuk tombol, label, frame, dan seterusnya, Swing memberi awalan huruf **J** pada nama kelasnya: `JFrame`, `JButton`, `JLabel`, `JTextField`, `JPanel`, `JMenuBar`. Tanpa huruf J, `Button` akan bermakna ganda. Satu pegangan yang akan menghemat berjam-jam waktu Anda: **dalam program**

Swing, pakailah hanya kelas J untuk komponen; mencampurkan `Button` milik AWT ke dalam jendela Swing menimbulkan masalah penggambaran yang sulit dilacak.

Dalam kosakata Java, jendela tingkat atas yang memiliki bilah judul dan bingkai disebut **frame**, kelas `JFrame`. Leluhur kelas-kelas yang akan Anda pakai diperlihatkan di sisi kiri **Gambar 10.2**. Ada dua hal yang layak diperhatikan di situ. Pertama, `JFrame` diturunkan dari `Window` dan `Frame` milik AWT, karena ia *memang* jendela asli; setiap komponen Swing lainnya diturunkan dari `JComponent`, cabang yang ringan. Kedua, setiap komponen, termasuk `JPanel`, juga merupakan `Container` — sesuatu yang dapat menampung komponen lain — dan begitulah jendela dibangun dari kotak-kotak yang bersarang.



Gambar 10.2 Sebagian hierarki kelas Swing/AWT (kiri) dan Tkinter (kanan). Kelas yang diarsir adalah yang dipakai bab ini. Panah menunjuk dari subkelas ke superkelasnya.

Pustaka GUI Java yang lain juga ada: IDE Eclipse dari Bab 2 dibangun dengan SWT, yang kembali ke widget asli, dan JavaFX adalah toolkit yang didistribusikan terpisah (OpenJFX) dan dirilis setiap enam bulan seperti JDK; JavaFX 27, versi terkini pada September 2026, memerlukan JDK 25 atau yang lebih baru [2]. Swing adalah bagian dari setiap JDK, termasuk Java 25, rilis dukungan jangka panjang (LTS) yang berlaku sekarang, dan Java 27 yang dirilis pada 15 September 2026 [3], dan karena itulah buku ini memakainya. Anda mungkin menjumpai **applet**, frame kecil yang disematkan di halaman web, dalam buku-buku lama; peramban sudah lama menghentikan dukungan plug-in Java, API Applet ditandai usang untuk dihapus sejak Java 17 [4], dan JDK 26 menghapusnya sama sekali [5], sehingga applet kini bahkan tidak dapat dikompilasi lagi.

10.3.3 Tkinter: toolkit standar Python

Pustaka standar Python memuat `tkinter`, antarmuka Python ke **Tk**, sebuah toolkit yang ditulis dalam C pada awal 1990-an oleh John Ousterhout untuk bahasa skripnya, Tcl [6]. Ketika program Python membuat sebuah `tkinter.Button`, objek Python itu mengirim perintah ke interpreter Tcl yang tertanam, dan Tk-lah yang membuat serta menggambar tombol yang sebenarnya. Seperti Swing, Tk menggambar sendiri sebagian besar widgetnya dan berjalan di Windows, macOS, dan Linux. Modul `tkinter.ttk` menambahkan widget *bertema* (seperti `Combobox` yang akan kita pakai untuk daftar layanan) yang lebih mengikuti tampilan sistem tuan rumah [7].

Di sisi kanan **Gambar 10.2** Anda dapat melihat bahwa rancangan Tkinter lebih datar daripada Swing. Setiap kelas widget (`Label`, `Button`, `Entry`, `Frame`, `Canvas`, …) mewarisi `Widget`, yang menyediakan metode-metode bersama — `pack`, `grid`, `bind`, `config`, `destroy`, dan banyak lagi — melalui kelas *mixin* `Misc`. Jendela akar, kelas `Tk`, istimewa: hanya ada satu per program, ia memiliki interpreter Tcl, dan setiap widget lain dibuat *di dalam* suatu induk, yang selalu menjadi argumen pertama konstruktornya.

> **PERINGATAN:** Tkinter adalah bagian dari pustaka standar, tetapi di beberapa sistem ia dikemas terpisah. Penginstal dari python.org untuk Windows dan macOS sudah memuatnya. Di Debian dan Ubuntu Linux, pasanglah dengan `sudo apt install python3-tk`; di Fedora, `sudo dnf install python3-tkinter`. Bila `python3 -c "import tkinter"` mencetak `ModuleNotFoundError: No module named 'tkinter'`, inilah penyebabnya, bukan kode Anda.

Tabel 10.1 memasangkan komponen kedua toolkit yang dipakai bab ini. Namanya berbeda; gagasannya sama.

Tabel 10.1 Komponen Swing yang umum dan padanannya di Tkinter.

Apa itu	Swing (Java)	Tkinter (Python)
Jendela tingkat atas	`JFrame`	`Tk` (akar), `Toplevel` (lainnya)
Kontainer polos	`JPanel`	`Frame`, `LabelFrame`
Teks atau gambar tetap	`JLabel`	`Label`
Tombol tekan	`JButton`	`Button`
Masukan teks satu baris	`JTextField`	`Entry`
Kotak centang	`JCheckBox`	`Checkbutton`
Satu dari beberapa (radio)	`JRadioButton` + `ButtonGroup`	`Radiobutton` yang berbagi satu variabel
Pilihan tarik-turun	`JComboBox<E>`	`ttk.Combobox`
Daftar baris	`JList<E>` + `DefaultListModel<E>`	`Listbox`
Penggulungan	`JScrollPane` di sekeliling komponen	`Scrollbar` yang ditautkan ke widget
Gambar bebas	subkelas `JPanel` yang meng-*override* `paintComponent`	`Canvas` dengan item gambar
Menu	`JMenuBar`, `JMenu`, `JMenuItem`	`Menu`
Tab	`JTabbedPane`	`ttk.Notebook`

Katalog setiap toolkit jauh lebih panjang. Selain unsur tampilan (label, gambar) dan kontrol dasar (field, tombol, tombol radio, kotak centang, combo box, menu, penggeser), ada kontainer (panel, *scroll pane*, *split pane*, *tabbed pane*, bilah alat), tampilan canggih (tabel dan pohon), dan kontrol canggih (pemilih berkas dan pemilih warna). Tutorial resmi Swing [1] dan dokumentasi `tkinter` [7] mendaftar semuanya; begitu Anda memahami gagasan bab ini, setiap komponen baru hanyalah soal membaca halaman rujukannya.

10.4 Jendela Pertama Anda

Kita mulai dengan jendela terkecil yang berguna: sebuah frame dengan judul dan satu label yang menyapa. **Kode 10.1** adalah versi Swing-nya.

Kode 10.1 `FirstWindow.java`: frame dengan satu label, dibangun dalam enam langkah, dengan mode `--selftest` yang berjalan tanpa layar.

```
import java.awt.BorderLayout;
import java.awt.Container;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.SwingUtilities;

public class FirstWindow {
    public static void main(String[] args) throws Exception {
        if (args.length > 0 && args[0].equals("--selftest")) {
            System.setProperty("java.awt.headless", "true");  // no screen needed
            SwingUtilities.invokeAndWait(FirstWindow::selfTest);
        } else {
            SwingUtilities.invokeLater(FirstWindow::createAndShow);
        }
    }

    static JLabel buildLabel() {
        return new JLabel("Hello, Profitina Laundry!");
    }

    static void createAndShow() {
        // 1. Opsional: biarkan Swing menggambar dekorasi jendela
        JFrame.setDefaultLookAndFeelDecorated(true);
        // 2. Buat frame
        JFrame frame = new JFrame("First window");
        // 3. Opsional: apa yang terjadi saat frame ditutup?
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        // 4. Buat komponen dan masukkan ke dalam frame
        frame.getContentPane().add(buildLabel(), BorderLayout.CENTER);
        // 5. Ukur frame agar pas dengan isinya
        frame.pack();
        // 6. Tampilkan
        frame.setVisible(true);
    }

    static void selfTest() {
        Container content = new Container();  // pengganti content pane
        content.setLayout(new BorderLayout());
        JLabel label = buildLabel();
        content.add(label, BorderLayout.CENTER);
        System.out.println("headless: " + java.awt.GraphicsEnvironment.isHeadless());
        System.out.println("components: " + content.getComponentCount());
        System.out.println("label text: " + label.getText());
        System.out.println("on the EDT: " + SwingUtilities.isEventDispatchThread());
        System.out.println("self-test passed");
    }
```

Kode 10.1 (Lanjutan) *FirstWindow.java: frame dengan satu label, dibangun dalam enam langkah, dengan mode --selftest yang berjalan tanpa layar.*

```
}
```

Metode `createAndShow` memuat enam langkah yang dilakukan hampir setiap program Swing, bernomor di komentar:

1. `JFrame.setDefaultLookAndFeelDecorated(true)` bersifat opsional. Ia meminta Swing menggambar sendiri bilah judul dan bingkai frame, dengan gaya Swing, alih-alih membiarkan sistem operasi menggambarnya. Jendela-jendela dalam gambar bab ini yang bilah judulnya biru berpola dibuat dengan cara ini.
2. `new JFrame("First window")` membuat objek frame, dengan judul yang diberikan. Membuatnya tidak berarti menampilkannya.
3. `setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE)` menyatakan apa yang harus terjadi ketika pengguna mengeklik tombol tutup: akhiri program. Bawaannya hanyalah

menyembunyikan frame, sehingga program tetap berjalan tanpa terlihat, tanpa jendela yang bisa ditutup — program tampak macet. Tulislah baris ini untuk setiap jendela utama.

4. `frame.getContentPane().add(buildLabel(), BorderLayout.CENTER)` memasukkan sebuah komponen ke frame. `JFrame` tidak menampung komponen secara langsung; ia menampung beberapa *pane*, dan yang menampung komponen Anda adalah **content pane**. Subbab 10.5 menjelaskan pane-pane itu dan argumen `BorderLayout.CENTER`.
5. `frame.pack()` mengukur frame agar pas cukup besar untuk isinya pada *ukuran yang disukai* (*preferred size*) masing-masing. Alternatifnya, `setSize(width, height)`, mematok ukuran dalam piksel berapa pun yang dibutuhkan isinya; `pack()` hampir selalu lebih baik, karena ukuran yang disukai sebuah label bergantung pada fonnya, yang berbeda dari satu komputer ke komputer lain.
6. `frame.setVisible(true)` menampilkan frame. Sebelum panggilan ini, jendela hanya ada sebagai objek di memori.

Sekarang perhatikan `main`. Ia tidak memanggil `createAndShow()` secara langsung. Ia menyerahkan metode itu kepada `SwingUtilities.invokeLater`, yang berarti "jalankan ini, segera, di thread yang menangani GUI". Thread itu, **event dispatch thread** (EDT, thread penyalur peristiwa), adalah pokok Subbab 10.11; untuk sekarang terimalah sebagai aturan bahwa `main` setiap program Swing membuat jendelanya dengan cara ini. Setelah `invokeLater` kembali, `main` sendiri berakhir — namun program tetap berjalan, karena EDT terus berjalan selama masih ada jendela yang terbuka. Itulah tanda pertama yang terlihat dari pembalikan kendali: `main` bukan lagi tempat program hidup.

Jendela yang ditampilkannya mungil:

```
+-- First window ------------[_][O][X]+
| Hello, Profitina Laundry!           |
+-------------------------------------+
```

Sebuah bilah judul dengan judul dan tiga tombol yang biasa, dan di bawahnya label, rata kiri dan di tengah secara vertikal, di dalam frame yang dibuat `pack()` pas cukup lebar untuknya.

10.4.1 Menjalankan jendela tanpa layar: uji mandiri

Buku ini tidak dapat memperlihatkan jendela yang hidup kepada Anda, begitu pula pengujian otomatis yang berjalan di server tanpa monitor. Maka setiap program GUI dalam bab ini memiliki mode kedua. Bila dijalankan dengan argumen `--selftest`, ia membangun komponen jendela persis seperti program sebenarnya, mengendalikannya dari kode, mencetak apa yang ditemukannya, dan sama sekali tidak pernah membuka jendela. Untuk **Kode 10.1**, perintah `java FirstWindow --selftest` mencetak:

```
headless: true
components: 1
label text: Hello, Profitina Laundry!
on the EDT: true
self-test passed
```

Baris pertama `main` dalam mode uji mandiri, `System.setProperty("java.awt.headless", "true")`, mengalihkan AWT ke **mode headless**: mode untuk komputer tanpa layar. Dalam mode *headless* Anda boleh membuat dan memakai komponen ringan apa pun — label, panel, tombol, field teks, daftar — tetapi apa pun yang memerlukan jendela asli sungguhan, seperti `JFrame`, melempar `HeadlessException`. Itulah sebabnya uji mandiri memasukkan label ke sebuah `Container` polos yang menggantikan content pane, bukan ke dalam frame. Properti itu harus disetel sebelum kelas AWT pertama dipakai, dan karena itu ia menjadi pernyataan paling awal.

Cobalah menjalankan program *secara normal* di mesin tanpa layar (misalnya melalui SSH, atau di dalam kontainer) dan Anda akan melihat apa yang dihindari uji mandiri:

```
Exception in thread "AWT-EventQueue-0" java.awt.HeadlessException:
No X11 DISPLAY variable was set,
or no headful library support was found,
but this program performed an operation which requires it.
```

Perhatikan nama thread dalam pesan itu, `AWT-EventQueue-0`: eksepsi terjadi di event dispatch thread, tempat `invokeLater` mengirim `createAndShow` kita. Uji mandiri memakai `invokeAndWait` sebagai gantinya, yang menjalankan kode di EDT *dan menunggu sampai selesai*, sehingga `main` tidak berakhir sebelum pengujian mencetak hasilnya. Baris `on the EDT: true` memastikan di mana kode itu berjalan.

Kode 10.2 adalah program yang sama dalam Tkinter.

Kode 10.2 `first_window.py`: jendela yang sama dalam Tkinter, dengan mode `--selftest` yang sama.

```python
"""first_window.py: jendela Tkinter terkecil yang berguna."""
import sys
import tkinter as tk


def build(root):
    root.title("First window")
    label = tk.Label(root, text="Hello, Profitina Laundry!", padx=20, pady=10)
    label.pack()
    return label


def self_test():
    root = tk.Tk()
    root.withdraw()  # bangun jendelanya, tetapi jangan pernah tampilkan
    label = build(root)
    root.update_idletasks()  # biarkan geometry manager bekerja
    print("title:", root.title())
    print("children:", [w.winfo_class() for w in root.winfo_children()])
    print("label text:", label.cget("text"))
    print("mapped on screen:", bool(root.winfo_ismapped()))
    root.destroy()
    print("self-test passed")


def main():
    if "--selftest" in sys.argv:
        self_test()
        return
    root = tk.Tk()
    build(root)
    root.mainloop()  # serahkan kendali ke gelung peristiwa sampai jendela ditutup


if __name__ == "__main__":
    main()
```

Strukturnya berpadanan satu lawan satu. `tk.Tk()` membuat jendela akar (frame-nya Swing), `root.title(...)` menyetel judul, `tk.Label(root, text=...)` membuat label *di dalam* akar — induk selalu menjadi argumen pertama — dan `label.pack()` menyerahkan label kepada **geometry manager pack**, yang menempatkannya dan mengukur jendela agar pas, mirip `pack()` di Swing. Tidak ada langkah 3: menutup jendela akar Tkinter mengakhiri `mainloop()`, dan bersamanya program. Dan baris terakhir `main`, `root.mainloop()`, adalah tempat Python

menyerahkan kendali ke gelung peristiwa Tk, padanan EDT milik Swing. `mainloop()` baru kembali ketika jendela ditutup.

Menjalankan `python3 first_window.py --selftest` mencetak:

```
title: First window
children: ['Label']
label text: Hello, Profitina Laundry!
mapped on screen: False
self-test passed
```

Uji mandiri memanggil `root.withdraw()` tepat setelah membuat akar, yang menyingkirkan jendela dari layar sebelum sempat digambar. `update_idletasks()` membiarkan Tk menyelesaikan urusan pembukuan yang tertunda (termasuk perhitungan geometri) tanpa memproses peristiwa pengguna apa pun. Baris `mapped on screen: False` memastikan bahwa tidak ada yang ditampilkan.

Ada satu perbedaan antara kedua toolkit yang penting untuk pengujian. Swing memiliki mode *headless* yang sesungguhnya. Tk tidak: bahkan jendela yang disembunyikan memerlukan sambungan ke sebuah *display*, dan di mesin yang tidak memilikinya, `tk.Tk()` gagal dengan

```
_tkinter.TclError: no display name and no $DISPLAY environment variable
```

Di server Linux obat bakunya adalah **Xvfb**, *X virtual frame buffer*, server tampilan yang menggambar ke memori, bukan ke monitor. Perintah `xvfb-run -a python3 first_window.py --selftest` menyalakan satu, menjalankan program di atasnya, lalu mematikannya. Setiap keluaran Tkinter dalam bab ini dihasilkan dengan cara itu; di komputer Anda sendiri, yang memiliki layar, jalankan saja perintahnya tanpa `xvfb-run`.

> **WAWASAN KUNCI:** Mode uji mandiri hanya menghabiskan belasan baris dan membayar dirinya berkali-kali lipat. Ia memaksa Anda membangun jendela di dalam metode yang dapat dipanggil tanpa menampilkan apa pun, ia memungkinkan Anda memeriksa perilaku jendela dari kode, dan ia memungkinkan server pengujian (atau sebuah buku) menjalankan program GUI Anda. Bab 11 membawa gagasan ini lebih jauh dengan memisahkan jendela sepenuhnya dari logika yang ditampilkannya.

10.5 Komponen, Kontainer, dan Panel

Hal-hal yang dapat kita tampilkan di jendela GUI disebut **komponen** (Java) atau **widget** (Tk): label, tombol, field teks, tabel, dan sebagainya. Sebagian komponen adalah **kontainer**: mereka dapat menampung komponen lain, yang dapat berupa kontainer pula, dan seterusnya sampai kedalaman berapa pun. Jadi jendela bukanlah kumpulan kontrol yang datar, melainkan sebuah *pohon*: frame di akar, kontainer sebagai simpul dalam, dan kontrol sederhana sebagai daun.

10.5.1 Content pane

`JFrame` adalah kontainer dari jenis yang khusus. Ia tidak menampung komponen Anda secara langsung; komponennya adalah beberapa *pane* — sebuah *root pane*, sebuah *layered pane* untuk menu pop-up dan *tool tip*, sebuah *glass pane* tak terlihat yang dapat mencegat tetikus, dan yang penting bagi Anda, **content pane**. Content pane adalah kontainer biasa, jadi ada dua cara memakainya. Anda dapat menambahkan komponen ke content pane yang sudah dimiliki frame:

```
Container pane = frame.getContentPane();
pane.add(new JLabel("Hello world!"), BorderLayout.PAGE_START);
pane.add(new JButton("Click me"), BorderLayout.PAGE_END);
```

atau Anda dapat membangun kontainer sendiri dan memasangnya sebagai content pane yang baru:

```
JPanel pane = new JPanel(new BorderLayout());
pane.add(new JLabel("Hello world!"), BorderLayout.PAGE_START);
pane.add(new JButton("Click me"), BorderLayout.PAGE_END);
frame.setContentPane(pane);
```

Bentuk kedua itulah yang dipakai bab ini untuk setiap jendela sungguhan, karena memungkinkan isinya berupa kelas kita sendiri. Sebagai jalan pintas, `frame.add(c)` diteruskan ke content pane, sehingga `frame.add(panel)` dan `frame.getContentPane().add(panel)` melakukan hal yang sama.

10.5.2 Panel, dan mengapa setiap jendela di sini adalah kelas

Panel, kelas `JPanel`, adalah kontainer ringan yang paling sederhana: sebuah persegi panjang yang dapat menampung komponen dan juga dapat digambari (Subbab 10.9). Merupakan praktik rancangan yang baik untuk mendefinisikan setiap bagian penting jendela sebagai *kelas yang memperluas* `JPanel`, menaruh komponennya di field dan membangunnya di konstruktor:

```
public class MyPanel extends JPanel {
    int count = 0;                              // keadaannya
    JButton button = new JButton("Click me");
    JLabel label = new JLabel("This is a label");

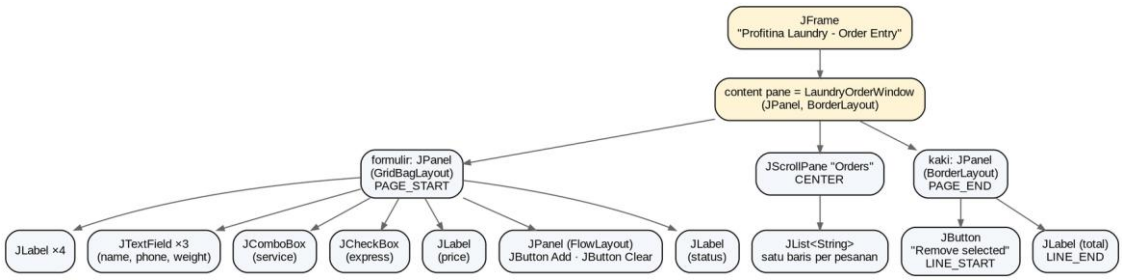
    public MyPanel() {
        add(button);                            // this.add(...): panel menambah ke
dirinya sendiri
        add(label);
    }
}
```

lalu, di `main`, memasang sebuah instansnya sebagai content pane frame dengan `frame.setContentPane(new MyPanel())`. Inilah pewarisan Bab 7 yang sedang bekerja: `MyPanel` *adalah sebuah* `JPanel`, sehingga ia mewarisi semua yang dapat dilakukan panel — ditambahkan ke frame, menata anak-anaknya, melukis dirinya — dan hanya menambahkan komponen serta keadaannya sendiri. Ketika kita perlu menggambar sendiri, memperluas `JPanel` bukan sekadar praktik yang baik melainkan keharusan, karena menggambar dilakukan dengan meng-*override* salah satu metodenya.

Program Tkinter mengikuti pola yang sama dengan `tk.Frame`: `class EchoPanel(tk.Frame)` memanggil `super().__init__(master)` lalu membuat widgetnya dengan `self` sebagai induk. Setiap panel dalam bab ini, dalam kedua bahasa, dibangun dengan cara ini.

10.5.3 Pohon komponen sebuah jendela sungguhan

Gambar 10.3 memperlihatkan pohon jendela laundry yang kita bangun di Subbab 10.12. Frame menampung content pane; content pane, sebuah `LaundryOrderWindow`, menampung tiga kontainer — formulir di atas, daftar yang dapat digulung di tengah, dan bagian kaki di bawah — dan masing-masing menampung komponennya sendiri. Formulir bahkan memuat sebuah panel kecil miliknya sendiri, sekadar untuk menjajarkan tombol *Add order* dan *Clear*. Membaca jendela sebagai pohon adalah kunci untuk menatanya: setiap kontainer menentukan posisi anak-anaknya sendiri dan tidak lebih.



Gambar 10.3 Pohon komponen jendela pencatatan pesanan Profitina Laundry. Setiap kontainer hanya menata anak-anaknya sendiri, dengan layout manager-nya sendiri.

10.6 Gelung Peristiwa, Dijelaskan dari Nol

Beberapa kali kita berkata bahwa program GUI "menyerahkan kendali kepada sebuah gelung yang menunggu peristiwa". Sebelum menulis satu listener pun, mari kita bongkar gelung itu, karena begitu Anda sudah melihatnya, tidak ada lagi yang misterius dalam bab ini.

10.6.1 Dari mana peristiwa datang

Ketika Anda menekan tombol tetikus, perangkat keras tetikus mengirim sinyal ke komputer. *Driver* masukan sistem operasi menangkapnya, menghitung di mana penunjuk berada, dan menentukan jendela mana yang ada di bawah penunjuk. Lalu ia menaruh sebuah catatan kecil — "tombol 1 ditekan di (x, y) pada jendela W, pada waktu t" — ke dalam antrean milik program yang memiliki jendela W. Hal yang sama terjadi untuk penekanan tombol papan ketik (fokus papan ketik menentukan jendela mana yang menerimanya), untuk gerakan tetikus, untuk jendela yang tersingkap dan perlu digambar ulang, dan untuk banyak kejadian lain. Setiap catatan semacam itu adalah sebuah **peristiwa**.

Di dalam program, toolkit membaca catatan mentah ini dan mengubahnya menjadi objek peristiwanya sendiri — di Java, `MouseEvent`, `KeyEvent`, `ActionEvent`, dan seterusnya; di Tk, objek peristiwa dengan field seperti `x`, `y`, `keysym`, dan `widget`. Toolkit menaruhnya di **antrean peristiwa** (*event queue*) miliknya sendiri, sebuah antrean masuk-pertama-keluar-pertama persis seperti `ArrayDeque` dalam program di Subbab 10.6.3.

10.6.2 Gelung itu sendiri

Di suatu tempat di dalam setiap toolkit ada gelung yang tampak seperti **Pseudokode 10.1**.

Pseudokode 10.1 Gelung peristiwa yang dijalankan setiap toolkit GUI: tunggu, ambil peristiwa berikutnya, panggil listener yang terdaftar untuknya, dan gambar ulang saat senggang.

```
EVENT-LOOP(queue)
1  while program belum diperintahkan berhenti
2      if queue kosong
3          tunggu sampai sistem operasi mengirim sebuah peristiwa
4      e = DEQUEUE(queue)
5      c = komponen yang menjadi sasaran e
6      for each listener l yang terdaftar pada c untuk jenis e
7          panggil l dengan e           // kode Anda berjalan di sini
8      if tidak ada yang menunggu dan ada daerah bertanda "kotor"
9          gambar ulang daerah kotor itu // paintComponent berjalan di sini
```

Bacalah perlahan. Baris 3 adalah tempat program GUI menghabiskan hampir seluruh hidupnya: tidak melakukan apa-apa, bahkan tidak memakai prosesor, sampai pengguna bertindak. Baris 4–7 adalah **penyaluran** (*dispatch*): cari tahu komponen mana yang dituju peristiwa, lalu panggil

setiap listener yang meminta diberi tahu tentang jenis peristiwa itu pada komponen itu. Baris 7 adalah satu-satunya tempat kode Anda pernah berjalan setelah `main` selesai — dan selama kode itu berjalan, gelung tertahan di baris 7: tidak ada peristiwa lain yang disalurkan dan tidak ada yang digambar ulang. Baris 8 dan 9 adalah cara layar diperbarui: kode tidak pernah menggambar langsung ke layar; ia menandai suatu daerah sebagai perlu digambar ulang, dan gelung menggambarnya ulang ketika tidak ada hal yang lebih mendesak.

Gambar 10.4 menggambarkan gelung yang sama, beserta rute yang ditempuh sebuah peristiwa dari jari pengguna sampai ke kode Anda.



Gambar 10.4 Riwayat sebuah peristiwa. Sistem operasi mengubah tindakan pengguna menjadi peristiwa dan mengantrekannya; gelung peristiwa mengambilnya satu per satu, memanggil listener yang terdaftar untuknya, dan menggambar ulang saat senggang. Listener yang mengubah apa yang harus ditampilkan hanya *meminta* penggambaran ulang.

10.6.3 Gelung peristiwa yang dapat Anda baca

Cara terbaik untuk memercayai pseudokode adalah menjalankannya. **Kode 10.3** dan **Kode 10.4** membangun sistem peristiwa mini yang lengkap tanpa pustaka GUI sama sekali. Ada komponen tiruan (`Widget`), masing-masing dengan daftar listener; sebuah antrean peristiwa; dan sebuah gelung yang nyaris baris demi baris sama dengan **Pseudokode 10.1**. “Pengguna” mengetik berat 4 kg, mengeklik tombol, mengetik 3 kg, mengeklik lagi, lalu menutup jendela; listener tombol menambahkan Rp7,000 per kilogram ke sebuah total dan meminta agar label total digambar ulang.

Kode 10.3 `ToyEventLoop.java`: gelung peristiwa tulisan tangan dengan komponen tiruan, listener, dan sebuah antrean.

```
import java.util.ArrayDeque;
import java.util.ArrayList;
import java.util.LinkedHashMap;
import java.util.List;
import java.util.Map;
import java.util.Queue;

public class ToyEventLoop {
    record Event(String target, String kind, String data) { }

    interface Listener {
        void handle(Event e);
    }

    static class Widget {  // komponen tiruan yang menyimpan daftar listener
        final String name;
        final List<Listener> listeners = new ArrayList<>();

        Widget(String name) {
            this.name = name;
        }

        void addListener(Listener l) {
            listeners.add(l);
        }

        void fire(Event e) {
            for (Listener l : listeners) {
                l.handle(e);
            }
        }
    }

    static final Queue<Event> queue = new ArrayDeque<>();
    static final Map<String, Widget> widgets = new LinkedHashMap<>();
    static int total = 0;
```

Kode 10.3 (Lanjutan) *ToyEventLoop.java: gelung peristiwa tulisan tangan dengan komponen tiruan, listener, dan sebuah antrean.*

```
    public static void main(String[] args) {
        Widget field = new Widget("weightField");
        Widget button = new Widget("addButton");
        Widget label = new Widget("totalLabel");
        for (Widget w : List.of(field, button, label)) {
            widgets.put(w.name, w);
        }
        final String[] typed = {""};  // what the field currently holds
        field.addListener(e -> typed[0] = e.data());
        button.addListener(e -> {
            total += Integer.parseInt(typed[0]) * 7000;
            queue.add(new Event("totalLabel", "repaint", "Rp" + total));
        });
        label.addListener(e -> System.out.println("    screen shows " + e.data()));

        // Yang akan dilakukan pengguna, satu aksi setiap kali:
        List<Event> userActions = List.of(
                new Event("weightField", "text", "4"),
                new Event("addButton", "click", ""),
                new Event("weightField", "text", "3"),
                new Event("addButton", "click", ""),
                new Event("window", "close", ""));
        int next = 0;

        // Gelung peristiwa: tunggu peristiwa, salurkan, ulangi.
        while (true) {
            if (queue.isEmpty()) {  // tidak ada kerja: tunggu pengguna
                queue.add(userActions.get(next++));
            }
            Event e = queue.remove();
            System.out.println("loop: " + e.kind() + " -> " + e.target());
            if (e.kind().equals("close")) {
                break;
            }
            widgets.get(e.target()).fire(e);
        }
        System.out.println("loop ended; total = Rp" + total);
    }
}
```

Kode 10.4 `toy_event_loop.py`: gelung peristiwa yang sama dalam Python, dengan callback berupa fungsi biasa.

```
"""toy_event_loop.py: gelung peristiwa yang ditulis tangan, tanpa pustaka GUI."""
from collections import deque, namedtuple

Event = namedtuple("Event", "target kind data")


class Widget:  # komponen tiruan yang menyimpan daftar callback
    def __init__(self, name):
        self.name = name
        self.listeners = []

    def add_listener(self, callback):
        self.listeners.append(callback)

    def fire(self, event):
        for callback in self.listeners:
            callback(event)


queue = deque()
widgets = {}
total = 0


def main():
    field = Widget("weightField")
    button = Widget("addButton")
    label = Widget("totalLabel")
    for w in (field, button, label):
        widgets[w.name] = w
    typed = [""]  # what the field currently holds

    def on_text(e):
        typed[0] = e.data

    def on_click(e):
        global total
        total += int(typed[0]) * 7000
        queue.append(Event("totalLabel", "repaint", f"Rp{total}"))

    field.add_listener(on_text)
    button.add_listener(on_click)
    label.add_listener(lambda e: print("    screen shows", e.data))
```

Kode 10.4 (Lanjutan) *toy_event_loop.py: gelung peristiwa yang sama dalam Python, dengan callback berupa fungsi biasa.*

```
    # Yang akan dilakukan pengguna, satu aksi setiap kali:
    user_actions = deque([Event("weightField", "text", "4"),
                          Event("addButton", "click", ""),
                          Event("weightField", "text", "3"),
                          Event("addButton", "click", ""),
                          Event("window", "close", "")])

    # Gelung peristiwa: tunggu peristiwa, salurkan, ulangi.
    while True:
        if not queue:  # tidak ada kerja: tunggu pengguna
            queue.append(user_actions.popleft())
        e = queue.popleft()
        print(f"loop: {e.kind} -> {e.target}")
        if e.kind == "close":
            break
        widgets[e.target].fire(e)
    print(f"loop ended; total = Rp{total}")


if __name__ == "__main__":
    main()
```

Kedua program mencetak baris-baris yang persis sama:

```
loop: text -> weightField
loop: click -> addButton
loop: repaint -> totalLabel
    screen shows Rp28000
loop: text -> weightField
loop: click -> addButton
loop: repaint -> totalLabel
    screen shows Rp49000
loop: close -> window
loop ended; total = Rp49000
```

Cocokkan keluaran itu dengan **Pseudokode 10.1**. Antrean mula-mula kosong, jadi gelung "menunggu" — di sini, ia mengambil tindakan pengguna berikutnya dari skrip, yang mewakili baris 3. Peristiwa pertama adalah teks "4" yang tiba di field berat; satu-satunya listener field menyimpannya. Berikutnya datang klik; listener tombol mengalikan $4 \times 7000 = 28{,}000$, menambahkannya ke total dan — inilah bagian pentingnya — *tidak* mencetak apa pun sendiri. Ia *mengantrekan peristiwa gambar ulang* untuk label. Putaran gelung berikutnya menemukan peristiwa gambar ulang itu sudah ada di antrean, jadi gelung menyalurkannya sebelum meminta apa pun lagi dari pengguna, dan listener label "menampilkan" Rp28000 di "layar". Hal yang sama terjadi untuk cucian 3 kg ($3 \times 7000 = 21{,}000$, total 49,000). Akhirnya peristiwa tutup mematahkan gelung, dan barulah saat itu `main` melanjutkan melewati gelung dan mencetak baris terakhir.

Tiga rincian kode layak diperhatikan sekarang, karena muncul lagi di setiap toolkit sungguhan.

- **Pendaftaran.** `button.addListener(e -> { … })` (Java) dan `button.add_listener(on_click)` (Python) tidak menjalankan apa pun. Keduanya hanya menyimpan listener di daftar milik widget, untuk dipanggil kelak. Pemula sering mengira kode di dalam listener berjalan di titik pendaftarannya; itu tidak pernah terjadi.
- **Penyaluran menurut sasaran.** Gelung tidak tahu apa yang dikerjakan listener mana pun. Ia mencari widget menurut nama sasaran peristiwa dan memanggil `fire`, yang memanggil setiap listener terdaftar sesuai urutan pendaftaran. Sebuah widget boleh memiliki berapa pun listener, termasuk tidak ada sama sekali.

- **Listener berbagi keadaan melalui objek.** Listener field dan listener tombol berkomunikasi melalui `typed`, larik satu elemen di Java dan list satu elemen di Python. Mengapa tidak `String typed` biasa? Di Java, lambda hanya boleh memakai variabel lokal yang tidak pernah berubah setelah diisi (Subbab 10.10 menjelaskan alasannya), maka kita biarkan lambda mengubah *isi* sebuah larik, bukan variabelnya. Di Python, fungsi bersarang boleh membaca variabel fungsi pembungkusnya, tetapi mengisinya akan menciptakan variabel lokal baru kecuali dideklarasikan `nonlocal`; lagi-lagi list satu elemen menghindari persoalan itu. Dalam program sungguhan, seperti di sisa bab ini, keadaan bersama cukup tinggal di field objek panel.

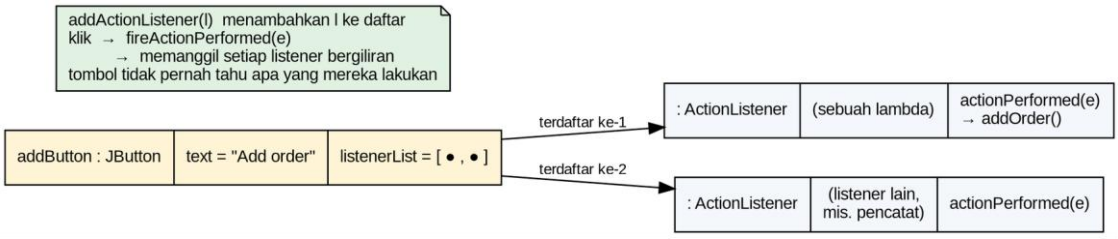
Toolkit sungguhan menambahkan banyak penyempurnaan — peristiwa untuk setiap jenis perangkat keras, penggabungan penggambaran ulang yang berulang menjadi satu, pewaktu, fokus papan ketik, prioritas — tetapi kerangkanya adalah **Kode 10.3**. Di Swing, gelung berjalan di event dispatch thread, yang dinyalakan AWT secara otomatis saat komponen pertama kali ditampilkan atau `invokeLater` dipanggil. Di Tkinter, gelungnya adalah `mainloop()`, yang berjalan di thread yang memanggilnya — biasanya thread utama.

10.7 Mendengarkan Peristiwa

Sekarang kita dapat membaca dan menulis penanganan peristiwa yang sungguhan. Di Java, agar sebuah program mendengarkan jenis peristiwa apa pun, ia harus melakukan tiga hal:

1. **mendeklarasikan** bahwa ia mendengarkan jenis peristiwa itu, dengan mengimplementasikan *antarmuka listener* yang sesuai (untuk klik tombol, `ActionListener`);
2. **mendaftarkan** dirinya pada setiap komponen yang dapat memicu peristiwa itu, dengan memanggil metode `add…Listener` milik komponen tersebut (untuk tombol, `addActionListener`);
3. **mendefinisikan** metode yang diminta antarmuka itu (untuk `ActionListener`, satu-satunya metode `actionPerformed(ActionEvent e)`).

Inilah antarmuka dari Bab 6 yang dipekerjakan. Tombol tidak tahu dan tidak peduli apa kelas listener Anda; ia hanya tahu bahwa listener itu berjanji, dengan mengimplementasikan `ActionListener`, memiliki metode `actionPerformed`. Ketika tombol diklik, gelung peristiwa meminta tombol memberi tahu listener-nya, dan tombol memanggil metode itu pada masing-masing. **Gambar 10.5** memperlihatkan susunannya: tombol menyimpan daftar listener, `addActionListener` menambah ke daftar itu, dan sebuah klik memanggil setiap listener secara bergiliran. Rancangan yang sama, sebuah subjek yang memberi tahu daftar pengamat tanpa tahu apa yang mereka lakukan, adalah pola **Observer** [8], dan pola itu menjadi tulang punggung setiap toolkit GUI.



Gambar 10.5 Sebuah tombol dan listener-nya. Tombol menyimpan daftar; pendaftaran menambah ke daftar itu; sebuah klik memanggil `actionPerformed` setiap listener secara bergiliran. Tombol tidak pernah tahu apa yang dikerjakan listener-nya.

10.7.1 Membaca field teks

Listener sungguhan kita yang pertama membaca apa yang diketik pengguna. Pengguna memasukkan teks di komponen field teks; `JTextField` memicu *peristiwa aksi* (*action event*) ketika pengguna menekan Enter (Return) di dalamnya. **Kode 10.5** adalah panel dengan sebuah field teks dan sebuah label: ketika pengguna menekan Enter, isi label diganti dengan apa pun yang diketik. Empat komentar bernomor menandai empat perubahan dibandingkan panel yang sekadar menampilkan komponen.

Kode 10.5 `EchoPanel.java`: panel yang mendengarkan Enter di field teksnya dan menyalin teksnya ke sebuah label.

```
import java.awt.event.ActionEvent;       // (1) pustaka penanganan peristiwa
import java.awt.event.ActionListener;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JTextField;
import javax.swing.SwingUtilities;

public class EchoPanel extends JPanel
        implements ActionListener {         // (2) deklarasikan: panel ini mendengarkan
    private final JTextField t = new JTextField(15);
    private final JLabel l = new JLabel("Write something on text-field & enter");

    public EchoPanel() {
        add(t);
        t.addActionListener(this);          // (3) daftarkan pada field teks
        add(l);
    }

    @Override
    public void actionPerformed(ActionEvent e) {  // (4) definisikan metodenya
        l.setText(t.getText());
    }

    public static void main(String[] args) throws Exception {
        if (args.length > 0 && args[0].equals("--selftest")) {
            System.setProperty("java.awt.headless", "true");
            SwingUtilities.invokeAndWait(EchoPanel::selfTest);
        } else {
            SwingUtilities.invokeLater(EchoPanel::createAndShow);
        }
    }

    static void createAndShow() {
        JFrame.setDefaultLookAndFeelDecorated(true);
        JFrame frame = new JFrame("JFrame demo");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setContentPane(new EchoPanel());
        frame.pack();
        frame.setVisible(true);
    }
```

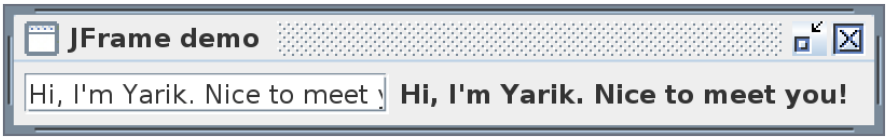
Kode 10.5 (Lanjutan) *EchoPanel.java: panel yang mendengarkan Enter di field teksnya dan menyalin teksnya ke sebuah label.*

```
    static void selfTest() {
        EchoPanel panel = new EchoPanel();
        System.out.println("before: " + panel.l.getText());
        panel.t.setText("Hi, I'm Yarik. Nice to meet you!");
        System.out.println("typed:  " + panel.t.getText());
        panel.t.postActionEvent();          // persis yang dilakukan penekanan Enter
        System.out.println("after:  " + panel.l.getText());
        System.out.println("listeners on the field: "
                + panel.t.getActionListeners().length);
    }
}
```

- (1) `import java.awt.event.ActionEvent` dan `ActionListener`: kelas-kelas penanganan peristiwa tinggal di `java.awt.event`, bukan di `javax.swing`. Banyak program menulis `import java.awt.event.*;` sebagai gantinya.
- (2) `implements ActionListener` mendeklarasikan bahwa `EchoPanel` *adalah sebuah* `ActionListener`: ia dapat menangani peristiwa aksi.
- (3) `t.addActionListener(this)` mendaftarkan panel *ini* pada field teks `t`: "bila terjadi aksi di `t`, panggillah `actionPerformed` milikku". Kata kunci `this` berarti objek yang sedang dibangun — panel itu sendiri.
- (4) `actionPerformed(ActionEvent e)` adalah metode yang diminta antarmuka. Ia memanggil `t.getText()` untuk membaca teks yang saat ini ada di field dan `l.setText(...)` untuk mengubah label.

Konstruktor `JTextField(15)` membuat field yang cukup lebar untuk sekitar 15 karakter; lebar itu hanya memengaruhi ukuran yang disukai, bukan berapa banyak yang boleh diketik. Panel memakai tata letak bawaan `JPanel`, yaitu `FlowLayout`, yang menaruh field dan label berdampingan.

Gambar 10.6 memperlihatkan jendela setelah pengguna mengetik sapaan dan menekan Enter: label, yang tadinya berbunyi *Write something on text-field & enter*, kini mengulang sapaan itu.



Gambar 10.6 Jendela gema setelah pengguna mengetik sapaan di field teks dan menekan Enter: listener telah menyalin teksnya ke label.

Uji mandiri membuktikan hal yang sama tanpa layar:

```
before: Write something on text-field & enter
typed:  Hi, I'm Yarik. Nice to meet you!
after:  Hi, I'm Yarik. Nice to meet you!
listeners on the field: 1
```

Baris kuncinya adalah `panel.t.postActionEvent()`. `JTextField.postActionEvent()` persis yang dilakukan field secara internal ketika Enter ditekan: ia membangun sebuah `ActionEvent` dan mengantarkannya ke setiap listener terdaftar. Baris terakhir memastikan bahwa ada satu listener

terdaftar — panel kita. Perhatikan bahwa pengujian tidak memanggil satu pun metode `EchoPanel` secara langsung; ia mengendalikan komponen, dan komponenlah yang memanggil panel, persis seperti yang terjadi untuk pengguna sungguhan.

Kode 10.6 adalah versi Tkinter-nya.

Kode 10.6 `echo_panel.py`: panel gema yang sama dalam Tkinter, memakai `bind` untuk memasang callback pada tombol Enter.

```
"""echo_panel.py: salin teks sebuah entry ke label ketika Enter ditekan."""
import sys
import tkinter as tk


class EchoPanel(tk.Frame):
    def __init__(self, master):
        super().__init__(master)
        self.t = tk.Entry(self, width=15)
        self.t.pack(side=tk.LEFT, padx=4, pady=4)
        self.t.bind("<Return>", self.on_enter)  # register the callback
        self.l = tk.Label(self, text="Write something on text-field & enter")
        self.l.pack(side=tk.LEFT, padx=4)

    def on_enter(self, event):  # callback-nya: Tk mengirim sebuah objek peristiwa
        self.l.config(text=self.t.get())


def self_test():
    root = tk.Tk()
    root.withdraw()
    panel = EchoPanel(root)
    panel.pack()
    print("before:", panel.l.cget("text"))
    panel.t.insert(0, "Hi, I'm Yarik. Nice to meet you!")
    print("typed: ", panel.t.get())
    panel.on_enter(None)  # yang dipanggil Tk saat Enter ditekan di layar
    print("after: ", panel.l.cget("text"))
    print("bindings on the entry:", panel.t.bind())
    root.destroy()


def main():
    if "--selftest" in sys.argv:
        self_test()
        return
    root = tk.Tk()
    root.title("JFrame demo")
    EchoPanel(root).pack()
    root.mainloop()


if __name__ == "__main__":
    main()
```

```
before: Write something on text-field & enter
typed:  Hi, I'm Yarik. Nice to meet you!
after:  Hi, I'm Yarik. Nice to meet you!
bindings on the entry: ('<Key-Return>',)
```

Tkinter tidak memakai antarmuka listener. Sebagai gantinya, `widget.bind(sequence, callback)` memasang apa pun yang dapat dipanggil pada sebuah **urutan peristiwa** (*event sequence*), yang ditulis sebagai string di dalam kurung sudut: `"<Return>"` adalah tombol Enter, `"<Button-1>"` penekanan tombol kiri tetikus, `"<B1-Motion>"` menggerakkan tetikus sambil menahan tombol kiri, `"<KeyPress-a>"` tombol A. Ketika peristiwa itu terjadi di atas widget tersebut (atau, untuk

papan ketik, ketika fokus ada padanya), Tk memanggil callback dengan satu argumen, sebuah **objek peristiwa** yang atributnya mendeskripsikan apa yang terjadi: `event.widget`, `event.x` dan `event.y`, `event.keysym`, dan seterusnya. `self.on_enter` adalah *bound method* — metode beserta objek pemiliknya — sehingga ketika Tk memanggilnya dengan peristiwa, Python menyertakan `self` secara otomatis. Baris terakhir pengujian memperlihatkan bagaimana Tk melaporkan ikatan itu: `'<Key-Return>'` adalah nama lengkap `"<Return>"`.

Uji mandiri Tkinter berbeda dalam satu hal yang mencerahkan: ia memanggil `panel.on_enter(None)` sendiri. Tk mengantarkan peristiwa papan ketik hanya ke jendela yang tampil di layar dan memegang fokus papan ketik; jendela yang disembunyikan tidak menerima satu pun, dan `event_generate("<Return>")` padanya diam-diam tidak berbuat apa-apa (cobalah). Jadi pengujian melakukan apa yang akan dilakukan Tk untuk penekanan tombol sungguhan — memanggil callback yang terikat — dan memberikan `None` sebagai objek peristiwa, yang memang tidak dipakai `on_enter`. Untuk tombol dan tombol centang, Tk menyediakan `invoke()`, yang bekerja tanpa layar, dan mulai sekarang kita akan memakainya.

10.7.2 Objek peristiwa, dan komponen mana yang memicunya

Setiap metode listener menerima sebuah objek peristiwa, dan informasi di dalamnya bergantung pada jenis peristiwa. Anda selalu dapat menanyakan keadaan sebuah *komponen* (`t.getText()`, `cb.isSelected()`); *peristiwa* memberi tahu Anda apa yang baru saja terjadi:

- untuk peristiwa aksi dan peristiwa item, `e.getSource()` mengembalikan komponen yang memicu peristiwa — tombol mana, field teks mana, kotak centang mana;
- untuk peristiwa tetikus, `e.getPoint()` mengembalikan posisi penunjuk sebagai sebuah `Point` yang field `.x` dan `.y`-nya adalah koordinatnya, dan `e.getX()`, `e.getY()` mengembalikannya secara langsung;
- untuk peristiwa papan ketik, `e.getKeyCode()` dan `e.getKeyChar()` menyebutkan tombol yang mana.

`getSource()` penting setiap kali satu listener didaftarkan pada beberapa komponen, karena hanya dengan itulah listener dapat membedakan mereka. Subbab 10.9 memakainya untuk membedakan tombol *Darken* dari tombol *Lighten*.

10.7.3 Peristiwa item: kotak centang

Kotak centang dapat dicentang dan dilepas; setiap perubahan keadaan adalah **peristiwa item** (*item event*). Tiga langkahnya sama, dengan nama yang berbeda: deklarasikan `implements ItemListener`, daftarkan dengan `cb.addItemListener(this)`, dan definisikan `itemStateChanged(ItemEvent e)`. **Kode 10.7** memiliki dua kotak centang dan satu listener untuk keduanya.

Kode 10.7 `FruitChoice.java`: dua kotak centang dan satu listener item yang melaporkan kotak mana yang berubah dan bagaimana.

```
import java.awt.event.ItemEvent;
import java.awt.event.ItemListener;
import javax.swing.JCheckBox;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.SwingUtilities;

public class FruitChoice extends JPanel implements ItemListener {
    private final JLabel label = new JLabel("Which one do you like?");
    private final JCheckBox cb1 = new JCheckBox("Mango");
    private final JCheckBox cb2 = new JCheckBox("Apple");
    private final JLabel explanation = new JLabel("You haven't checked anything");

    public FruitChoice() {
        add(label);
        add(cb1);
        add(cb2);
        add(explanation);
        cb1.addItemListener(this);
        cb2.addItemListener(this);
    }

    @Override
    public void itemStateChanged(ItemEvent e) {
        JCheckBox cb3 = (JCheckBox) e.getItemSelectable();  // kotak mana yang berubah?
        if (cb3.isSelected()) {
            explanation.setText(cb3.getText() + " is checked");
        } else {
            explanation.setText(cb3.getText() + " is unchecked");
        }
    }

    public static void main(String[] args) throws Exception {
        if (args.length > 0 && args[0].equals("--selftest")) {
            System.setProperty("java.awt.headless", "true");
            SwingUtilities.invokeAndWait(FruitChoice::selfTest);
        } else {
            SwingUtilities.invokeLater(() -> {
                JFrame.setDefaultLookAndFeelDecorated(true);
                JFrame frame = new JFrame("Item event");
                frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                frame.add(new FruitChoice());
                frame.pack();
                frame.setVisible(true);
            });
```

Kode 10.7 (Lanjutan) *FruitChoice.java: dua kotak centang dan satu listener item yang melaporkan kotak mana yang berubah dan bagaimana.*

```
        }
    }

    static void selfTest() {
        FruitChoice panel = new FruitChoice();
        System.out.println("start:        " + panel.explanation.getText());
        panel.cb1.doClick();  // jalur yang sama dengan klik tetikus sungguhan
        System.out.println("click Mango:  " + panel.explanation.getText());
        panel.cb1.doClick();
        System.out.println("click Mango:  " + panel.explanation.getText());
        panel.cb2.doClick();
        System.out.println("click Apple:  " + panel.explanation.getText());
        panel.cb2.doClick();
        System.out.println("click Apple:  " + panel.explanation.getText());
    }
}
```

`e.getItemSelectable()` mengembalikan komponen yang keadaannya berubah (untuk kotak centang, objek yang sama dengan `getSource()`, tetapi bertipe `ItemSelectable`); *cast* `(JCheckBox)` memberi kita akses ke `isSelected()` dan `getText()`. **Gambar 10.7** memperlihatkan jendela setelah pengguna mencentang *Mango*.



Gambar 10.7 Jendela kotak centang setelah *Mango* dicentang: listener item telah menulis ulang label penjelasan.

Uji mandiri mengeklik setiap kotak dua kali dengan `doClick()`, yang melakukan klik secara terprogram melalui jalur yang persis sama dengan klik tetikus, termasuk memicu listener-nya:

```
start:        You haven't checked anything
click Mango:  Mango is checked
click Mango:  Mango is unchecked
click Apple:  Apple is checked
click Apple:  Apple is unchecked
```

Kode 10.8 adalah versi Tkinter-nya.

Kode 10.8 `fruit_choice.py`: pilihan yang sama dalam Tkinter, dengan satu `BooleanVar` per tombol centang dan sebuah callback `command`.

```
"""fruit_choice.py: dua tombol centang yang melaporkan mana yang berubah."""
import sys
import tkinter as tk


class FruitChoice(tk.Frame):
    def __init__(self, master):
        super().__init__(master)
        tk.Label(self, text="Which one do you like?").pack(side=tk.LEFT)
        self.mango = tk.BooleanVar(value=False)
        self.apple = tk.BooleanVar(value=False)
        self.cb1 = tk.Checkbutton(self, text="Mango", variable=self.mango,
                                  command=lambda: self.changed("Mango", self.mango))
        self.cb2 = tk.Checkbutton(self, text="Apple", variable=self.apple,
                                  command=lambda: self.changed("Apple", self.apple))
        self.cb1.pack(side=tk.LEFT)
        self.cb2.pack(side=tk.LEFT)
        self.explanation = tk.Label(self, text="You haven't checked anything")
        self.explanation.pack(side=tk.LEFT)

    def changed(self, name, var):  # lambda di atas memberi tahu kotak mana yang berubah
        if var.get():
            self.explanation.config(text=name + " is checked")
        else:
            self.explanation.config(text=name + " is unchecked")


def self_test():
    root = tk.Tk()
    root.withdraw()
    panel = FruitChoice(root)
    panel.pack()
    print("start:       ", panel.explanation.cget("text"))
    for box, name in [(panel.cb1, "Mango"), (panel.cb1, "Mango"),
                      (panel.cb2, "Apple"), (panel.cb2, "Apple")]:
        box.invoke()  # jalur yang sama dengan klik tetikus sungguhan
        print(f"click {name}: ", panel.explanation.cget("text"))
    root.destroy()
```

Kode 10.8 (Lanjutan) *fruit_choice.py: pilihan yang sama dalam Tkinter, dengan satu BooleanVar per tombol centang dan sebuah callback command.*

```
def main():
    if "--selftest" in sys.argv:
        self_test()
        return
    root = tk.Tk()
    root.title("Item event")
    FruitChoice(root).pack(padx=6, pady=6)
    root.mainloop()


if __name__ == "__main__":
    main()
```

```
start:       You haven't checked anything
click Mango:  Mango is checked
click Mango:  Mango is unchecked
click Apple:  Apple is checked
click Apple:  Apple is unchecked
```

Versi Tkinter memperlihatkan dua gagasan yang akan kita pakai lagi. Pertama, `Checkbutton` Tkinter menyimpan keadaannya di sebuah **variabel kendali** (*control variable*), di sini

`BooleanVar`: mencentang kotak menyetel variabel menjadi `True`, dan `var.get()` membacanya. Variabel kendali (`StringVar`, `IntVar`, `DoubleVar`, `BooleanVar`) dapat dipakai bersama oleh beberapa widget dan dapat *diawasi* perubahannya, yang akan dimanfaatkan jendela laundry untuk harganya yang langsung diperbarui. Kedua, alih-alih satu listener yang bertanya "kotak yang mana?", setiap tombol centang mendapat `command`-nya sendiri, sebuah `lambda` kecil yang meneruskan nama dan variabel kotak itu ke metode bersama. Lambda itulah jawaban atas "kotak yang mana?": masing-masing mengingat argumennya sendiri. Subbab 10.10 menjelaskan bagaimana persisnya.

10.7.4 Empat jenis peristiwa dalam bab ini

Tabel 10.2 merangkum, untuk empat jenis peristiwa utama, apa yang harus dideklarasikan program Java, bagaimana ia mendaftar, metode mana yang harus didefinisikannya — dan padanannya di Tkinter. Peristiwa tetikus dan gerak tetikus tidak memerlukan komponen untuk didaftari bila panel mendengarkan dirinya sendiri: panel cukup memanggil `addMouseListener(this)` pada dirinya, karena panel itulah komponen yang ada di bawah tetikus.

Tabel 10.2 Empat jenis peristiwa utama: cara mendengarkannya di Swing dan di Tkinter.

Peristiwa	Java: deklarasi	Java: pendaftaran	Java: metode yang didefinisikan	Tkinter
Aksi (tombol diklik, Enter di field)	`implements ActionListener`	`c.addActionListener(l)`	`actionPerformed(ActionEvent)`	`command=f`, atau `bind("<Return>", f)`
Item (kotak centang dicentang atau dilepas)	`implements ItemListener`	`c.addItemListener(l)`	`itemStateChanged(ItemEvent)`	`command=f` pada `Checkbutton` dengan variabel
Tetikus (tekan, lepas, klik, masuk, keluar)	`implements MouseListener`	`addMouseListener(l)`	`mousePressed`, `mouseReleased`, `mouseClicked`, `mouseEntered`, `mouseExited` (semuanya `MouseEvent`)	`bind("<ButtonPress-1>", f)`, `"<ButtonRelease-1>"`, `"<Enter>"`, `"<Leave>"`
Gerak tetikus (gerak, seret)	`implements MouseMotionListener`	`addMouseMotionListener(l)`	`mouseMoved`, `mouseDragged` (`MouseEvent`)	`bind("<Motion>", f)`, `bind("<B1-Motion>", f)`

Sebuah panel dapat mendengarkan beberapa jenis sekaligus: ia mencantumkan beberapa antarmuka setelah `implements`, mendaftar untuk masing-masing, dan mendefinisikan semua metode yang diminta. `ShadeBox` di Subbab 10.9 melakukan persis itu untuk peristiwa aksi dan peristiwa tetikus.

10.8 Layout Manager versus pack dan grid

Di mana tombol *Add order* harus diletakkan? Satu jawaban adalah "di piksel (192, 82), selebar 104 piksel". Jawaban itu keliru di komputer berikutnya, yang fonnya lebih besar sehingga teksnya tidak lagi muat, dan keliru lagi ketika pengguna melebarkan jendela, atau ketika program diterjemahkan ke bahasa yang kata-katanya lebih panjang. Menata panel GUI adalah urusan yang

pelik: tata letaknya harus memperhitungkan perubahan ukuran jendela, dan harus luwes terhadap ukuran label dan tombol, yang bergantung pada fon, bahasa, dan resolusi layar.

Karena itu kedua toolkit memungkinkan Anda mendeskripsikan tata letak dengan *aturan*, bukan koordinat. Di Java, setiap kontainer memiliki sebuah **layout manager** (pengelola tata letak), objek yang mengimplementasikan antarmuka `LayoutManager` dan menentukan, setiap kali kontainer perlu ditata, di mana setiap anak diletakkan dan seberapa besar ukurannya. Ia memperhitungkan ukuran *yang disukai*, *minimum*, dan *maksimum* setiap anak (yang dihitung komponen dari fon dan isinya) serta ukuran kontainer. Di Tkinter, tugas yang sama dikerjakan **geometry manager** (pengelola geometri), yang jumlahnya tiga: `pack`, `grid`, dan `place`. Pustaka Java mendefinisikan sejumlah layout manager; Anda juga dapat menulis sendiri atau memakai buatan pihak ketiga. Kita melihat empat yang mencakup hampir semua kebutuhan — panduan visual tutorial resmi memperlihatkan semuanya [9] — serta `pack` dan `grid`.

10.8.1 FlowLayout: seperti kata dalam paragraf

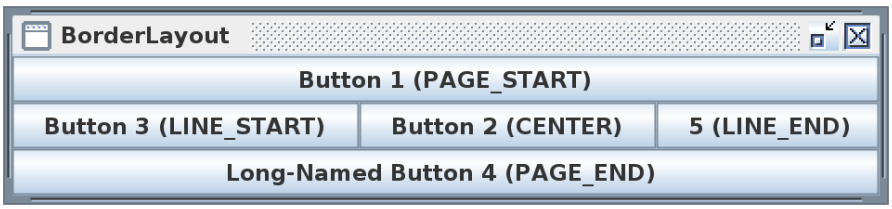
`FlowLayout` adalah layout manager bawaan `JPanel`. Ia menaruh komponen dari kiri ke kanan dalam satu baris, seperti kata-kata di pengolah kata, dan ketika sebuah baris penuh ia memulai baris baru di bawahnya. Komponen dipertahankan pada ukuran yang disukai sejauh mungkin — tidak direntangkan atau disusutkan — dan setiap baris secara bawaan diletakkan di tengah secara horizontal. **Gambar 10.8** memperlihatkan lima tombol dalam satu baris.



Gambar 10.8 Lima tombol dalam `FlowLayout`: masing-masing mempertahankan ukuran yang disukainya, dan barisnya mengalir dari kiri ke kanan.

10.8.2 BorderLayout: lima wilayah

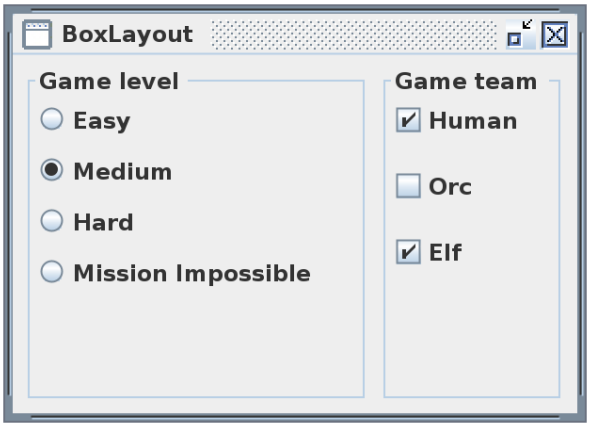
`BorderLayout` adalah bawaan untuk content pane sebuah frame. Ia membagi kontainer menjadi tepat lima wilayah — `PAGE_START` (atas), `PAGE_END` (bawah), `LINE_START` (kiri, untuk bahasa yang ditulis dari kiri ke kanan), `LINE_END` (kanan), dan `CENTER` — dan setiap wilayah menampung paling banyak satu komponen. Anda memilih wilayahnya saat menambahkan: `container.add(component, BorderLayout.CENTER)`. Untuk kontainer lain, setel tata letaknya lebih dulu dengan `container.setLayout(new BorderLayout())` atau berikan kepada konstruktornya. Kelima komponen direntangkan untuk mengisi seluruh ruang yang tersedia: yang atas dan bawah mengambil lebar penuh pada tinggi yang disukainya, yang kiri dan kanan mengambil sisa tinggi pada lebar yang disukainya, dan yang tengah mendapat semua sisanya. **Gambar 10.9** memperlihatkan kelima wilayah itu. Tentu saja, komponen mana pun dari kelimanya dapat berupa panel yang mengelompokkan komponen-komponen yang lebih kecil — dan persis begitulah jendela laundry dibangun.



Gambar 10.9 Lima wilayah `BorderLayout`. Wilayah tengah mengambil seluruh ruang yang disisakan keempat tepi.

10.8.3 BoxLayout: satu baris atau satu kolom

Box (kotak) adalah larik komponen yang disusun mendatar atau tegak. `BoxLayout` menyusun anak-anak kontainer dalam satu kolom (`BoxLayout.Y_AXIS`) atau satu baris (`X_AXIS`): `panel.setLayout(new BoxLayout(panel, BoxLayout.Y_AXIS))`, atau, melalui kontainer `Box`, `Box b = Box.createVerticalBox()`. Komponen dapat diratakan ke kiri, ke kanan, atau ke tengah, dan pengisi tak terlihat memberi jarak di antaranya: `Box.createVerticalStrut(20)` adalah celah tetap 20 piksel, `Box.createGlue()` pegas yang menyerap ruang lebih. **Gambar 10.10** memperlihatkan dua kotak tegak, masing-masing di panel berbingkai berjudul, berdampingan di dalam sebuah kotak mendatar.



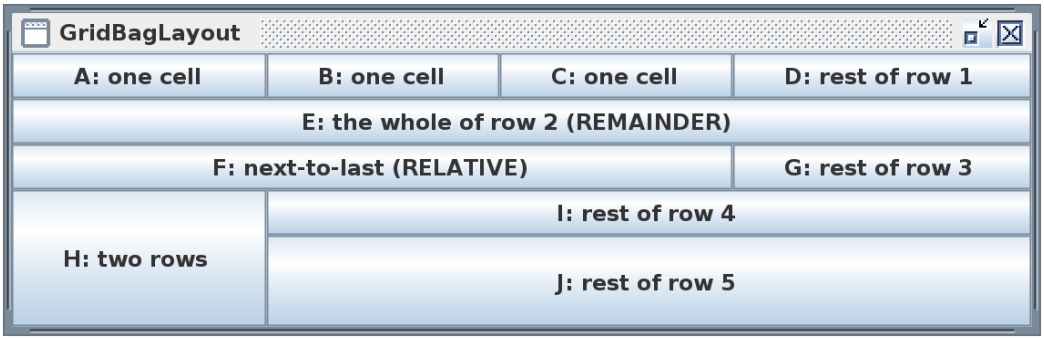
Gambar 10.10 Dua kotak tegak di dalam sebuah kotak mendatar. Tombol radio di kiri tergabung dalam satu `ButtonGroup`, sehingga hanya satu yang dapat dipilih; kotak centang di kanan saling bebas.

10.8.4 GridBagLayout: yang paling luwes

`GridBagLayout` adalah layout manager paling luwes di Swing, sekaligus yang paling bertele-tele. Bayangkan panel dibagi menjadi kisi sel tak terlihat, yang baris dan kolomnya boleh berbeda ukuran. Setiap komponen menempati sebuah blok sel berbentuk persegi panjang, dan di dalam bloknya ia boleh merentang atau tidak, rata ke suatu sisi atau pojok, dan diberi bantalan. Semua pilihan ini diberikan dalam objek terpisah dari kelas `GridBagConstraints`, yang Anda isi sebelum setiap `add`:

```
panel.setLayout(new GridBagLayout());
GridBagConstraints c = new GridBagConstraints();
c.gridx = 0; c.gridy = 0;           // posisi: kolom 0, baris 0
c.gridwidth = 1; c.gridheight = 3;  // ukuran: selebar satu kolom, setinggi tiga baris
c.weightx = 100; c.weighty = 100;   // rentang: bagian dari ruang lebih
JList style = new JList();
panel.add(style, c);
```

Bobot menentukan siapa yang menerima ruang lebih: sisa lebar sebuah baris dibagikan kepada komponen-komponennya sebanding dengan `weightx`, dan bobot 0 berarti "jangan pernah membesar". Biasanya kolom-kolom samping mempertahankan ukurannya (bobot 0.0) dan satu kolom mengambil seluruh sisanya (bobot 1.0). `fill` menyatakan apakah komponen membesar mengisi selnya (`NONE`, `HORIZONTAL`, `VERTICAL`, `BOTH`), `anchor` di mana ia berada bila tidak, `insets` seberapa lebar margin di sekelilingnya, dan `gridwidth = GridBagConstraints.REMAINDER` menjadikan komponen yang terakhir di barisnya. **Gambar 10.11** memperlihatkan komponen yang merentang pada jumlah sel yang berbeda-beda.



Gambar 10.11 Sebuah `GridBagLayout` tempat tombol-tombol merentang satu sel, sisa satu baris, atau dua baris, sesuai `gridwidth`, `gridheight`, dan bobotnya.

Karena objek batasan itu dipakai ulang dari satu `add` ke `add` berikutnya, field-fieldnya mempertahankan nilainya: lupa mengembalikan `gridwidth` atau sebuah bobot adalah *bug* klasik `GridBagLayout`. Karena itu merupakan praktik yang baik untuk tidak memakai `GridBagLayout` langsung di seluruh program, melainkan menyembunyikannya di dalam pembantu kecil — sebuah metode, atau subkelas `JPanel` dengan metode seperti `add(Component c, int x, int y, int w, int h)` yang mengisi batasan itu untuk Anda. Formulir laundry di Subbab 10.12 melakukan hal yang sama dengan sebuah perulangan atas baris-barisnya.

10.8.5 Tkinter: pack, grid, dan place

Geometry manager Tkinter bukanlah objek milik kontainer; mereka adalah metode yang dipanggil pada setiap *anak*, yang sekaligus memberinya aturan penempatan.

- `widget.pack(side=..., fill=..., expand=..., padx=..., pady=...)` menumpuk widget ke sisi-sisi induknya, dalam urutan pemanggilan `pack`. `side=tk.LEFT` menaruh setiap widget di sebelah kanan widget sebelumnya, seperti sebuah baris; `side=tk.TOP` (bawaan) menaruh masing-masing di bawah yang sebelumnya, seperti sebuah kolom. `fill=tk.X` merentangkan widget secara mendatar sepanjang ruang yang diberikan kepadanya, dan `expand=True` memberinya bagian dari ruang lebih.
- `widget.grid(row=..., column=..., sticky=..., columnspan=..., padx=...)` menaruh widget ke sel-sel sebuah tabel, seperti `GridBagLayout` tanpa objek batasan terpisah. `sticky="ew"` merentangkan widget ke tepi timur dan barat selnya (`"w"` saja meratakannya ke kiri), dan `parent.columnconfigure(1, weight=1)` memberi kolom 1 seluruh sisa lebar, persis seperti `weightx`.
- `widget.place(x=..., y=...)` memakai koordinat mutlak dan jarang merupakan gagasan yang baik, karena alasan-alasan di awal subbab ini.

Jangan pernah mencampur `pack` dan `grid` untuk anak-anak dari induk yang *sama*. Kedua pengelola itu akan berebut ukuran induk; Tk versi lama sekadar membeku selama mereka bernegosiasi, dan Tk 8.6 menolak mentah-mentah:

```
_tkinter.TclError: cannot use geometry manager grid inside . which already has slaves
managed by pack
```

(Tanda `.` adalah nama Tk untuk jendela akar.) Memakai `pack` di satu frame dan `grid` di frame lain tidak masalah; jendela laundry melakukan persis itu.

10.8.6 Mengamati para pengelola bekerja

Kode 10.9 membangun keempat jenis panel. Bila dijalankan normal, ia menampilkannya di empat tab sebuah `JTabbedPane`. Bila dijalankan dengan `--selftest`, ia memberi setiap panel sebuah ukuran, meminta layout manager menatanya *sekarang* dengan `doLayout()`, dan mencetak di mana setiap anak akhirnya berada — sehingga kita dapat melihat aturannya bekerja tanpa layar.

Kode 10.9 `LayoutTour.java`: empat layout manager utama; uji mandirinya mencetak batas-batas yang ditetapkan setiap pengelola.

```
import java.awt.BorderLayout;
import java.awt.Component;
import java.awt.FlowLayout;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.Insets;
import java.awt.Rectangle;
import javax.swing.AbstractButton;
import javax.swing.BoxLayout;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JTabbedPane;
import javax.swing.JTextField;
import javax.swing.SwingUtilities;

public class LayoutTour {
    static final String[] NAMES = {"Button 1", "Button 2", "Button 3",
                                   "Long-Named Button 4", "5"};

    static JPanel flow() {
        JPanel p = new JPanel(new FlowLayout());  // bawaan untuk JPanel
        for (String n : NAMES) {
            p.add(new JButton(n));
        }
        return p;
    }

    static JPanel border() {
        JPanel p = new JPanel(new BorderLayout());  // bawaan untuk content pane
        p.add(new JButton("Button 1 (PAGE_START)"), BorderLayout.PAGE_START);
        p.add(new JButton("Button 2 (CENTER)"), BorderLayout.CENTER);
        p.add(new JButton("Button 3 (LINE_START)"), BorderLayout.LINE_START);
        p.add(new JButton("Long-Named Button 4 (PAGE_END)"), BorderLayout.PAGE_END);
        p.add(new JButton("5 (LINE_END)"), BorderLayout.LINE_END);
        return p;
    }
```

Kode 10.9 (Lanjutan) *LayoutTour.java: empat layout manager utama; uji mandirinya mencetak batas-batas yang ditetapkan setiap pengelola.*

```
    static JPanel box() {
        JPanel p = new JPanel();
        p.setLayout(new BoxLayout(p, BoxLayout.Y_AXIS));  // satu kolom
        for (String n : NAMES) {
            JButton b = new JButton(n);
            b.setAlignmentX(Component.CENTER_ALIGNMENT);
            p.add(b);
        }
        return p;
    }

    static JPanel gridBag() {
        JPanel p = new JPanel(new GridBagLayout());
        GridBagConstraints c = new GridBagConstraints();
        c.insets = new Insets(2, 4, 2, 4);
        String[] labels = {"Customer", "Phone", "Weight (kg)"};
        for (int row = 0; row < labels.length; row++) {
            c.gridx = 0; c.gridy = row;              // posisi: kolom 0
            c.weightx = 0; c.fill = GridBagConstraints.NONE;
            c.anchor = GridBagConstraints.LINE_END;
            p.add(new JLabel(labels[row]), c);
            c.gridx = 1;                             // kolom 1 merentang
            c.weightx = 1; c.fill = GridBagConstraints.HORIZONTAL;
            p.add(new JTextField(10), c);
        }
        c.gridx = 1; c.gridy = labels.length;
        c.weightx = 0; c.fill = GridBagConstraints.NONE;
        c.anchor = GridBagConstraints.LINE_END;
        p.add(new JButton("Add order"), c);
        return p;
    }

    static String name(Component c) {
        if (c instanceof AbstractButton b) {
            return b.getText();
        }
        if (c instanceof JLabel l) {
            return l.getText();
        }
        return "[text field]";
    }
```

Kode 10.9 (Lanjutan) *LayoutTour.java: empat layout manager utama; uji mandirinya mencetak batas-batas yang ditetapkan setiap pengelola.*

```
    static void report(String title, JPanel p, int w, int h) {
        p.setSize(w, h);
        p.doLayout();  // minta layout manager menempatkan anak-anaknya sekarang
        System.out.println(title + " in " + w + " x " + h + ":");
        for (Component c : p.getComponents()) {
            Rectangle r = c.getBounds();
            System.out.printf("  %-31s x=%3d y=%3d w=%3d h=%3d%n",
                    name(c), r.x, r.y, r.width, r.height);
        }
    }

    public static void main(String[] args) throws Exception {
        if (args.length > 0 && args[0].equals("--selftest")) {
            System.setProperty("java.awt.headless", "true");
            SwingUtilities.invokeAndWait(() -> {
                report("FlowLayout", flow(), 560, 40);
                report("FlowLayout", flow(), 230, 100);
                report("BorderLayout", border(), 460, 150);
                report("BoxLayout", box(), 230, 150);
                report("GridBagLayout", gridBag(), 300, 120);
            });
        } else {
            SwingUtilities.invokeLater(() -> {
                JFrame.setDefaultLookAndFeelDecorated(true);
                JFrame frame = new JFrame("Layout tour");
                frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                JTabbedPane tabs = new JTabbedPane();
                tabs.addTab("Flow", flow());
                tabs.addTab("Border", border());
                tabs.addTab("Box", box());
                tabs.addTab("GridBag", gridBag());
                frame.add(tabs);
                frame.pack();
                frame.setVisible(true);
            });
        }
    }
}
```

```
FlowLayout in 560 x 40:
  Button 1                        x= 14 y=  5 w= 94 h= 25
  Button 2                        x=113 y=  5 w= 94 h= 25
  Button 3                        x=212 y=  5 w= 94 h= 25
  Long-Named Button 4             x=311 y=  5 w=187 h= 25
  5                               x=503 y=  5 w= 42 h= 25
FlowLayout in 230 x 100:
  Button 1                        x= 18 y=  5 w= 94 h= 25
  Button 2                        x=117 y=  5 w= 94 h= 25
  Button 3                        x= 68 y= 35 w= 94 h= 25
  Long-Named Button 4             x= 21 y= 65 w=187 h= 25
  5                               x= 94 y= 95 w= 42 h= 25
BorderLayout in 460 x 150:
  Button 1 (PAGE_START)           x=  0 y=  0 w=460 h= 25
  Button 2 (CENTER)               x=187 y= 25 w=153 h=100
  Button 3 (LINE_START)           x=  0 y= 25 w=187 h=100
  Long-Named Button 4 (PAGE_END)  x=  0 y=125 w=460 h= 25
  5 (LINE_END)                    x=340 y= 25 w=120 h=100
BoxLayout in 230 x 150:
  Button 1                        x= 67 y=  0 w= 94 h= 25
  Button 2                        x= 67 y= 25 w= 94 h= 25
  Button 3                        x= 67 y= 50 w= 94 h= 25
  Long-Named Button 4             x= 21 y= 75 w=187 h= 25
  5                               x= 93 y=100 w= 42 h= 25
GridBagLayout in 300 x 120:
  Customer                        x= 18 y= 15 w= 68 h= 15
  [text field]                    x= 94 y= 13 w=202 h= 19
  Phone                           x= 41 y= 38 w= 45 h= 15
  [text field]                    x= 94 y= 36 w=202 h= 19
  Weight (kg)                     x=  4 y= 61 w= 82 h= 15
  [text field]                    x= 94 y= 59 w=202 h= 19
  Add order                       x=192 y= 82 w=104 h= 25
```

Nilai piksel sedikit bergantung pada fon yang terpasang; polanya tidak. Bacalah keluarannya dengan aturan-aturan tadi dalam pikiran.

- **FlowLayout, lebar 560 piksel.** Kelima tombol muat dalam satu baris pada ukuran yang disukainya, setinggi 25 piksel, dengan celah 5 piksel. Barisnya total selebar 531 piksel ($503 + 42 - 14$), jadi pengelola meletakkannya di tengah dengan mulai dari $x = 14$.
- **FlowLayout, lebar 230 piksel.** Tombol yang sama tidak lagi muat, jadi pengelola membungkusnya menjadi empat baris, masing-masing di tengah. Tidak ada tombol yang disusutkan.
- **BorderLayout.** Tombol atas dan bawah merentang selebar penuh 460 piksel pada tinggi yang disukainya, 25. Tombol kiri dan kanan mempertahankan lebar yang disukainya (187 dan 120) dan merentang setinggi seluruh sisa tinggi, 100. Tombol tengah menerima semua sisanya: lebar $460 - 187 - 120 = 153$ piksel.
- **BoxLayout.** Satu kolom; setiap tombol mempertahankan ukuran yang disukainya dan berada di tengah karena kita menyetel `setAlignmentX(CENTER_ALIGNMENT)`.
- **GridBagLayout.** Kolom 0 selebar labelnya yang paling lebar, *Weight (kg)*, dan setiap label ditambatkan ke ujung kanan selnya (`LINE_END`); itulah sebabnya *Customer* mulai di $x = 18$ dan *Phone* di $x = 41$: keduanya rata kanan. Kolom 1 memiliki `weightx = 1` dan `fill = HORIZONTAL`, sehingga field teks mengambil seluruh sisa lebar (202 piksel) walaupun kita hanya meminta 10 kolom. Tombol *Add order*, dengan `fill = NONE`, mempertahankan ukurannya dan duduk di ujung kanan.

Kode 10.10 adalah tur Tkinter-nya, dengan `pack` dan `grid`.

Kode 10.10 `layout_tour.py`: `pack` dan `grid` sedang bekerja; uji mandirinya mencetak geometri yang ditetapkan Tk.

```python
"""layout_tour.py: geometry manager pack dan grid milik Tkinter sedang bekerja."""
import sys
import tkinter as tk

NAMES = ["Button 1", "Button 2", "Button 3", "Long-Named Button 4", "5"]


def row_of_buttons(master):  # pack side=LEFT: satu baris, seperti FlowLayout
    f = tk.Frame(master)
    for n in NAMES:
        tk.Button(f, text=n).pack(side=tk.LEFT, padx=2, pady=2)
    return f


def border_like(master):  # urutan pack menentukan siapa yang mendapat tepi
    f = tk.Frame(master)
    tk.Button(f, text="Button 1 (top)").pack(side=tk.TOP, fill=tk.X)
    tk.Button(f, text="Long-Named Button 4 (bottom)").pack(side=tk.BOTTOM, fill=tk.X)
    tk.Button(f, text="Button 3 (left)").pack(side=tk.LEFT, fill=tk.Y)
    tk.Button(f, text="5 (right)").pack(side=tk.RIGHT, fill=tk.Y)
    tk.Button(f, text="Button 2 (center)").pack(fill=tk.BOTH, expand=True)
    return f


def column_of_buttons(master):  # pack side=TOP: satu kolom, seperti BoxLayout
    f = tk.Frame(master)
    for n in NAMES:
        tk.Button(f, text=n).pack(side=tk.TOP)
    return f


def form_grid(master):  # grid: baris dan kolom, seperti GridBagLayout
    f = tk.Frame(master)
    for row, text in enumerate(["Customer", "Phone", "Weight (kg)"]):
        tk.Label(f, text=text).grid(row=row, column=0, sticky="e", padx=4, pady=2)
        tk.Entry(f, width=10).grid(row=row, column=1, sticky="ew", padx=4, pady=2)
    tk.Button(f, text="Add order").grid(row=3, column=1, sticky="e", padx=4, pady=2)
    f.columnconfigure(1, weight=1)  # kolom 1 mengambil sisa lebar
    return f


def name(w):
    return w.cget("text") if w.winfo_class() != "Entry" else "[entry]"
```

Kode 10.10 (Lanjutan) *layout_tour.py: pack dan grid sedang bekerja; uji mandirinya mencetak geometri yang ditetapkan Tk.*

```
def self_test():
    root = tk.Tk()
    demos = [("pack LEFT", row_of_buttons, 560, 40),
             ("pack LEFT", row_of_buttons, 230, 100),
             ("pack edges", border_like, 460, 150),
             ("pack TOP", column_of_buttons, 230, 170),
             ("grid", form_grid, 300, 120)]
    frames = []
    for title, make, w, h in demos:
        holder = tk.Frame(root, width=w, height=h)
        holder.pack_propagate(False)  # pertahankan ukuran yang kita minta
        holder.pack(anchor="w")
        inner = make(holder)
        inner.pack(fill=tk.BOTH, expand=True)
        frames.append((title, inner, w, h))
    root.geometry("+0+0")
    root.update()  # Tk menghitung geometri hanya untuk jendela yang tampil
    for title, inner, w, h in frames:
        print(f"{title} in {w} x {h}:")
        for c in inner.winfo_children():
            print(f"  {name(c):31} x={c.winfo_x():3} y={c.winfo_y():3}"
                  f" w={c.winfo_width():3} h={c.winfo_height():3}")
    root.destroy()


def main():
    if "--selftest" in sys.argv:
        self_test()
        return
    root = tk.Tk()
    root.title("Layout tour")
    for make in (row_of_buttons, border_like, column_of_buttons, form_grid):
        make(root).pack(fill=tk.X, pady=6)
    root.mainloop()


if __name__ == "__main__":
    main()
```

```
pack LEFT in 560 x 40:
  Button 1                        x=  2 y=  4 w= 90 h= 31
  Button 2                        x= 96 y=  4 w= 90 h= 31
  Button 3                        x=190 y=  4 w= 90 h= 31
  Long-Named Button 4             x=284 y=  4 w=185 h= 31
  5                               x=473 y=  4 w= 39 h= 31
pack LEFT in 230 x 100:
  Button 1                        x=  2 y= 34 w= 90 h= 31
  Button 2                        x= 96 y= 34 w= 90 h= 31
  Button 3                        x=190 y= 34 w= 38 h= 31
  Long-Named Button 4             x=  0 y=  0 w=  1 h=  1
  5                               x=  0 y=  0 w=  1 h=  1
pack edges in 460 x 150:
  Button 1 (top)                  x=  0 y=  0 w=460 h= 31
  Long-Named Button 4 (bottom)    x=  0 y=119 w=460 h= 31
  Button 3 (left)                 x=  0 y= 31 w=127 h= 88
  5 (right)                       x=374 y= 31 w= 86 h= 88
  Button 2 (center)               x=127 y= 31 w=247 h= 88
pack TOP in 230 x 170:
  Button 1                        x= 70 y=  0 w= 90 h= 31
  Button 2                        x= 70 y= 31 w= 90 h= 31
  Button 3                        x= 70 y= 62 w= 90 h= 31
  Long-Named Button 4             x= 22 y= 93 w=185 h= 31
  5                               x= 95 y=124 w= 39 h= 31
grid in 300 x 120:
  Customer                        x= 16 y=  3 w= 73 h= 21
  [entry]                         x= 97 y=  2 w=199 h= 23
  Phone                           x= 41 y= 30 w= 48 h= 21
  [entry]                         x= 97 y= 29 w=199 h= 23
  Weight (kg)                     x=  4 y= 57 w= 85 h= 21
  [entry]                         x= 97 y= 56 w=199 h= 23
  Add order                       x=195 y= 83 w=101 h= 31
```

Bandingkan kedua keluaran, karena di sinilah perbedaan terpenting antara tata letak sederhana kedua toolkit.

- **`pack(side=LEFT)` tidak membungkus.** Di frame selebar 230 piksel, *Button 1* dan *Button 2* mendapat ukuran penuhnya, *Button 3* terjepit menjadi 38 piksel, dan *Long-Named Button 4* serta *5* sama sekali tidak mendapat ruang (lebar dan tinggi 1 adalah cara Tk mengatakan "tidak ditampilkan"). `pack` membagikan ruang sesuai urutan widget di-*pack* sampai ruangnya habis; tidak seperti `FlowLayout`, ia tidak pernah memulai baris baru. Bila Anda membutuhkan pembungkusan di Tkinter, pakailah `grid` dan hitung sendiri barisnya.
- **`pack` ke tepi-tepi meniru `BorderLayout`**, tetapi *urutan*-lah yang menentukan siapa memiliki pojok. Kita mem-*pack* atas dan bawah lebih dulu, jadi keduanya merentang selebar penuh, dan tombol kiri serta kanan hanya mendapat tinggi di antaranya — bentuk yang sama dengan **Gambar 10.9**. Seandainya kiri dan kanan di-*pack* lebih dulu, keduanya akan setinggi penuh dan atas serta bawah akan terjepit di antaranya. Widget terakhir, di-*pack* dengan `fill=BOTH, expand=True`, memainkan peran `CENTER`.
- **`pack(side=TOP)`** adalah sebuah kolom, secara bawaan di tengah, seperti `BoxLayout`.
- **`grid`** menghasilkan ulang formulir `GridBagLayout` nyaris persis: label rata kanan dengan `sticky="e"`, entry direntangkan dengan `sticky="ew"` di kolom berbobot 1.

Uji mandiri Tk harus menampilkan jendelanya sekejap (`root.update()` setelah `root.geometry("+0+0")`), karena Tk menghitung geometri sungguhan hanya untuk jendela yang tampil di layar; di server, `xvfb-run` menyediakan layar yang tak terlihat.

Tabel 10.3 menjajarkan pilihan-pilihannya.

Tabel 10.3 Layout manager Swing dan geometry manager Tkinter untuk kebutuhan umum.

Kebutuhan	Swing	Tkinter
Baris yang membungkus, seperti kata	`FlowLayout` (bawaan untuk `JPanel`)	tidak ada padanan langsung; `grid` dengan baris yang dihitung sendiri
Satu baris atau satu kolom	`BoxLayout` / `Box`	`pack(side=LEFT)` / `pack(side=TOP)`
Tengah ditambah empat tepi	`BorderLayout` (bawaan untuk content pane)	`pack` ke sisi-sisi, tepi lebih dulu, lalu `fill=BOTH, expand=True`
Formulir: baris dan kolom, sebagian merentang	`GridBagLayout` + `GridBagConstraints`	`grid(row, column, sticky)` + `columnconfigure(weight)`
Sel yang sama besar	`GridLayout(rows, cols)`	`grid` dengan kolom `uniform`
Posisi mutlak (hindari)	`setLayout(null)` + `setBounds`	`place(x, y)`

> **WAWASAN KUNCI:** Jendela sungguhan dibangun dengan *menyarangkan*: sebuah `BorderLayout` untuk bentuk keseluruhan, sebuah `GridBagLayout` (atau `grid`) untuk formulir di salah satu wilayah, sebuah `FlowLayout` (atau frame yang di-*pack*) untuk sebaris tombol di dalam formulir. Setiap kontainer menyelesaikan satu masalah tata letak yang sederhana. Bila sebuah tata letak tampak mustahil, jawabannya hampir selalu satu tingkat panel lagi, bukan pengelola yang lebih cerdik.

10.9 Melukis: paintComponent dan Canvas

Komponen bawaan toolkit tahu cara menggambar dirinya sendiri. Bila Anda menginginkan sesuatu yang tidak disediakan komponen mana pun — kotak abu-abu yang menggelap, sebuah diagram, denah mesin-mesin laundry — Anda menggambarnya sendiri. Kedua toolkit menempuh pendekatan yang berlawanan di sini, dan memahami keduanya akan membuat Anda lebih mahir di masing-masing.

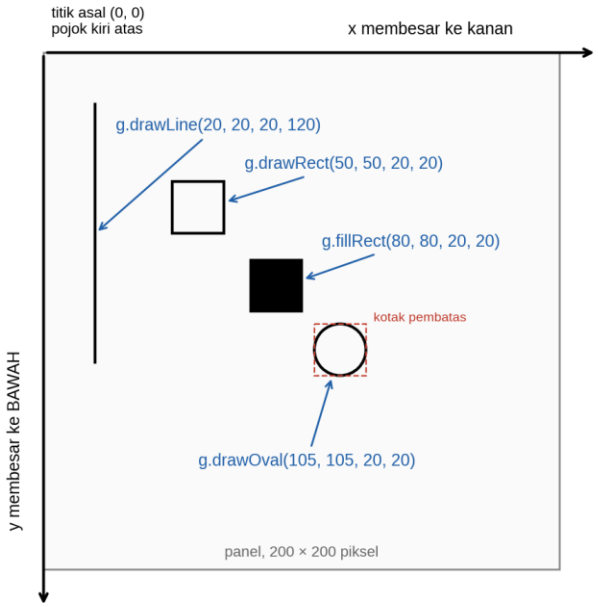
10.9.1 Sistem koordinat, dan objek Graphics

Sebuah `JPanel` memiliki metode

```
protected void paintComponent(Graphics g)
```

yang di-*override* subkelas untuk melukis bentuk apa pun. Parameter `g` adalah objek `Graphics`: sebuah pena, sudah diarahkan ke panel, dengan warna dan fon saat ini. Operasi gambarnya memakai sistem koordinat yang pasti mengejutkan setiap orang sekali: titik asal (0,0) adalah pojok **kiri atas** panel, x membesar ke kanan, dan y membesar ke **bawah**, karena layar digambar baris demi baris dari atas. **Gambar 10.12** memperlihatkan empat operasi dasar:

```
g.drawLine(x1, y1, x2, y2);      // garis dari (x1, y1) ke (x2, y2)
g.drawRect(x, y, xSize, ySize);  // garis luar persegi panjang; (x, y) pojok kiri atasnya
g.fillRect(x, y, xSize, ySize);  // persegi panjang yang sama, terisi
g.drawOval(x, y, xSize, ySize);  // elips yang pas di dalam persegi panjang itu
```



Gambar 10.12 Sistem koordinat Swing: titik asal adalah pojok kiri atas dan *y* membesar ke bawah. Oval ditentukan oleh kotak pembatasnya, bukan oleh pusatnya.

Perhatikan bahwa oval ditentukan oleh *kotak pembatasnya* (*bounding box*): `drawOval(105, 105, 20, 20)` menggambar lingkaran berdiameter 20 yang pusatnya di (115,115). Masih banyak operasi lain — `fillOval`, `drawString`, `drawPolygon`, `drawArc`, `drawImage` — dan subkelas `Graphics2D`, yang sebenarnya merupakan tipe parameter `g` di Swing, menambahkan tebal garis, *antialiasing*, transformasi, dan bentuk sembarang.

10.9.2 Warna

Warna dinyatakan dalam model **RGB**: banyaknya cahaya merah, hijau, dan biru, masing-masing bilangan bulat dari 0 sampai 255. Makin besar angkanya, makin banyak cahaya itu, jadi makin besar makin terang. `new Color(0, 0, 0)` adalah hitam (tanpa cahaya), `new Color(255, 255, 255)` adalah putih (ketiganya penuh), dan `new Color(255, 0, 0)` adalah merah terang. Bila ketiga banyaknya sama, hasilnya adalah suatu gradasi abu-abu. Banyaknya warna berbeda yang dapat dinyatakan model ini adalah

$$256 \times 256 \times 256 = 256^3 = 16{,}777{,}216 \tag{10.1}$$

seperti dihitung Persamaan (10.1): 256 pilihan untuk masing-masing dari tiga kanal yang saling bebas. Kelas `Color` juga memiliki konstanta untuk warna-warna umum (`Color.RED`, `Color.BLUE`, `Color.WHITE`, ...). `g.setColor(c)` menyetel warna untuk semua operasi lukis *berikutnya* dari objek `Graphics` itu, jadi ia dipanggil sebelum bentuk yang hendak dipengaruhinya. Tk menulis warna yang sama sebagai string, dengan nama (`"blue"`) atau sebagai RGB heksadesimal, `"#rrggbb"`: `"#ff0000"` merah terang, `"#0a0a0a"` abu-abu sangat gelap (10, 10, 10).

10.9.3 Siapa yang memanggil paintComponent?

Anda tidak pernah memanggil `paintComponent` sendiri. Sistem *run-time* Java memanggilnya setiap kali panel perlu dilukis: ketika jendela pertama kali muncul, ketika dipulihkan setelah diminimalkan, ketika jendela lain yang menutupinya bergeser, ketika ukurannya diubah. Karena

itu metode Anda harus mampu melukis *seluruh* tampilan panel saat ini, dari awal, kapan saja — artinya semua yang digambarnya harus berasal dari field-field panel, bukan dari apa yang kebetulan tergambar terakhir kali.

Ketika program Anda mengubah keadaan panel dan ingin tampilannya mengikuti, ia memanggil

```
void repaint()
```

yang tidak langsung melukis apa pun. Ia menandai panel sebagai "kotor" lalu kembali [10]; gelung peristiwa kemudian memanggil `paintComponent` (baris 9 **Pseudokode 10.1**). Beberapa panggilan `repaint()` yang beruntun digabung menjadi satu kali pelukisan. Itulah sebabnya listener yang mengubah sebuah field lalu memanggil `repaint()` tidak pernah melihat efeknya selama ia sendiri masih berjalan: pelukisan terjadi setelah listener kembali.

Baris pertama setiap `paintComponent` sebaiknya `super.paintComponent(g)`. Ia melukis latar belakang panel (menghapus gambar sebelumnya) dan wajib ada: tanpanya, gambar lama tetap terlihat di bawah gambar baru, dan di sebagian sistem muncul sampah acak.

10.9.4 Kotak yang menggelap dan menerang

Kode 10.11 menggabungkan pelukisan, peristiwa aksi, peristiwa tetikus, dan `getSource()`. Sebuah panel menggambar kotak abu-abu. Dua tombol, *Darken* dan *Lighten*, membuatnya 10 satuan lebih gelap atau lebih terang; mengeklik kotak itu sendiri juga menerangkannya. Tingkat keabuan tidak boleh di bawah 0 atau di atas 255, dan tombol yang akan mendorongnya melewati batas dinonaktifkan.

Kode 10.11 `ShadeBox.java`: pelukisan di `paintComponent`, dua tombol yang dibedakan dengan `getSource()`, dan klik tetikus pada kotak yang dilukis.

```
import java.awt.Color;
import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.Rectangle;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import java.awt.image.BufferedImage;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.SwingUtilities;

public class ShadeBox extends JPanel implements ActionListener, MouseListener {
    static final Rectangle BOX = new Rectangle(10, 40, 100, 50);
    private final JButton darken = new JButton("Darken");
    private final JButton lighten = new JButton("Lighten");
    private int level = 255;  // 255 = putih, 0 = hitam

    public ShadeBox() {
        setPreferredSize(new Dimension(300, 150));
        add(darken);
        add(lighten);
        darken.addActionListener(this);
        lighten.addActionListener(this);
        addMouseListener(this);  // panel itu sendiri menghasilkan peristiwa tetikus
        lighten.setEnabled(false);
    }

    @Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);  // wajib: lukis latar belakang lebih dulu
        g.setColor(new Color(level, level, level));
        g.fillRect(BOX.x, BOX.y, BOX.width, BOX.height);
        g.setColor(Color.BLACK);
        g.drawRect(BOX.x, BOX.y, BOX.width, BOX.height);
    }

    static int clamp(int x) {
        return Math.max(0, Math.min(255, x));
    }
```

Kode 10.11 (Lanjutan) *ShadeBox.java: pelukisan di paintComponent, dua tombol yang dibedakan dengan getSource(), dan klik tetikus pada kotak yang dilukis.*

```
    private void shade(int delta) {
        level = clamp(level + delta);
        darken.setEnabled(level > 0);
        lighten.setEnabled(level < 255);
        repaint();  // minta agar paintComponent segera dijalankan
    }

    @Override
    public void actionPerformed(ActionEvent e) {
        if (e.getSource() == darken) {  // tombol yang mana?
            shade(-10);
        } else if (e.getSource() == lighten) {
            shade(+10);
        }
    }

    @Override
    public void mouseClicked(MouseEvent e) {
        if (BOX.contains(e.getPoint())) {
            shade(+10);
        }
    }

    @Override public void mousePressed(MouseEvent e) { }
    @Override public void mouseReleased(MouseEvent e) { }
    @Override public void mouseEntered(MouseEvent e) { }
    @Override public void mouseExited(MouseEvent e) { }

    public static void main(String[] args) throws Exception {
        if (args.length > 0 && args[0].equals("--selftest")) {
            System.setProperty("java.awt.headless", "true");
            SwingUtilities.invokeAndWait(ShadeBox::selfTest);
        } else {
            SwingUtilities.invokeLater(() -> {
                JFrame.setDefaultLookAndFeelDecorated(true);
                JFrame frame = new JFrame("Shade box");
                frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                frame.add(new ShadeBox());
                frame.pack();
                frame.setVisible(true);
            });
        }
    }
```

Kode 10.11 (Lanjutan) *ShadeBox.java: pelukisan di paintComponent, dua tombol yang dibedakan dengan getSource(), dan klik tetikus pada kotak yang dilukis.*

```
    String state() {
        return "level=" + level + " darken=" + (darken.isEnabled() ? "on" : "off")
                + " lighten=" + (lighten.isEnabled() ? "on" : "off");
    }

    void click(int x, int y) {  // kirim klik tetikus persis seperti yang dilakukan AWT
        dispatchEvent(new MouseEvent(this, MouseEvent.MOUSE_CLICKED,
                System.currentTimeMillis(), 0, x, y, 1, false, MouseEvent.BUTTON1));
    }

    static void selfTest() {
        ShadeBox p = new ShadeBox();
        System.out.println("start:           " + p.state());
        p.darken.doClick();
        System.out.println("darken x1:       " + p.state());
        for (int i = 0; i < 30; i++) {
            p.darken.doClick();  // tombol yang nonaktif mengabaikan klik
        }
        System.out.println("darken x30 more: " + p.state());
        p.click(60, 65);
        System.out.println("click in box:    " + p.state());
        p.click(250, 130);
        System.out.println("click outside:   " + p.state());
        p.setSize(300, 150);
        BufferedImage img = new BufferedImage(300, 150, BufferedImage.TYPE_INT_RGB);
        p.paint(img.getGraphics());  // jalankan kode pelukisan ke dalam sebuah citra
        System.out.printf("pixel in box:    #%06x%n", img.getRGB(60, 65) & 0xFFFFFF);
        System.out.printf("pixel outside:   #%06x%n", img.getRGB(250, 130) & 0xFFFFFF);
    }
}
```

Kelas ini mendeklarasikan bahwa ia mendengarkan dua jenis peristiwa: `implements ActionListener, MouseListener`. Konstruktornya mendaftarkan panel pada kedua tombol (`addActionListener(this)`) dan pada dirinya sendiri untuk peristiwa tetikus (`addMouseListener(this)` — panel adalah komponen yang diklik tetikus). Hanya ada satu `actionPerformed` untuk kedua tombol, jadi ia memakai `e.getSource() == darken` untuk membedakannya; `==` benar di sini, karena kita menanyakan apakah sumbernya adalah *objek yang sama persis* dengan tombol `darken` kita (kesamaan referensi dari Bab 3).

Jantung kelas ini adalah `shade(delta)`. Ia mengubah keadaan, `level`, dan menjaganya dalam rentang dengan `clamp`, yang menghitung

$$v' = \min(255, \max(0,\ v + \delta)) \tag{10.2}$$

dengan v tingkat saat ini dan δ perubahannya, -10 atau $+10$. Persamaan (10.2) dibaca dari dalam ke luar: tambahkan perubahannya, naikkan hasilnya ke 0 bila menjadi negatif, turunkan ke 255 bila melewatinya. Lalu `shade` mengaktifkan *Darken* hanya selama kotak belum hitam dan *Lighten* hanya selama belum putih, dan memanggil `repaint()`. Ia tidak menggambar. Penggambaran terjadi di `paintComponent`, yang membaca `level` dan melukis kotak sesuai nilainya, kapan pun Swing memutuskan untuk memanggilnya.

`MouseListener` memiliki lima metode, dan kelas yang mengimplementasikan antarmuka itu harus mendefinisikan kelimanya, walaupun kita hanya membutuhkan `mouseClicked`. Empat lainnya berbadan kosong. Ini gangguan yang akan dihilangkan Subbab 10.10 dengan *kelas adaptor*. Penangan klik menanyakan `BOX.contains(e.getPoint())` — apakah titik yang diklik ada di dalam persegi panjang kotak? — sehingga klik di tempat lain pada panel diabaikan.

Berapa kali klik *Darken* diperlukan untuk beralih dari putih ke hitam? Setiap klik mengurangi 10, dan klik terakhir dijepit, jadi jawabannya

$$\lceil \frac{255}{10} \rceil = \lceil 25.5 \rceil = 26 \tag{10.3}$$

menurut Persamaan (10.3): setelah 25 klik tingkatnya $255 - 250 = 5$, dan klik ke-26 akan menghasilkan -5, yang dijepit Persamaan (10.2) menjadi 0. Uji mandiri memeriksa persis hal ini, lalu memeriksa pelukisannya sendiri: ia memberi panel sebuah ukuran, memanggil `paint` pada sebuah `BufferedImage` di memori alih-alih layar — kode lukis yang sama, menggambar ke dalam citra — dan membaca kembali warna dua piksel.

```
start:           level=255 darken=on lighten=off
darken x1:       level=245 darken=on lighten=on
darken x30 more: level=0 darken=off lighten=on
click in box:    level=10 darken=on lighten=on
click outside:   level=10 darken=on lighten=on
pixel in box:    #0a0a0a
pixel outside:   #eeeeee
```

Setelah satu klik tingkatnya 245 dan kedua tombol aktif. Perulangan lalu mengeklik 30 kali lagi; tingkat mencapai 0 setelah klik ke-25 dari antaranya (klik ke-26 secara keseluruhan), *Darken* dinonaktifkan, dan klik-klik sisanya pada tombol nonaktif tidak berbuat apa-apa — `doClick()` pada tombol nonaktif tidak memicu apa pun. Klik di (60,65), di dalam kotak, menerangkannya menjadi 10; klik di (250,130), di luar, tidak mengubah apa pun. Dan piksel di dalam kotak adalah `#0a0a0a`, yaitu merah, hijau, dan biru semuanya sama dengan $0a_{16} = 10$: kode lukis sungguh melukis apa yang dikatakan keadaannya. Piksel di luar adalah latar bawaan panel, abu-abu muda `#eeeeee`. **Gambar 10.13** memperlihatkan program yang sama di layar pada saat kotak sudah menjadi hitam dan *Darken* sudah dinonaktifkan.



Gambar 10.13 Jendela penggelap setelah cukup banyak klik pada *Darken*: kotaknya hitam, dan tombol *Darken* nonaktif karena tingkatnya tidak bisa lebih rendah lagi.

Sebuah varian, di **Gambar 10.14**, menggelapkan dengan tombol, menerangkan dengan klik tetikus, dan menjelaskan keduanya dalam label yang berubah di batas-batasnya — satu `setText` yang menyelamatkan pengguna dari kebingungan mengapa sebuah tombol berhenti bekerja.



Gambar 10.14 Varian yang menggelapkan dengan tombol dan menerangkan dengan klik tetikus, lalu mengatakannya di sebuah label; di sini kotak sudah mencapai hitam dan label melaporkan bahwa ia tidak bisa digelapkan lagi.

10.9.5 Canvas Tkinter: bentuk yang mengingat dirinya sendiri

Kode 10.12 adalah kotak yang sama dalam Tkinter.

Kode 10.12 `shade_box.py`: kotak penggelap di atas `Canvas` Tkinter, tempat kotak itu adalah item yang diingat dan digambar ulang oleh Tk.

```
"""shade_box.py: gelapkan dan terangkan kotak abu-abu yang digambar di kanvas Tkinter."""
import sys
import tkinter as tk


def clamp(x):
    return max(0, min(255, x))


class ShadeBox(tk.Frame):
    def __init__(self, master):
        super().__init__(master)
        self.level = 255  # 255 = putih, 0 = hitam
        bar = tk.Frame(self)
        bar.pack(side=tk.TOP, pady=4)
        self.darken = tk.Button(bar, text="Darken", command=lambda: self.shade(-10))
        self.lighten = tk.Button(bar, text="Lighten", command=lambda: self.shade(+10),
                                 state=tk.DISABLED)
        self.darken.pack(side=tk.LEFT, padx=2)
        self.lighten.pack(side=tk.LEFT, padx=2)
        self.canvas = tk.Canvas(self, width=300, height=110, highlightthickness=0)
        self.canvas.pack()
        # Item kanvas adalah objek yang tetap di kanvas sampai dihapus.
        self.box = self.canvas.create_rectangle(10, 10, 110, 60,
                                                fill=self.color(), outline="black")
        self.canvas.tag_bind(self.box, "<Button-1>", self.on_box_click)

    def color(self):
        return "#%02x%02x%02x" % (self.level, self.level, self.level)

    def shade(self, delta):
        self.level = clamp(self.level + delta)
        self.darken.config(state=tk.NORMAL if self.level > 0 else tk.DISABLED)
        self.lighten.config(state=tk.NORMAL if self.level < 255 else tk.DISABLED)
        self.canvas.itemconfig(self.box, fill=self.color())  # Tk menggambar ulang dengan
sendirinya

    def on_box_click(self, event):  # Tk memanggil ini hanya untuk klik pada kotak
        self.shade(+10)

    def state(self):
        return (f"level={self.level}"
                f" darken={'on' if self.darken['state'] == tk.NORMAL else 'off'}"
                f" lighten={'on' if self.lighten['state'] == tk.NORMAL else 'off'}")
```

Kode 10.12 (Lanjutan) *shade_box.py: kotak penggelap di atas Canvas Tkinter, tempat kotak itu adalah item yang diingat dan digambar ulang oleh Tk.*

```
def self_test():
    root = tk.Tk()
    root.withdraw()
    p = ShadeBox(root)
    p.pack()
    print("start:          ", p.state())
    p.darken.invoke()
    print("darken x1:      ", p.state())
    for _ in range(30):
        p.darken.invoke()  # tombol yang nonaktif mengabaikan invoke()
    print("darken x30 more:", p.state())
    p.on_box_click(None)  # yang dipanggil Tk untuk klik pada kotak
    print("click in box:   ", p.state())
    print("box fill:       ", p.canvas.itemcget(p.box, "fill"))
    print("canvas items:   ", p.canvas.find_all())
    root.destroy()


def main():
    if "--selftest" in sys.argv:
        self_test()
        return
    root = tk.Tk()
    root.title("Shade box")
    ShadeBox(root).pack()
    root.mainloop()


if __name__ == "__main__":
    main()
```

```
start:           level=255 darken=on lighten=off
darken x1:       level=245 darken=on lighten=on
darken x30 more: level=0 darken=off lighten=on
click in box:    level=10 darken=on lighten=on
box fill:        #0a0a0a
canvas items:    (1,)
```

Tidak ada `paintComponent` dan tidak ada `repaint()`. `Canvas` Tkinter bekerja dalam **mode tertahan** (*retained mode*): `create_rectangle(10, 10, 110, 60, fill=...)` tidak menggambar piksel lalu melupakannya, seperti `fillRect`; ia menciptakan sebuah *item*, objek yang hidup di kanvas dan memiliki pengenal (di sini 1, seperti diperlihatkan baris terakhir keluaran). Tk mengingat semua item dan menggambarnya ulang sendiri kapan pun jendela membutuhkannya. Untuk mengubah gambar, Anda mengubah item — `itemconfig(self.box, fill=...)` — atau memindahkannya (`move`, `coords`) atau menghapusnya (`delete`), dan Tk memperbarui layar sendiri. Pendekatan Swing disebut **mode langsung** (*immediate mode*): panel tidak menyimpan gambar apa pun, hanya keadaan Anda, dan `paintComponent` mengubah keadaan menjadi piksel setiap kali.

Mode tertahan membawa hadiah kedua: item dapat menerima peristiwa. `canvas.tag_bind(self.box, "<Button-1>", self.on_box_click)` mengikat callback pada *item kotak*, bukan pada seluruh kanvas, sehingga Tk sendiri yang memutuskan apakah klik mengenai kotak — tidak perlu uji `contains`. Uji mandiri memanggil `on_box_click(None)` karena alasan yang dijelaskan di Subbab 10.7.1, dan memakai `itemcget(self.box, "fill")` untuk membaca kembali warnanya, padanan mode tertahan untuk membaca sebuah piksel.

> **WAWASAN KUNCI:** Di Swing (mode langsung) aturannya: *keadaan di field, piksel di `paintComponent`, dan `repaint()` untuk menghubungkan keduanya*. Di kanvas Tk (mode

tertahan) aturannya: *ciptakan item sekali, lalu ubah itemnya*. Keduanya menjaga gambar sebagai fungsi yang setia dari keadaan program; bedanya hanya pada siapa yang menyimpan gambarnya.

10.10 Kelas Dalam, Kelas Anonim, dan Lambda

Sejauh ini panel-panel kita menjadi listener bagi dirinya sendiri: `class ShadeBox extends JPanel implements ActionListener, MouseListener`. Cara itu berhasil, tetapi punya tiga kekurangan yang membesar seiring jendela. Pertama, satu `actionPerformed` harus melayani setiap tombol, sehingga ia penuh dengan uji `if (e.getSource() == …)`. Kedua, kelas harus mendefinisikan kelima metode `MouseListener` walaupun hanya membutuhkan satu. Ketiga, antarmuka publik panel kini memuat `actionPerformed` dan `mouseClicked`, metode yang semestinya tidak pernah dipanggil kelas lain. Jendela laundry memiliki tujuh reaksi berbeda terhadap tujuh peristiwa berbeda; satu metode dengan tujuh cabang tidak akan terbaca.

Jawaban Java, sejak JDK 1.1, adalah **kelas dalam** (*inner class*); Java 8 menambahkan **ekspresi lambda** sebagai bentuk yang lebih ringkas untuk kasus yang paling umum. Jawaban Python adalah bahwa fungsi adalah objek dan dapat didefinisikan di mana saja, termasuk di dalam fungsi lain, tempat ia menjadi **closure**.

10.10.1 Kelas dalam

DEFINISI. Kelas dalam (*inner class*) adalah kelas yang dideklarasikan di dalam badan kelas lain (kelas *luar*) tanpa pengubah `static`. Setiap objek kelas dalam diciptakan *di dalam* sebuah objek kelas luar dan menyimpan referensi tersembunyi kepadanya, sehingga objek dalam dapat memakai field dan metode objek luar — bahkan yang `private` — seolah-olah miliknya sendiri. Referensi objek dalam ke objek luarnya dapat ditulis secara eksplisit sebagai `Outer.this`.

Kelas dalam sering dipakai untuk mendefinisikan listener peristiwa. Alih-alih satu `actionPerformed` yang menanyakan tombol mana yang diklik, kita menulis satu kelas kecil per reaksi:

```
private class DarkenListener implements ActionListener {
    @Override
    public void actionPerformed(ActionEvent e) {
        shade(-10);           // metode milik panel luar
    }
}
...
darken.addActionListener(new DarkenListener());
```

Kelas dalam dapat mengakses variabel instans kelas luar (`level`, `darken`, `lighten`) maupun metodenya (`shade`, `repaint()`), dan sebuah instansnya dapat diberikan sebagai listener. Setiap peristiwa kini ditangani terpisah dalam metode yang mandiri — `DarkenListener.actionPerformed`, `LightenListener.actionPerformed` — dan ini menghasilkan struktur yang lebih baik ketika sebuah kelas harus mendengarkan banyak peristiwa.

10.10.2 Kelas anonim dan lambda

Sering kali sebuah kelas dalam dipakai tepat satu kali, di tempat ia dideklarasikan. Java memungkinkan Anda mendeklarasikan dan menginstansiasinya dalam satu ekspresi, tanpa nama, sebagai **kelas anonim** (*anonymous class*):

```
darken.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        shade(-10);
    }
});
```

Bacalah `new ActionListener() { … }` sebagai "ciptakan objek dari sebuah kelas baru tanpa nama yang mengimplementasikan `ActionListener` dengan badan ini". Kelas anonim juga dapat *memperluas* sebuah kelas, dan itu menyelesaikan masalah lima metode: `java.awt.event.MouseAdapter` adalah kelas yang mengimplementasikan `MouseListener`, `MouseMotionListener`, dan `MouseWheelListener` dengan metode kosong, sehingga subkelas anonimnya cukup meng-*override* metode yang dibutuhkan. Implementasi antarmuka listener yang tidak berbuat apa-apa seperti itu disebut **kelas adaptor** (*adapter class*); AWT menyediakan satu untuk setiap antarmuka listener yang metodenya lebih dari satu (`MouseAdapter`, `KeyAdapter`, `WindowAdapter`, …).

Antarmuka yang memiliki tepat satu metode abstrak, seperti `ActionListener`, adalah **antarmuka fungsional** (*functional interface*), dan Java 8 memperkenalkan notasi yang lebih ringkas lagi untuk mengimplementasikannya: **ekspresi lambda**

```
darken.addActionListener(e -> shade(-10));
```

yang berarti: "sebuah `ActionListener` yang `actionPerformed(e)`-nya melakukan `shade(-10)`". Kompilator tahu, dari tipe parameter `addActionListener`, antarmuka dan metode mana yang dimaksud. Bila badannya hanya memanggil metode yang sudah ada dengan parameter yang sama, **referensi metode** (*method reference*) lebih ringkas lagi: `reset.addActionListener(this::resetCount)` memberikan metode `resetCount(ActionEvent e)` milik objek ini sebagai listener [11].

10.10.3 Lima gaya berdampingan

Kode 10.13 menghubungkan lima tombol dengan lima gaya — panel itu sendiri, kelas dalam bernama, kelas anonim, lambda, dan referensi metode — ditambah tiga tombol lagi dengan lambda yang diciptakan di dalam perulangan.

Kode 10.13 `ListenerStyles.java`: lima cara menulis action listener di Java, dan lambda yang diciptakan di dalam perulangan.

```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.ArrayList;
import java.util.List;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.SwingUtilities;

public class ListenerStyles extends JPanel implements ActionListener {
    private int count = 0;
    private final JLabel shown = new JLabel("count = 0");
    private final JButton plusOne = new JButton("+1");     // style 1: the panel itself
    private final JButton plusTwo = new JButton("+2");     // style 2: named inner class
    private final JButton plusThree = new JButton("+3");   // style 3: anonymous class
    private final JButton plusFour = new JButton("+4");    // style 4: lambda
    private final JButton reset = new JButton("Reset");    // style 5: method reference
    private final List<JButton> loopButtons = new ArrayList<>();

    public ListenerStyles() {
        for (JButton b : List.of(plusOne, plusTwo, plusThree, plusFour, reset)) {
            add(b);
        }
        plusOne.addActionListener(this);
        plusTwo.addActionListener(new AddListener(2));
        plusThree.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                bump(3);
            }
        });
        int step = 4;  // effectively final, jadi boleh ditangkap lambda
        plusFour.addActionListener(e -> bump(step));
        reset.addActionListener(this::resetCount);
        for (int n : new int[] {10, 20, 30}) {  // n yang baru untuk setiap putaran
            JButton b = new JButton("+" + n);
            b.addActionListener(e -> bump(n));
            loopButtons.add(b);
            add(b);
        }
        add(shown);
    }
```

Kode 10.13 (Lanjutan) *ListenerStyles.java: lima cara menulis action listener di Java, dan lambda yang diciptakan di dalam perulangan.*

```
    private void bump(int n) {
        count += n;
        shown.setText("count = " + count);
    }

    private void resetCount(ActionEvent e) {
        count = 0;
        shown.setText("count = " + count);
    }

    @Override
    public void actionPerformed(ActionEvent e) {  // gaya 1
        bump(1);
    }

    private class AddListener implements ActionListener {  // gaya 2
        private final int n;

        AddListener(int n) {
            this.n = n;
        }

        @Override
        public void actionPerformed(ActionEvent e) {
            bump(n);  // objek dalam boleh memanggil metode objek luarnya
            System.out.println("  AddListener sees ListenerStyles.this.count = "
                    + ListenerStyles.this.count);
        }
    }

    public static void main(String[] args) throws Exception {
        if (args.length > 0 && args[0].equals("--selftest")) {
            System.setProperty("java.awt.headless", "true");
            SwingUtilities.invokeAndWait(ListenerStyles::selfTest);
        } else {
            SwingUtilities.invokeLater(() -> {
                JFrame.setDefaultLookAndFeelDecorated(true);
                JFrame frame = new JFrame("Listener styles");
                frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                frame.add(new ListenerStyles());
                frame.pack();
                frame.setVisible(true);
            });
        }
    }
```

Kode 10.13 (Lanjutan) *ListenerStyles.java: lima cara menulis action listener di Java, dan lambda yang diciptakan di dalam perulangan.*

```
    static void selfTest() {
        ListenerStyles p = new ListenerStyles();
        for (JButton b : List.of(p.plusOne, p.plusTwo, p.plusThree, p.plusFour)) {
            b.doClick();
            System.out.println(b.getText() + " -> " + p.shown.getText());
        }
        for (JButton b : p.loopButtons) {
            b.doClick();
            System.out.println(b.getText() + " -> " + p.shown.getText());
        }
        p.reset.doClick();
        System.out.println("Reset -> " + p.shown.getText());
        System.out.println("listener classes:");
        System.out.println("  " + p.plusTwo.getActionListeners()[0].getClass().getName());
        System.out.println("  " +
p.plusThree.getActionListeners()[0].getClass().getName());
        System.out.println("  lambda class is hidden: "
                + p.plusFour.getActionListeners()[0].getClass().isHidden());
    }
}
```

```
+1 -> count = 1
  AddListener sees ListenerStyles.this.count = 3
+2 -> count = 3
+3 -> count = 6
+4 -> count = 10
+10 -> count = 20
+20 -> count = 40
+30 -> count = 70
Reset -> count = 0
listener classes:
  ListenerStyles$AddListener
  ListenerStyles$1
  lambda class is hidden: true
```

Keluaran memastikan bahwa kelima gaya melakukan pekerjaan yang sama: $1{,}1 + 2 = 3{,}3 + 3 = 6{,}6 + 4 = 10$, lalu $10 + 10 = 20$, $20 + 20 = 40$, $40 + 30 = 70$, dan *Reset* kembali ke 0. Baris yang dicetak dari dalam `AddListener` memperlihatkan bahwa objek dalam membaca `count` milik objek luar melalui `ListenerStyles.this.count` — saat itu sudah 3, karena `bump(n)` berjalan lebih dulu.

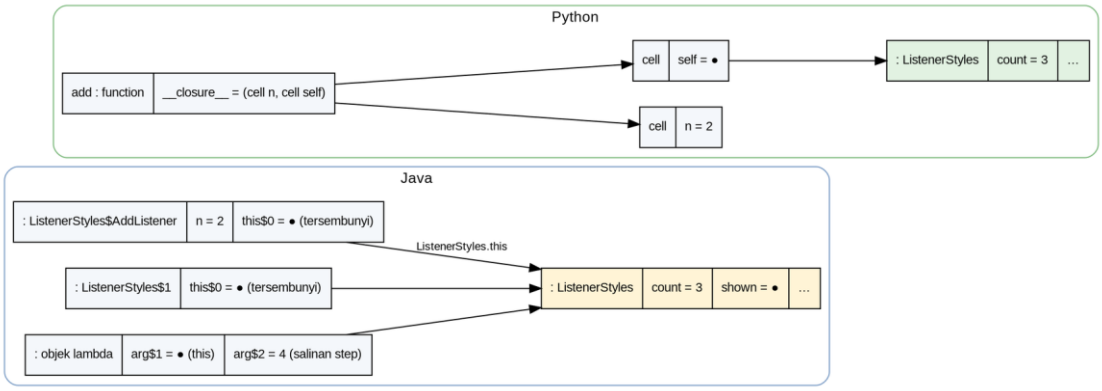
Tiga baris terakhir memperlihatkan apa yang dibuat kompilator dari listener kita. Kelas dalam bernama adalah kelas sungguhan bernama `ListenerStyles$AddListener`; kelas anonimnya mendapat nama `ListenerStyles$1` (kelas anonim pertama di `ListenerStyles`). Mengompilasi berkas itu memang menghasilkan tiga berkas kelas:

```
ListenerStyles$1.class
ListenerStyles$AddListener.class
ListenerStyles.class
```

Lambda tidak menghasilkan berkas kelas: JVM membangkitkan sebuah *kelas tersembunyi* (*hidden class*) untuknya pada saat program berjalan, ketika lambda itu pertama kali dipakai; itulah sebabnya `isHidden()` bernilai `true` dan namanya (sesuatu seperti `ListenerStyles$$Lambda/0x…`) berubah dari satu kali jalan ke kali lainnya.

Gambar 10.15 memperlihatkan objek-objek di balik listener itu. Listener dalam dan listener anonim masing-masing memegang field tersembunyi, yang dinamai kompilator `this$0`, yang menunjuk ke panel yang menciptakannya; begitulah `bump(n)` di dalamnya mencapai `count` milik

panel. Objek lambda memegang nilai-nilai yang *ditangkapnya*: referensi ke panel (karena `bump` adalah metode instans) dan *salinan* `step`.



Gambar 10.15 Apa yang diingat listener. Objek dalam dan objek anonim Java menyimpan referensi tersembunyi ke objek luarnya; objek lambda memegang salinan nilai yang ditangkapnya. Closure Python menyimpan *sel* yang merujuk ke variabel fungsi yang menciptakannya. (Nama-nama field adalah rincian implementasi kompilator.)

10.10.4 Effectively final: mengapa Java menyalin

Lihat lagi `int step = 4;` dan `e -> bump(step)`. Sebuah lambda (atau kelas anonim atau kelas lokal) hanya boleh memakai variabel lokal metode pembungkusnya bila variabel itu **effectively final** — diisi sekali dan tidak pernah diubah. Alasannya adalah salinan di **Gambar 10.15**. Lambda mungkin dipanggil lama setelah metode yang menciptakannya kembali dan variabel lokalnya lenyap dari tumpukan; karena itu Java menyimpan *salinan* nilainya di dalam objek lambda. Bila variabel itu dapat berubah sesudahnya, salinan dan variabelnya akan berselisih, dan program menjadi membingungkan. Java meniadakan kebingungan itu saat kompilasi [11]. Inilah yang terjadi bila kita mencoba perulangan yang wajar:

```
for (int n = 10; n <= 30; n += 10) {
    JButton b = new JButton("+" + n);
    b.addActionListener(e -> count += n);
}
```

```
LoopCapture.java:9: error: local variables referenced from a lambda expression must be
final or effectively final
            b.addActionListener(e -> count += n);
                                              ^
1 error
```

Variabel `n` berubah (`n += 10`), jadi ia tidak dapat ditangkap. Perulangan `for` yang diperluas di **Kode 10.13**, `for (int n : new int[] {10, 20, 30})`, tidak bermasalah, karena ia mendeklarasikan variabel `n` yang *baru* pada setiap putaran, dan masing-masing hanya diisi sekali. Field lain lagi ceritanya: lambda dapat membaca dan mengubah `count` dengan bebas, karena ia mencapai `count` melalui referensi ke panel yang ditangkapnya, bukan melalui salinan. Itu pula sebabnya gelung peristiwa mainan di Subbab 10.6 memakai larik satu elemen: *referensi larik*-nya effectively final, sementara isinya boleh berubah.

10.10.5 Python: bound method, closure, lambda, partial, objek yang dapat dipanggil

Python tidak memiliki antarmuka listener, jadi apa pun yang dapat dipanggil boleh menjadi callback. **Kode 10.14** memperlihatkan lima jenisnya, beserta satu jebakan yang sama-sama mereka miliki.

Kode 10.14 `listener_styles.py`: lima jenis callback Tkinter, jebakan *late binding* dalam perulangan, dan obatnya.

```
"""listener_styles.py: lima cara memberi tombol Tkinter callback-nya."""
import sys
import tkinter as tk
from functools import partial


class Resetter:  # gaya 5: objek yang dapat dipanggil seperti fungsi
    def __init__(self, panel):
        self.panel = panel

    def __call__(self):
        self.panel.count = 0
        self.panel.show()


class ListenerStyles(tk.Frame):
    def __init__(self, master):
        super().__init__(master)
        self.count = 0
        self.shown = tk.Label(self, text="count = 0")

        def make_adder(n):  # gaya 2: fungsi bersarang yang mengingat n
            def add():
                self.bump(n)
            return add

        self.buttons = [
            tk.Button(self, text="+1", command=self.plus_one),        # 1: bound method
            tk.Button(self, text="+2", command=make_adder(2)),        # 2: closure
            tk.Button(self, text="+3", command=lambda: self.bump(3)),  # 3: lambda
            tk.Button(self, text="+4", command=partial(self.bump, 4)),  # 4: partial
        ]
        self.reset = tk.Button(self, text="Reset", command=Resetter(self))
        self.adder_two = make_adder(2)  # disimpan hanya agar uji mandiri dapat
memeriksanya
        self.late_first = None
        self.late = []   # jebakan late binding
        self.fixed = []  # obatnya: argumen bawaan membekukan n sekarang
        for n in (10, 20, 30):
            wrong = lambda: self.bump(n)  # mencari n saat DIPANGGIL, bukan sekarang
            self.late_first = self.late_first or wrong
            self.late.append(tk.Button(self, text=f"+{n}", command=wrong))
            self.fixed.append(tk.Button(self, text=f"+{n}", command=lambda n=n:
self.bump(n)))
        for b in self.buttons + [self.reset] + self.fixed:
            b.pack(side=tk.LEFT)
        self.shown.pack(side=tk.LEFT, padx=6)
```

Kode 10.14 (Lanjutan) *listener_styles.py: lima jenis callback Tkinter, jebakan late binding dalam perulangan, dan obatnya.*

```
    def plus_one(self):
        self.bump(1)

    def bump(self, n):
        self.count += n
        self.show()

    def show(self):
        self.shown.config(text=f"count = {self.count}")


def self_test():
    root = tk.Tk()
    root.withdraw()
    p = ListenerStyles(root)
    for b in p.buttons:
        b.invoke()
        print(b.cget("text"), "->", p.shown.cget("text"))
    p.reset.invoke()
    print("Reset ->", p.shown.cget("text"))
    for label, row in (("late", p.late), ("fixed", p.fixed)):
        for b in row:
            b.invoke()
            print(f"{label:5} {b.cget('text')} -> {p.shown.cget('text')}")
        p.reset.invoke()
    code = p.adder_two.__code__
    cells = [c.cell_contents for c in p.adder_two.__closure__]
    print("make_adder(2) remembers", code.co_freevars, "=", cells[0], "and the panel")
    print("first late lambda sees n =", p.late_first.__closure__[0].cell_contents)
    root.destroy()


def main():
    if "--selftest" in sys.argv:
        self_test()
        return
    root = tk.Tk()
    root.title("Listener styles")
    ListenerStyles(root).pack(padx=6, pady=6)
    root.mainloop()


if __name__ == "__main__":
    main()
```

```
+1 -> count = 1
+2 -> count = 3
+3 -> count = 6
+4 -> count = 10
Reset -> count = 0
late  +10 -> count = 30
late  +20 -> count = 60
late  +30 -> count = 90
fixed +10 -> count = 10
fixed +20 -> count = 30
fixed +30 -> count = 60
make_adder(2) remembers ('n', 'self') = 2 and the panel
first late lambda sees n = 30
```

1. **Bound method**, `self.plus_one`, adalah metode beserta objek pemiliknya. Tk memanggilnya tanpa argumen (callback `command` tidak menerima argumen), dan Python menyertakan `self`. Ini gaya yang paling umum dan paling dekat dengan `this::method` di Java.

2. **Closure.** `make_adder(2)` mengembalikan fungsi bersarang `add`, yang memakai `n` dan `self` dari fungsi yang menciptakannya. Fungsi yang mengingat variabel fungsi pembungkusnya setelah fungsi itu kembali disebut **closure** [12], dan baris-baris terakhir keluaran memeriksanya: `add.__code__.co_freevars` mendaftar nama-nama yang diingat, `('n', 'self')`, dan `add.__closure__` memegang sebuah **sel** (*cell*) untuk masing-masing, yang pertama berisi 2. **Gambar 10.15** menggambarkan sel-sel itu.
3. **Lambda**, `lambda: self.bump(3)`, adalah fungsi anonim berisi satu ekspresi — padanan lambda Java di Python. Ia juga sebuah closure; ia mengingat `self`.
4. `functools.partial(self.bump, 4)` membangun objek baru yang dapat dipanggil, yang bila dipanggil akan memanggil `self.bump(4)`: argumennya dipatok sekarang, pemanggilannya terjadi kelak.
5. **Objek yang dapat dipanggil.** `Resetter` memiliki metode `__call__`, sehingga instansnya dapat dipanggil seperti fungsi. Inilah yang paling dekat di Python dengan kelas listener bernama di Java, dengan satu perbedaan: kelas Python yang bersarang di kelas lain tidak memiliki referensi tersembunyi ke objek luar, sehingga `Resetter` menerima panel secara eksplisit di konstruktornya.

Sekarang jebakannya. Perulangan pertama menciptakan tiga tombol dengan `lambda: self.bump(n)`, dan ketiganya menambahkan 30. Penyebabnya adalah perbedaan antara *salinan Java* dan *sel Python*. Closure Python tidak menyimpan nilai `n` saat diciptakan; ia menyimpan referensi ke *variabel* `n`, dan mencari nilai variabel itu ketika lambda *dipanggil*. Saat tombol mana pun diklik, perulangan sudah selesai dan `n` bernilai 30, jadi ketiga lambda melihat 30 — baris terakhir keluaran memastikan bahwa lambda pertama "melihat" 30. Inilah perilaku **late binding** (pengikatan terlambat) Python yang terkenal [13]. Java mencegah kekeliruan yang sama dengan menolak mengompilasinya; Python mengizinkannya dan mengejutkan Anda saat program berjalan.

Obatnya adalah perulangan kedua: `lambda n=n: self.bump(n)`. Argumen bawaan dievaluasi *sekali, ketika lambda diciptakan*, sehingga setiap lambda mendapat parameter `n` sendiri dengan nilai saat itu sebagai bawaannya. `partial(self.bump, n)` sama baiknya, begitu pula memanggil fungsi pabrik seperti `make_adder(n)`, yang menciptakan sel baru pada setiap pemanggilan. **Tabel 10.4** merangkum pilihan-pilihannya.

Tabel 10.4 Cara-cara menulis listener atau callback, dan apa yang diingat masing-masing.

Gaya	Java	Python	Yang diingat	Paling cocok untuk
Komponen mengimplementasikan listener	`class P extends JPanel implements ActionListener`	—	dirinya sendiri	hanya program yang sangat kecil
Kelas dalam bernama	`private class DarkenListener implements ActionListener`	objek yang dapat dipanggil (`__call__`)	objek luar (Java: `this$0` tersembunyi)	listener dengan keadaan sendiri atau beberapa metode
Kelas anonim	`new ActionListener() { … }`, `new MouseAdapter() { … }`	—	objek luar, variabel lokal yang effectively final	listener bermetode banyak dan adaptor
Lambda	`e -> shade(-10)`	`lambda: self.shade(-10)`	Java: salinan lokal yang effectively final; Python: sel, dibaca saat dipanggil	reaksi satu baris
Referensi metode / bound method	`this::resetCount`	`self.reset_count`	objeknya	reaksi yang layak diberi nama
Argumen yang dipatok	lambda dengan konstanta yang ditangkap	`functools.partial(f, 4)`, `lambda n=n: f(n)`	nilai-nilai yang diberikan	banyak tombol serupa dalam perulangan

Pedoman Java modern adalah mengutamakan lambda dan referensi metode ketimbang kelas anonim setiap kali antarmukanya hanya bermetode satu, dan kelas anonim (biasanya dari sebuah adaptor) bila metodenya beberapa [14]. Jendela laundry mengikuti pedoman itu, begitu pula program berikut.

10.10.6 Menggambar dengan tetikus

Peristiwa gerak tetikus di **Tabel 10.2** melaporkan setiap gerakan penunjuk (`mouseMoved`) dan setiap gerakan sambil menahan tombol (`mouseDragged`). **Kode 10.15** mengubahnya menjadi program menggambar. Ia memakai satu `MouseAdapter` anonim sebagai listener tetikus (untuk penekanan) sekaligus listener gerak tetikus (untuk seretan dan gerakan).

Kode 10.15 `Doodle.java`: menggambar dengan tetikus, dengan `MouseAdapter` anonim dan metode lukis yang menggambar ulang setiap goresan yang tersimpan.

```
import java.awt.BasicStroke;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Point;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.awt.image.BufferedImage;
import java.util.ArrayList;
import java.util.List;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.SwingUtilities;

public class Doodle extends JPanel {
    private final List<List<Point>> strokes = new ArrayList<>();  // gambarnya
    private final JLabel xy = new JLabel("Use your mouse to draw a picture.");

    public Doodle() {
        setPreferredSize(new Dimension(300, 300));
        add(xy);
        MouseAdapter pen = new MouseAdapter() {  // sebuah kelas dalam anonim
            @Override
            public void mousePressed(MouseEvent e) {
                strokes.add(new ArrayList<>(List.of(e.getPoint())));  // goresan baru
                repaint();
            }

            @Override
            public void mouseDragged(MouseEvent e) {
                strokes.get(strokes.size() - 1).add(e.getPoint());
                report(e);
                repaint();
            }

            @Override
            public void mouseMoved(MouseEvent e) {
                report(e);
            }
        };
        addMouseListener(pen);        // tekan, lepas, klik
        addMouseMotionListener(pen);  // gerak dan seret
    }
```

Kode 10.15 (Lanjutan) *Doodle.java: menggambar dengan tetikus, dengan MouseAdapter anonim dan metode lukis yang menggambar ulang setiap goresan yang tersimpan.*

```
    private void report(MouseEvent e) {
        xy.setText("X: " + e.getX() + ", Y: " + e.getY());
    }

    @Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);
        Graphics2D g2 = (Graphics2D) g;
        g2.setColor(Color.BLUE);
        g2.setStroke(new BasicStroke(12, BasicStroke.CAP_ROUND, BasicStroke.JOIN_ROUND));
        for (List<Point> s : strokes) {  // gambar ulang SEMUANYA dari titik yang tersimpan
            Point a = s.get(0);
            g2.drawLine(a.x, a.y, a.x, a.y);  // sebuah titik, kalau-kalau goresan hanya
satu titik
            for (Point b : s) {
                g2.drawLine(a.x, a.y, b.x, b.y);
                a = b;
            }
        }
    }

    public static void main(String[] args) throws Exception {
        if (args.length > 0 && args[0].equals("--selftest")) {
            System.setProperty("java.awt.headless", "true");
            SwingUtilities.invokeAndWait(Doodle::selfTest);
        } else {
            SwingUtilities.invokeLater(() -> {
                JFrame.setDefaultLookAndFeelDecorated(true);
                JFrame frame = new JFrame("Mouse motion event");
                frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                frame.add(new Doodle());
                frame.pack();
                frame.setVisible(true);
            });
        }
    }

    void send(int id, int x, int y) {
        dispatchEvent(new MouseEvent(this, id, System.currentTimeMillis(),
                MouseEvent.BUTTON1_DOWN_MASK, x, y, 1, false, MouseEvent.BUTTON1));
    }
```

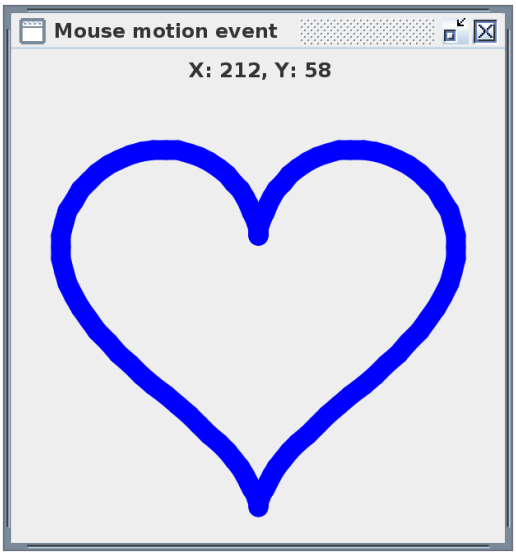
Kode 10.15 (Lanjutan) *Doodle.java: menggambar dengan tetikus, dengan MouseAdapter anonim dan metode lukis yang menggambar ulang setiap goresan yang tersimpan.*

```
    static void selfTest() {
        Doodle d = new Doodle();
        d.send(MouseEvent.MOUSE_PRESSED, 50, 100);
        d.send(MouseEvent.MOUSE_DRAGGED, 100, 100);
        d.send(MouseEvent.MOUSE_DRAGGED, 150, 150);
        d.send(MouseEvent.MOUSE_RELEASED, 150, 150);
        d.send(MouseEvent.MOUSE_MOVED, 200, 60);
        System.out.println("strokes: " + d.strokes.size()
                + ", points in stroke 1: " + d.strokes.get(0).size());
        System.out.println("label:   " + d.xy.getText());
        d.setSize(300, 300);
        BufferedImage img = new BufferedImage(300, 300, BufferedImage.TYPE_INT_RGB);
        d.paint(img.getGraphics());
        System.out.printf("pixel at (75, 100):  #%06x%n", img.getRGB(75, 100) & 0xFFFFFF);
        System.out.printf("pixel at (250, 250): #%06x%n", img.getRGB(250, 250) & 0xFFFFFF);
    }
}
```

```
strokes: 1, points in stroke 1: 3
label:   X: 200, Y: 60
pixel at (75, 100):  #0000ff
pixel at (250, 250): #eeeeee
```

Keadaan panel adalah `strokes`, sebuah `List<List<Point>>`. Menekan memulai goresan baru, menyeret menambahkan satu titik dan memanggil `repaint()`, dan bergerak melaporkan koordinat. `paintComponent` menggambar ulang *setiap* goresan dari titik-titik yang tersimpan dengan pena biru tebal berujung bulat. Uji mandiri mengirim `MouseEvent` sungguhan dengan `dispatchEvent`, persis seperti yang diantarkan AWT — tekan di (50,100), seret ke (100,100) dan (150,150), lepas, gerak ke (200,60) — lalu melukis ke dalam sebuah citra dan menemukan warna biru di tengah ruas pertama serta latar polos di tempat yang jauh.

Mengapa titik-titiknya harus disimpan? Mengapa tidak menggambar setiap ruas langsung di `mouseDragged`, dengan `Graphics g = getGraphics(); g.fillOval(...)`? Jalan pintas itu muncul di banyak tutorial, dan tampaknya berhasil: Anda dapat menggambar hati seperti di **Gambar 10.16**.



Gambar 10.16 Gambar hati yang dibuat dengan menyeret tetikus, dengan koordinat penunjuk ditampilkan di label.

Tetapi minimalkan jendela lalu pulihkan, atau tutupi dengan jendela lain, dan hati itu lenyap. Lukisan yang dibuat di luar `paintComponent` tidak diingat di mana pun; lain kali Swing melukis ulang panel — yang dapat dilakukannya kapan saja, dengan alasannya sendiri — ia memanggil `paintComponent`, yang tidak tahu apa-apa tentang hati itu. Menyimpan titik-titik di sebuah field dan menggambarnya di `paintComponent`-lah yang membuat gambar bertahan. (Untuk panel yang tidak tampil di layar, `getGraphics()` bahkan mengembalikan `null`, sehingga versi jalan pintas itu sama sekali tidak dapat diuji tanpa layar.)

Kode 10.16 melakukan hal yang sama dalam Tkinter, tempat mode tertahan membuat masalah itu lenyap dengan sendirinya: setiap ruas menjadi item garis di kanvas, dan Tk menggambar ulang item-item itu kapan pun diperlukan.

Kode 10.16 `doodle.py`: menggambar dengan tetikus di kanvas Tkinter; setiap ruas adalah item garis yang disimpan Tk.

```
"""doodle.py: menggambar dengan tetikus di kanvas Tkinter."""
import sys
import tkinter as tk
from types import SimpleNamespace


class Doodle(tk.Frame):
    def __init__(self, master):
        super().__init__(master)
        self.xy = tk.Label(self, text="Use your mouse to draw a picture.")
        self.xy.pack()
        self.canvas = tk.Canvas(self, width=300, height=280, bg="white")
        self.canvas.pack()
        self.last = None  # titik sebelumnya pada goresan ini
        self.canvas.bind("<ButtonPress-1>", self.on_press)
        self.canvas.bind("<B1-Motion>", self.on_drag)  # move with button 1 held
        self.canvas.bind("<Motion>", self.report)

    def on_press(self, event):
        self.last = (event.x, event.y)

    def on_drag(self, event):
        x0, y0 = self.last
        # Setiap ruas menjadi item kanvas; Tk mengingat dan menggambarnya ulang.
        self.canvas.create_line(x0, y0, event.x, event.y, fill="blue",
                                width=12, capstyle=tk.ROUND)
        self.last = (event.x, event.y)
        self.report(event)

    def report(self, event):
        self.xy.config(text=f"X: {event.x}, Y: {event.y}")


def self_test():
    root = tk.Tk()
    root.withdraw()
    d = Doodle(root)
    d.pack()

    def at(x, y):  # pengganti objek peristiwa yang akan dikirim Tk
        return SimpleNamespace(x=x, y=y)
```

Kode 10.16 (Lanjutan) *doodle.py: menggambar dengan tetikus di kanvas Tkinter; setiap ruas adalah item garis yang disimpan Tk.*

```
    d.on_press(at(50, 100))
    d.on_drag(at(100, 100))
    d.on_drag(at(150, 150))
    d.report(at(200, 60))
    items = d.canvas.find_all()
    print("canvas items:", len(items))
    print("label:       ", d.xy.cget("text"))
    print("item 1:      ", d.canvas.type(items[0]), d.canvas.coords(items[0]),
          d.canvas.itemcget(items[0], "fill"))
    root.destroy()


def main():
    if "--selftest" in sys.argv:
        self_test()
        return
    root = tk.Tk()
    root.title("Mouse motion event")
    Doodle(root).pack()
    root.mainloop()


if __name__ == "__main__":
    main()
```

```
canvas items: 2
label:        X: 200, Y: 60
item 1:       line [50.0, 100.0, 100.0, 100.0] blue
```

Urutan peristiwa `"<B1-Motion>"` berarti "bergerak sambil menahan tombol 1", nama Tk untuk menyeret. Uji mandiri membangun objek peristiwa pengganti dengan `types.SimpleNamespace(x=..., y=...)`, yang memiliki persis atribut yang dipakai callback, lalu memanggil callback dalam urutan yang akan dipakai Tk: satu tekan dan dua seret menghasilkan dua item garis, yang pertama dari (50,100) ke (100,100) berwarna biru.

10.11 Event Dispatch Thread dan Aturan Thread

Sekarang kita kembali ke pertanyaan yang dibiarkan terbuka di Subbab 10.4: mengapa `main` membuat jendelanya dengan `SwingUtilities.invokeLater`? Jawabannya adalah aturan paling penting dalam pemrograman GUI, dan aturan itu berlaku di Swing, di Tk, di peramban, di Android, dan di iOS.

10.11.1 Satu thread memiliki jendela

Thread adalah rangkaian instruksi yang dieksekusi di dalam sebuah program; sebuah program dapat memiliki beberapa thread yang berjalan bersamaan dan berbagi objek-objeknya. Bab 14 dikhususkan untuk thread dan kerepotan yang dibawanya. Di sini kita hanya memerlukan satu fakta: bila dua thread mengubah objek yang sama pada saat yang sama, objek itu dapat tertinggal dalam keadaan tidak konsisten, kecuali setiap aksesnya dikoordinasikan dengan cermat.

Toolkit GUI memuat objek dalam jumlah luar biasa banyak yang terus berubah — teks, ukuran, posisi, warna, fokus, dan pilihan setiap komponen. Mengoordinasikan setiap akses di antara beberapa thread akan membuat setiap toolkit lambat dan rumit, dan pengalaman sejak 1990-an menunjukkan bahwa toolkit multithread mudah mengalami *deadlock* [15]. Maka hampir setiap toolkit mengambil keputusan yang sama: **semua akses ke GUI terjadi di satu thread**. Di Swing

thread itu adalah **event dispatch thread** (EDT), thread yang menjalankan gelung peristiwa **Pseudokode 10.1**. Di Tkinter thread itu adalah thread yang menjalankan `mainloop()`, biasanya thread utama program.

> **DEFINISI. Aturan thread.** Komponen Swing, dan widget Tkinter, hanya boleh diciptakan, dibaca, dan diubah di thread GUI — event dispatch thread di Swing, thread yang menjalankan `mainloop()` di Tkinter. Sebaliknya, thread GUI tidak boleh dibuat sibuk lama, karena selama ia sibuk tidak ada peristiwa yang ditangani dan tidak ada yang digambar ulang.

Kedua bagian aturan itu menarik ke arah yang berlawanan, dan seni pemrograman GUI adalah memenuhi keduanya.

- **Semua urusan GUI di thread GUI.** Listener sudah berjalan di EDT (gelung peristiwa yang memanggilnya), jadi kode di dalam listener boleh menyentuh komponen dengan bebas. Kode yang berjalan *di tempat lain* — `main`, yang berjalan di thread utama, atau sebuah thread latar — harus mengirim pekerjaan GUI ke EDT. `SwingUtilities.invokeLater(r)` menaruh `Runnable r` ke antrean peristiwa dan langsung kembali; `SwingUtilities.invokeAndWait(r)` melakukan hal yang sama lalu menunggu sampai `r` selesai berjalan. Itulah sebabnya `main` membangun jendela dengan `invokeLater` [16].
- **Tidak ada yang lambat di thread GUI.** Listener yang membaca berkas besar, menanyai basis data, memanggil layanan web, atau berhitung selama satu detik menahan gelung peristiwa di baris 7 **Pseudokode 10.1** selama itu pula. Jendela berhenti menggambar ulang (menjadi kelabu atau putih bila tersingkap), tombol tidak bereaksi, dan sistem operasi mungkin menawarkan untuk mematikan program yang "tidak merespons". Sebagai pegangan, reaksi dalam kira-kira sepersepuluh detik terasa seketika bagi pengguna, dan setelah kira-kira 1 detik alur pikiran pengguna terputus [17]; sebuah penangan harus selesai jauh di bawah batas pertama.

10.11.2 Pekerjaan lambat, cara yang salah dan cara yang benar

Kode 10.17 memiliki dua tombol yang menghitung hal yang sama: pendapatan laundry selama sebulan, dengan pembacaan pesanan setiap hari memakan 20 milidetik (disimulasikan dengan `Thread.sleep`), sehingga seluruh penghitungan makan waktu sekitar 600 ms. Dengan pendapatan harian $100{,}000 + 5{,}000\ (d \bmod 7)$ untuk hari d, total sebulan adalah

$$\sum_{d=1}^{30} (100{,}000 + 5{,}000\ (d \bmod 7)) = 30 \times 100{,}000 + 5{,}000 \times 87 = 3{,}435{,}000 \tag{10.4}$$

karena sisa bagi $d \bmod 7$ untuk $d = 1, \ldots, 30$ adalah empat minggu penuh bernilai $1 + 2 + \cdots + 6 + 0 = 21$, ditambah $1 + 2$ untuk hari ke-29 dan ke-30, yaitu $4 \times 21 + 3 = 87$. Persamaan (10.4) adalah angka yang harus ditampilkan kedua tombol. *Tally (freezes)* menghitungnya langsung di dalam listener-nya. *Tally the month* menyerahkannya kepada **`SwingWorker`**, pembantu Swing untuk menjalankan tugas di thread latar dan mengantarkan hasilnya kembali di EDT.

Kode 10.17 `SlowWork.java`: pekerjaan lambat di EDT membekukan jendela; sebuah `SwingWorker` menjalankannya di thread pekerja dan menampilkan hasilnya di EDT.

```
import java.util.Locale;
import java.util.concurrent.CountDownLatch;
import java.util.concurrent.ExecutionException;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.SwingUtilities;
import javax.swing.SwingWorker;

public class SlowWork extends JPanel {
    private final JButton start = new JButton("Tally the month");
    private final JButton freeze = new JButton("Tally (freezes)");
    private final JLabel status = new JLabel("Ready");
    private Runnable onDone = () -> { };  // memberi tahu uji mandiri bahwa kita sudah
selesai

    public SlowWork() {
        add(start);
        add(freeze);
        add(status);
        freeze.addActionListener(e -> {  // SALAH: pekerjaan lambat di EDT
            log("click handled on");
            long total = tally();         // jendela membeku selama seluruh penghitungan
            status.setText("Month total: " + rupiah(total));
        });
        start.addActionListener(e -> {   // BENAR: serahkan pekerjaan lambat kepada pekerja
            log("click handled on");
            start.setEnabled(false);
            status.setText("Working...");
            new SwingWorker<Long, Void>() {
                @Override
                protected Long doInBackground() {  // berjalan di thread pekerja
                    return tally();
                }

                @Override
                protected void done() {            // berjalan lagi di EDT
                    log("done() runs on");
                    try {
                        status.setText("Month total: " + rupiah(get()));
                    } catch (InterruptedException | ExecutionException ex) {
                        status.setText("Failed: " + ex.getMessage());
                    }
                    start.setEnabled(true);
                    onDone.run();
                }
```

Kode 10.17 (Lanjutan) *SlowWork.java: pekerjaan lambat di EDT membekukan jendela; sebuah SwingWorker menjalankannya di thread pekerja dan menampilkan hasilnya di EDT.*

```
            }.execute();
        });
    }

    static void log(String what) {
        System.out.println(what + " " + Thread.currentThread().getName());
    }

    static String rupiah(long amount) {
        return String.format(Locale.US, "Rp%,d", amount);
    }

    static long tally() {  // anggap membaca pesanan tiap hari butuh 20 ms
        log("tally runs on");
        long total = 0;
        for (int day = 1; day <= 30; day++) {
            try {
                Thread.sleep(20);
            } catch (InterruptedException ex) {
                Thread.currentThread().interrupt();
                return total;
            }
            total += 100_000 + 5_000 * (day % 7);
        }
        return total;
    }

    public static void main(String[] args) throws Exception {
        if (args.length > 0 && args[0].equals("--selftest")) {
            System.setProperty("java.awt.headless", "true");
            selfTest();
        } else {
            SwingUtilities.invokeLater(() -> {
                JFrame.setDefaultLookAndFeelDecorated(true);
                JFrame frame = new JFrame("Slow work");
                frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                frame.add(new SlowWork());
                frame.pack();
                frame.setVisible(true);
            });
        }
    }
```

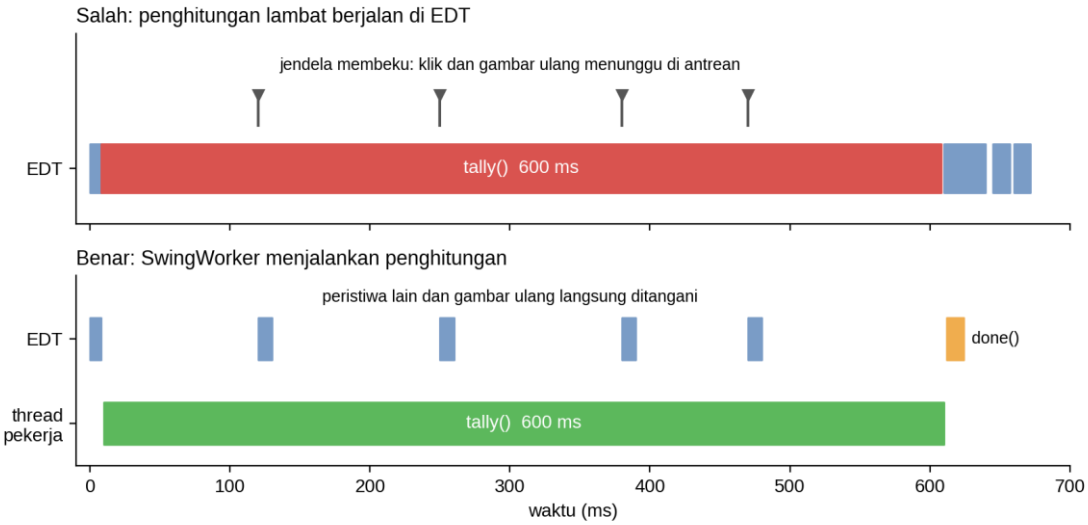
Kode 10.17 (Lanjutan) *SlowWork.java: pekerjaan lambat di EDT membekukan jendela; sebuah SwingWorker menjalankannya di thread pekerja dan menampilkan hasilnya di EDT.*

```
    static void selfTest() throws Exception {
        CountDownLatch finished = new CountDownLatch(1);
        SlowWork[] holder = new SlowWork[1];
        SwingUtilities.invokeAndWait(() -> {
            SlowWork w = new SlowWork();
            holder[0] = w;
            w.onDone = finished::countDown;
            long t0 = System.nanoTime();
            w.freeze.doClick();
            long ms = (System.nanoTime() - t0) / 1_000_000;
            System.out.println("EDT blocked for at least 600 ms: " + (ms >= 600));
            System.out.println("status: " + w.status.getText());
            t0 = System.nanoTime();
            w.start.doClick();
            ms = (System.nanoTime() - t0) / 1_000_000;
            System.out.println("EDT free again within 100 ms: " + (ms < 100));
            System.out.println("status: " + w.status.getText());
        });
        finished.await();  // thread utama menunggu; EDT tetap bebas
        SwingUtilities.invokeAndWait(() ->
                System.out.println("status: " + holder[0].status.getText()));
    }
}
```

```
click handled on AWT-EventQueue-0
tally runs on AWT-EventQueue-0
EDT blocked for at least 600 ms: true
status: Month total: Rp3,435,000
click handled on AWT-EventQueue-0
EDT free again within 100 ms: true
status: Working...
tally runs on SwingWorker-pool-1-thread-1
done() runs on AWT-EventQueue-0
status: Month total: Rp3,435,000
```

Log menyebutkan thread untuk setiap langkah. Untuk tombol yang membekukan, penangan klik *dan* penghitungannya berjalan di `AWT-EventQueue-0`, yaitu EDT, yang tertahan sedikitnya 600 ms — jendela sungguhan akan mati selama itu. Untuk tombol yang benar, penangan menonaktifkan tombol, menampilkan *Working…*, menyalakan pekerja, dan kembali dalam 100 ms; penghitungannya berjalan di `SwingWorker-pool-1-thread-1`; dan `done()` berjalan kembali di EDT, tempat ia aman menyetel label. `get()` mengembalikan hasil `doInBackground()`, atau melempar `ExecutionException` yang membungkus eksepsinya. (Baris dari thread yang berbeda dapat berselang-seling secara berbeda di mesin Anda; nama-nama thread tidak akan berubah.)

`SwingWorker<T, V>` bersifat generik, dalam arti Bab 9: `T` adalah tipe hasil keseluruhan (di sini `Long`) dan `V` tipe hasil antara, yang dapat dikirim pekerja dengan `publish(...)` dan diterima di EDT dalam `process(...)` untuk menggerakkan bilah kemajuan; kita tidak memerlukannya (`Void`). **Gambar 10.17** memperlihatkan kedua jalannya program pada garis waktu.



Gambar 10.17 Aturan thread pada garis waktu. Atas: penghitungan berjalan di EDT, yang tidak dapat menangani klik atau penggambaran ulang selama 600 ms. Bawah: thread pekerja menjalankan penghitungan sementara EDT terus menangani peristiwa; `done()` kembali ke EDT untuk menampilkan hasilnya.

Tkinter tidak memiliki `SwingWorker`, tetapi resepnya sama dan layak dihafal: *nyalakan sebuah thread untuk pekerjaan itu; biarkan thread menaruh hasilnya di antrean yang aman-thread; biarkan thread GUI memeriksa antrean itu dengan pewaktu*. **Kode 10.18** mengikuti resep itu.

Kode 10.18 `slow_work.py`: thread pekerja menghitung, sebuah `queue.Queue` membawa hasilnya, dan `after` memeriksa antrean di thread GUI.

```
"""slow_work.py: jaga jendela Tkinter tetap responsif saat pekerjaan lambat berjalan."""
import queue
import sys
import threading
import time
import tkinter as tk


def log(what):
    print(what, threading.current_thread().name)


def tally():  # anggap membaca pesanan tiap hari butuh 20 ms
    log("tally runs on")
    total = 0
    for day in range(1, 31):
        time.sleep(0.02)
        total += 100_000 + 5_000 * (day % 7)
    return total


class SlowWork(tk.Frame):
    def __init__(self, master):
        super().__init__(master)
        self.start = tk.Button(self, text="Tally the month", command=self.on_start)
        self.freeze = tk.Button(self, text="Tally (freezes)", command=self.on_freeze)
        self.status = tk.Label(self, text="Ready")
        for w in (self.start, self.freeze, self.status):
            w.pack(side=tk.LEFT, padx=4, pady=4)
        self.results = queue.Queue()  # satu-satunya yang disentuh kedua thread

    def on_freeze(self):  # SALAH: pekerjaan lambat di dalam gelung peristiwa
        log("click handled on")
        self.status.config(text=f"Month total: Rp{tally():,}")

    def on_start(self):  # BENAR: pekerjaan lambat di thread pekerja
        log("click handled on")
        self.start.config(state=tk.DISABLED)
        self.status.config(text="Working...")
        threading.Thread(target=self.work, daemon=True).start()
        self.after(50, self.poll)

    def work(self):  # thread pekerja: jangan panggil widget di sini!
        self.results.put(tally())
```

Kode 10.18 (Lanjutan) *slow_work.py: thread pekerja menghitung, sebuah queue.Queue membawa hasilnya, dan after memeriksa antrean di thread GUI.*

```
    def poll(self):  # thread gelung peristiwa: periksa antrean setiap 50 ms
        try:
            total = self.results.get_nowait()
        except queue.Empty:
            self.after(50, self.poll)
            return
        log("result shown on")
        self.status.config(text=f"Month total: Rp{total:,}")
        self.start.config(state=tk.NORMAL)


def self_test():
    root = tk.Tk()
    root.withdraw()
    w = SlowWork(root)
    w.pack()
    t0 = time.perf_counter()
    w.freeze.invoke()
    print("loop blocked for at least 600 ms:", time.perf_counter() - t0 >= 0.6)
    print("status:", w.status.cget("text"))
    t0 = time.perf_counter()
    w.start.invoke()
    print("loop free again within 100 ms:", time.perf_counter() - t0 < 0.1)
    print("status:", w.status.cget("text"))

    def finish():
        if w.status.cget("text").startswith("Month"):
            print("status:", w.status.cget("text"))
            root.destroy()  # mengakhiri mainloop
        else:
            root.after(50, finish)

    root.after(50, finish)
    root.mainloop()


def main():
    if "--selftest" in sys.argv:
        self_test()
        return
    root = tk.Tk()
    root.title("Slow work")
    SlowWork(root).pack()
    root.mainloop()
```

Kode 10.18 (Lanjutan) *slow_work.py: thread pekerja menghitung, sebuah queue.Queue membawa hasilnya, dan after memeriksa antrean di thread GUI.*

```
if __name__ == "__main__":
    main()
```

```
click handled on MainThread
tally runs on MainThread
loop blocked for at least 600 ms: True
status: Month total: Rp3,435,000
click handled on MainThread
loop free again within 100 ms: True
status: Working...
tally runs on Thread-1 (work)
result shown on MainThread
status: Month total: Rp3,435,000
```

`threading.Thread(target=self.work, daemon=True).start()` menjalankan `work` di thread baru (thread *daemon* tidak membuat program tetap hidup setelah jendela ditutup). `work` tidak menyentuh widget apa pun; ia menaruh total ke `self.results`, sebuah `queue.Queue` yang aman-thread. Di thread GUI, `self.after(50, self.poll)` meminta gelung Tk memanggil `poll` dalam 50 ms; `poll` memeriksa antrean tanpa menunggu, lalu menjadwalkan dirinya lagi atau menampilkan hasilnya. Log memastikan bahwa widget diperbarui di `MainThread`, sedangkan penghitungannya berjalan di `Thread-1 (work)`.

Tabel 10.5 mengumpulkan perkakas aturan itu untuk kedua toolkit.

Tabel 10.5 Perkakas aturan thread di Swing dan di Tkinter.

Tugas	Swing	Tkinter
Menjalankan kode GUI dari thread lain, nanti	`SwingUtilities.invokeLater(r)`	taruh pesan di `queue.Queue`; periksa dengan `after`
Menjalankan kode GUI dari thread lain lalu menunggu	`SwingUtilities.invokeAndWait(r)`	(hindari; pakai antrean)
Apakah saya di thread GUI?	`SwingUtilities.isEventDispatchThread()`	`threading.current_thread() is threading.main_thread()`
Pekerjaan lambat di latar	`SwingWorker.doInBackground()`, hasil di `done()`	`threading.Thread(target=...)` ditambah antrean
Kemajuan dari latar	`publish(...)` / `process(...)`	taruh pesan kemajuan di antrean yang sama
Pewaktu di thread GUI	`javax.swing.Timer(ms, listener)`	`widget.after(ms, callback)`

10.12 Profitina Laundry: Jendela Pencatatan Pesanan

Sekarang kita dapat membangun jendela yang sungguh-sungguh dibutuhkan meja laundry. Kebutuhannya datang dari para staf:

- **Formulir.** Nama pelanggan, nomor telepon, layanan (Wash & Fold, Iron Only, atau Wash, Dry & Iron), berat dalam kilogram, dan sebuah kotak *Express*. Harganya adalah harga Bab 7: Wash & Fold Rp7,000 per kilogram dengan minimum 3 kg, selesai dalam 48 jam; Iron Only Rp5,000 per kilogram dengan minimum 2 kg, selesai dalam 24 jam; Wash, Dry & Iron Rp10,000 per kilogram dengan minimum 3 kg, selesai dalam 72 jam. Layanan kilat 50% lebih mahal dan empat kali lebih cepat, tetapi tidak pernah lebih cepat dari 6 jam.
- **Harga yang langsung diperbarui.** Begitu berat, layanan, atau kotak kilat berubah, jendela menampilkan harga, berat yang ditagih, dan lama pengerjaan — sebelum apa pun ditambahkan — sehingga staf dapat memberi tahu pelanggan.
- **Validasi.** *Add order* menerima pesanan hanya bila nama diisi (paling banyak 40 karakter), nomor telepon terdiri atas 10 sampai 13 digit yang diawali 0 (spasi dan tanda hubung boleh ada dan akan dibuang), dan berat berupa bilangan di atas 0 dan paling banyak 50 kg. Staf di Indonesia sering mengetik koma desimal, jadi `4,2` harus diterima sebagaimana `4.2`. Bila ada kekeliruan, jendela menyebutkan apa yang salah dengan warna merah dan menaruh kursor di field yang harus dibetulkan; tidak ada hal lain yang berubah.

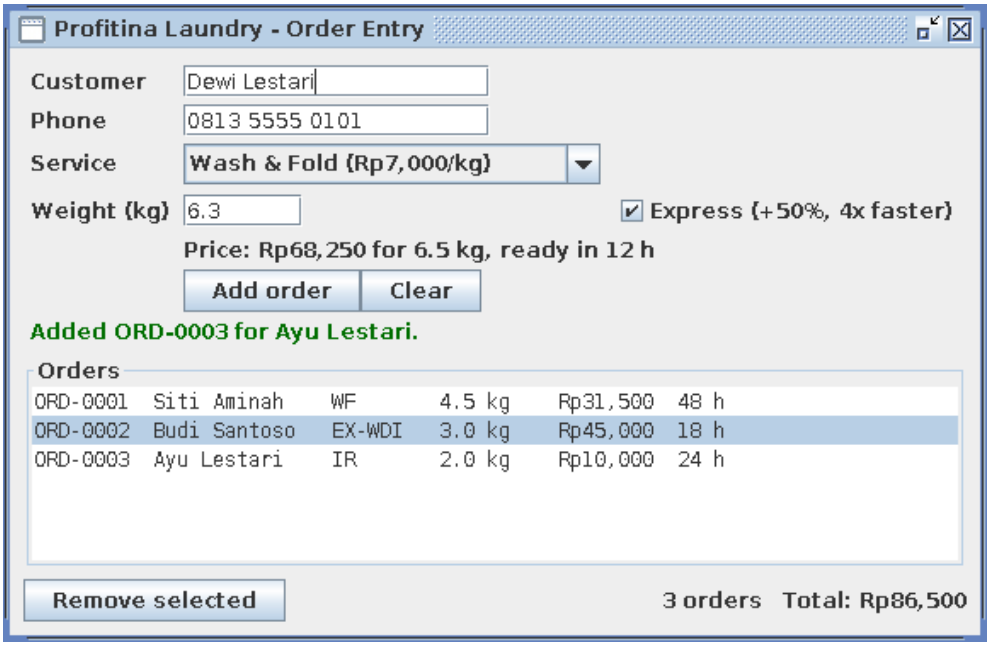
- **Daftar.** Setiap pesanan yang diterima mendapat nomor berikutnya (ORD-0001, ORD-0002, …) dan satu baris di daftar: nomor, pelanggan, kode layanan (diawali `EX-` untuk kilat), berat yang ditagih, harga, dan lama pengerjaan. Formulir lalu dikosongkan untuk pelanggan berikutnya.
- **Hapus dan total.** Tombol *Remove selected*, yang aktif hanya selama ada baris yang dipilih, menghapus pesanan itu. Bagian kaki selalu menampilkan jumlah pesanan dan totalnya.

10.12.1 Jendela, dideskripsikan dengan cermat

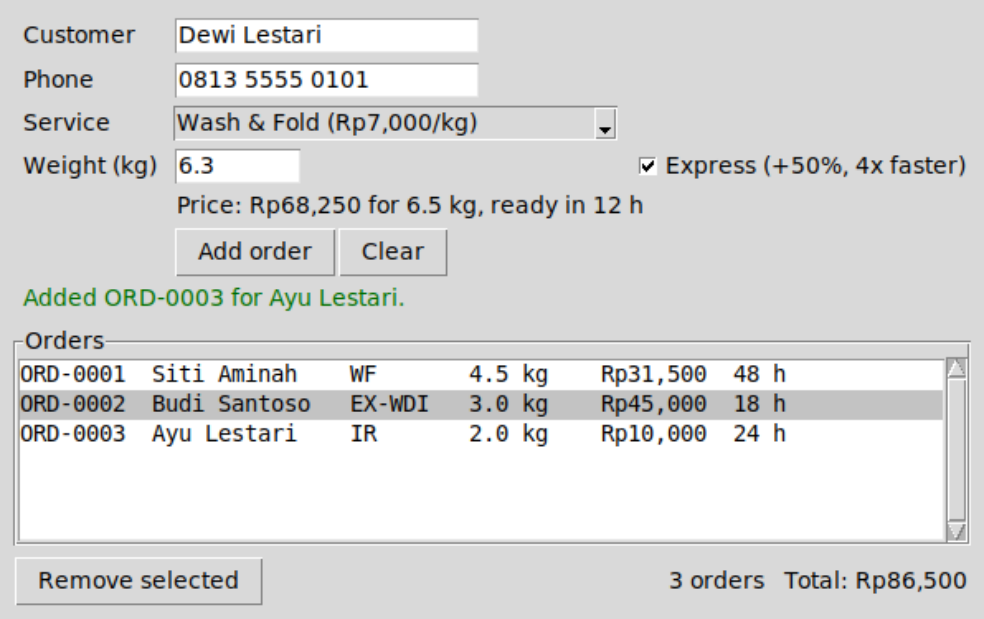
Inilah jendela sebagaimana dilihat staf, dengan tiga pesanan sudah masuk, pesanan keempat sedang diketik, dan baris kedua terpilih:

```
+-- Profitina Laundry - Order Entry ------------------------------[_][O][X]+
|  Customer     [Dewi Lestari        ]                                      |
|  Phone        [0813 5555 0101      ]                                      |
|  Service      [Wash & Fold (Rp7,000/kg)       v]                          |
|  Weight (kg)  [6.3    ]                  [x] Express (+50%, 4x faster)    |
|               Price: Rp68,250 for 6.5 kg, ready in 12 h                   |
|               ( Add order ) ( Clear )                                     |
|  Added ORD-0003 for Ayu Lestari.                   (hijau)                |
| +- Orders ---------------------------------------------------------------+ |
| | ORD-0001  Siti Aminah    WF       4.5 kg    Rp31,500  48 h             | |
| |#ORD-0002  Budi Santoso   EX-WDI   3.0 kg    Rp45,000  18 h#############| |
| | ORD-0003  Ayu Lestari    IR       2.0 kg    Rp10,000  24 h             | |
| | |                                                                       | |
| +-------------------------------------------------------------------------+ |
|  ( Remove selected )                          3 orders   Total: Rp86,500    |
+-----------------------------------------------------------------------------+
```

Dari atas ke bawah: empat field berlabel dalam dua kolom, dengan *Express* di kolom ketiga di samping berat; harga yang langsung diperbarui; dua tombol; baris status, hijau setelah berhasil dan merah setelah galat. Di tengah, sebuah daftar berjudul *Orders* dengan fon berlebar tetap, dengan bilah gulung bila perlu; daftar inilah yang membesar ketika jendela diperbesar, dan baris yang terpilih, digambar dengan `#`, disorot. Di bawah, *Remove selected* di kiri serta jumlah dan total di kanan. **Gambar 10.18** adalah jendela Swing sungguhan dalam keadaan persis ini, dan **Gambar 10.19** jendela Tkinter-nya.



Gambar 10.18 Jendela pencatatan pesanan Swing dengan tiga pesanan, pesanan keempat sedang diketik (harga langsungnya sudah menunjukkan Rp68,250), dan baris kedua terpilih.



Gambar 10.19 Jendela pencatatan pesanan Tkinter dalam keadaan yang sama. (Tangkapan layar memperlihatkan isi jendela tanpa bilah judul yang ditambahkan sistem operasi.)

10.12.2 Harga, secara cermat

Aturan harganya adalah aturan Bab 7, dirumuskan ulang untuk satu baris pesanan. Untuk layanan bertarif r rupiah per kilogram dengan minimum $w_{\min}$, cucian seberat w kilogram ditagih menurut **berat tagihan** $\widehat{w}$: beratnya dinaikkan ke minimum lalu dibulatkan ke atas ke setengah kilogram berikutnya,

$$\widehat{w} = \frac{\lceil 2\max(w, w_{\min}) \rceil}{2} \tag{10.5}$$

dan harga serta lama pengerjaan normalnya adalah

$$P = r \cdot \widehat{w}, \qquad H = h \tag{10.6}$$

dengan h lama pengerjaan normal layanan itu dalam jam. Untuk pesanan kilat, harga dan lama pengerjaannya menjadi

$$P_{\text{ex}} = \lfloor 3P/2 \rfloor, \qquad H_{\text{ex}} = \max(6, \lfloor h/4 \rfloor) \tag{10.7}$$

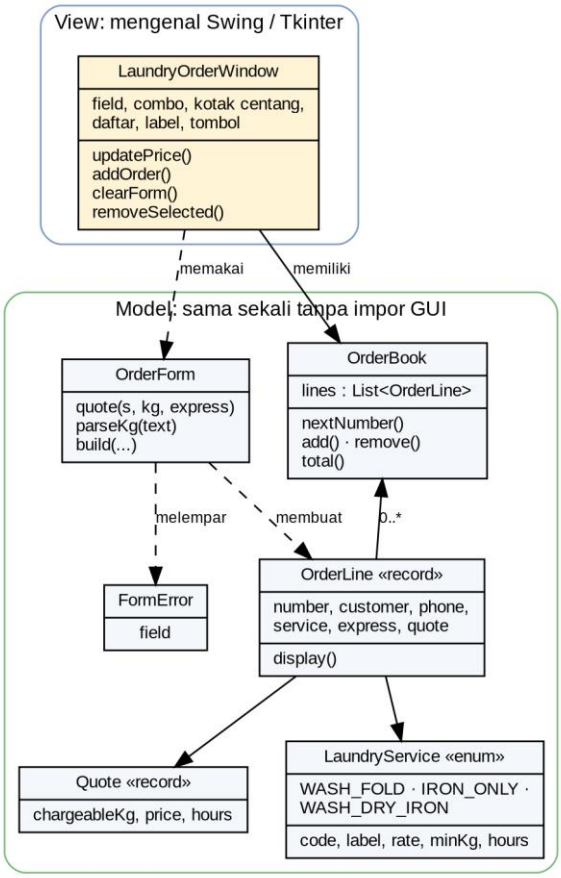
Mari kita periksa tiga pesanan di **Gambar 10.18**. Wash & Fold 4.2 kg milik Siti: menurut Persamaan (10.5), $\max(4.2,3) = 4.2$, $2 \times 4.2 = 8.4$, dibulatkan ke atas menjadi 9, dibagi dua menjadi $\widehat{w} = 4.5$ kg; menurut Persamaan (10.6), $P = 7000 \times 4.5 = 31{,}500$, selesai dalam 48 jam. Wash, Dry & Iron 2.6 kg milik Budi, kilat: $\max(2.6,3) = 3$, jadi $\widehat{w} = 3.0$ dan $P = 10{,}000 \times 3 = 30{,}000$; menurut Persamaan (10.7), $P_{\text{ex}} = \lfloor 90{,}000/2 \rfloor = 45{,}000$ dan $H_{\text{ex}} = \max(6, 72/4) = 18$ jam. Setrika 1.3 kg milik Ayu dinaikkan ke minimum 2 kg: $5000 \times 2 = 10{,}000$, selesai dalam 24 jam. Totalnya $31{,}500 + 45{,}000 + 10{,}000 = 86{,}500$. Dan pesanan Dewi yang sedang diketik, Wash & Fold 6.3 kg, kilat: $\lceil 12.6 \rceil = 13$, $\widehat{w} = 6.5$, $P = 45{,}500$, $P_{\text{ex}} = 68{,}250$, selesai dalam $\max(6,12) = 12$ jam — persis yang ditunjukkan harga langsung.

Di dalam kode, $\lceil 2\max(w, w_{\min}) \rceil$ dihitung sebagai `Math.ceil(2 * Math.max(kg, s.minKg) - 1e-9)`. Nilai mungil 10^{-9} melindungi dari pembulatan biner: berat yang diketik `3.5` tersimpan persis, tetapi berat seperti `4.15` tersimpan sebagai pecahan biner yang sedikit di atas atau di bawah 4.15,

dan menggandakan nilai yang sedikit *di atas* batas setengah kilogram akan membulatkan ke atas satu langkah terlalu jauh. Bab 3 sudah menjelaskan mengapa pecahan desimal jarang tepat dalam biner. Karena setiap tarif adalah kelipatan 2, $r \cdot \hat{w} = r \cdot \lceil \dots \rceil / 2$ dihitung tepat dengan aritmetika bilangan bulat sebagai `s.rate * halfKg / 2`.

10.12.3 Rancangan: model yang tidak tahu apa-apa tentang jendela

Keputusan rancangan yang paling penting tampak di **Gambar 10.20**: program dibagi menjadi sebuah **model**, yang tahu tentang layanan, harga, validasi, dan daftar pesanan tetapi tidak mengimpor apa pun dari Swing atau Tkinter, dan sebuah **view** (tampilan), kelas jendela, yang tahu tentang komponen dan peristiwa serta menanyakan setiap keputusan kepada model.



Gambar 10.20 Rancangan program pencatatan pesanan. View memiliki sebuah `OrderBook` dan memakai `OrderForm` untuk setiap perhitungan dan pemeriksaan; kelas-kelas model tidak pernah menyebut jendela.

- **`LaundryService`** adalah enum yang ketiga konstantanya membawa kode, label, tarif, minimum, dan lama pengerjaannya. `toString()`-nya menghasilkan teks yang ditampilkan combo box, seperti *Wash & Fold (Rp7,000/kg)*.
- **`Quote`** adalah nilai kecil yang tidak dapat diubah: berat tagihan, harga, dan lama pengerjaan.
- **`OrderLine`** adalah rekaman tak-berubah dari satu pesanan yang diterima, dengan `code()` dan `display()`, yang memformat baris yang tampil di daftar.
- **`FormError`** adalah eksepsi yang membawa pesan ramah pengguna *dan* nama field yang harus dibetulkan, sehingga view tahu di mana harus menaruh kursor.
- **`OrderForm`** memegang aturan-aturannya: `quote` mewujudkan Persamaan (10.5), Persamaan (10.6), dan Persamaan (10.7); `parseKg` mengubah teks field berat menjadi bilangan

atau melempar `FormError`; `build` memvalidasi seluruh formulir dan mengembalikan sebuah `OrderLine`.

- **`OrderBook`** menyimpan pesanan yang diterima, membagikan nomor pesanan, dan menghitung total.
- **`LaundryOrderWindow`** membangun komponen, menghubungkan setiap peristiwa dengan penangan yang singkat, dan mengubah jawaban model menjadi teks dan warna.

Mengapa repot-repot? Karena kini setiap belahan dapat dipahami, diubah, dan diuji sendiri-sendiri. Aturan harga dapat diuji tanpa menciptakan satu komponen pun; jendela dapat dirancang ulang — tata letak lain, toolkit lain — tanpa menyentuh satu aturan harga pun. Bab 11 menjadikan pemisahan ini sebuah pola, Model–View–Controller [18], dan memperlihatkan bagaimana model dapat memberi tahu view tentang perubahan dengan sendirinya.

10.12.4 Program Swing

Kode 10.19 adalah seluruh program dalam satu berkas, `LaundryOrderWindow.java`. Kelas-kelas modelnya adalah kelas tingkat atas berakses paket (Bab 7) di berkas yang sama; dalam program yang lebih besar masing-masing akan memiliki berkasnya sendiri.

Kode 10.19 `LaundryOrderWindow.java`: jendela pencatatan pesanan Profitina Laundry dalam Swing, dengan kelas-kelas modelnya dan mode `--selftest`.

```
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.FlowLayout;
import java.awt.Font;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.Insets;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import java.util.Locale;
import javax.swing.BorderFactory;
import javax.swing.DefaultListModel;
import javax.swing.JButton;
import javax.swing.JCheckBox;
import javax.swing.JComboBox;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JList;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTextField;
import javax.swing.ListSelectionModel;
import javax.swing.SwingUtilities;
import javax.swing.event.DocumentEvent;
import javax.swing.event.DocumentListener;

// ---------- Model: Java biasa, tanpa Swing ----------

enum LaundryService {
    WASH_FOLD("WF", "Wash & Fold", 7000, 3.0, 48),
    IRON_ONLY("IR", "Iron Only", 5000, 2.0, 24),
    WASH_DRY_IRON("WDI", "Wash, Dry & Iron", 10000, 3.0, 72);

    final String code;
    final String label;
    final long rate;     // rupiah per kilogram
    final double minKg;  // berat minimum yang ditagih
    final int hours;     // lama pengerjaan normal
```

Kode 10.19 (Lanjutan) *LaundryOrderWindow.java: jendela pencatatan pesanan Profitina Laundry dalam Swing, dengan kelas-kelas modelnya dan mode --selftest.*

```
    LaundryService(String code, String label, long rate, double minKg, int hours) {
        this.code = code;
        this.label = label;
        this.rate = rate;
        this.minKg = minKg;
        this.hours = hours;
    }

    @Override
    public String toString() {  // yang ditampilkan combo box
        return label + " (" + LaundryOrderWindow.rupiah(rate) + "/kg)";
    }
}

record Quote(double chargeableKg, long price, int hours) { }

record OrderLine(String number, String customer, String phone,
                 LaundryService service, boolean express, Quote quote) {
    String code() {
        return (express ? "EX-" : "") + service.code;
    }

    String display() {
        return String.format(Locale.US, "%s  %-14.14s %-6s %5.1f kg %11s %3d h",
                number, customer, code(), quote.chargeableKg(),
                LaundryOrderWindow.rupiah(quote.price()), quote.hours());
    }
}

class FormError extends Exception {  // pesan untuk pengguna, dan field yang harus
dibetulkan
    final String field;

    FormError(String field, String message) {
        super(message);
        this.field = field;
    }
}

final class OrderForm {
    private OrderForm() { }
```

Kode 10.19 (Lanjutan) *LaundryOrderWindow.java: jendela pencatatan pesanan Profitina Laundry dalam Swing, dengan kelas-kelas modelnya dan mode --selftest.*

```
    static Quote quote(LaundryService s, double kg, boolean express) {
        long halfKg = (long) Math.ceil(2 * Math.max(kg, s.minKg) - 1e-9);
        long price = s.rate * halfKg / 2;
        int hours = s.hours;
        if (express) {
            price = 3 * price / 2;
            hours = Math.max(6, hours / 4);
        }
        return new Quote(halfKg / 2.0, price, hours);
    }

    static double parseKg(String text) throws FormError {
        String t = text.strip().replace(',', '.');  // terima 4,2 maupun 4.2
        if (t.isEmpty()) {
            throw new FormError("weight", "Weight is required.");
        }
        if (!t.matches("\\d+(\\.\\d+)?")) {
            throw new FormError("weight", "Weight must be a number, e.g. 4.5");
        }
        double kg = Double.parseDouble(t);
        if (kg <= 0) {
            throw new FormError("weight", "Weight must be more than 0 kg.");
        }
        if (kg > 50) {
            throw new FormError("weight", "Weight must be at most 50 kg.");
        }
        return kg;
    }

    static OrderLine build(String number, String name, String phone,
                           LaundryService s, String kgText, boolean express)
            throws FormError {
        String n = name.strip();
        if (n.isEmpty()) {
            throw new FormError("name", "Customer name is required.");
        }
        if (n.length() > 40) {
            throw new FormError("name", "Customer name must be at most 40 characters.");
        }
        String p = phone.replaceAll("[\\s-]", "");
        if (!p.matches("0\\d{9,12}")) {
            throw new FormError("phone", "Phone must be 10 to 13 digits starting with 0.");
        }
        double kg = parseKg(kgText);
        return new OrderLine(number, n, p, s, express, quote(s, kg, express));
    }
```

Kode 10.19 (Lanjutan) *LaundryOrderWindow.java: jendela pencatatan pesanan Profitina Laundry dalam Swing, dengan kelas-kelas modelnya dan mode --selftest.*

```
}

class OrderBook {
    private final List<OrderLine> lines = new ArrayList<>();
    private int next = 1;

    String nextNumber() {
        return String.format("ORD-%04d", next);
    }

    void add(OrderLine line) {
        lines.add(line);
        next++;
    }

    OrderLine remove(int index) {
        return lines.remove(index);
    }

    List<OrderLine> lines() {
        return Collections.unmodifiableList(lines);
    }

    long total() {
        long t = 0;
        for (OrderLine l : lines) {
            t += l.quote().price();
        }
        return t;
    }
}

// ---------- View: komponen Swing dan listener-nya ----------

public class LaundryOrderWindow extends JPanel {
    private static final Color ERROR = new Color(170, 0, 0);
    private static final Color OK = new Color(0, 110, 0);
```

Kode 10.19 (Lanjutan) *LaundryOrderWindow.java: jendela pencatatan pesanan Profitina Laundry dalam Swing, dengan kelas-kelas modelnya dan mode --selftest.*

```
    private final OrderBook book = new OrderBook();
    final JTextField nameField = new JTextField(16);
    final JTextField phoneField = new JTextField(16);
    final JComboBox<LaundryService> serviceBox = new JComboBox<>(LaundryService.values());
    final JTextField weightField = new JTextField(6);
    final JCheckBox expressBox = new JCheckBox("Express (+50%, 4x faster)");
    final JLabel priceLabel = new JLabel("Price: -");
    final JButton addButton = new JButton("Add order");
    final JButton clearButton = new JButton("Clear");
    final JLabel statusLabel = new JLabel(" ");
    final DefaultListModel<String> listModel = new DefaultListModel<>();
    final JList<String> orderList = new JList<>(listModel);
    final JButton removeButton = new JButton("Remove selected");
    final JLabel totalLabel = new JLabel("0 orders   Total: Rp0");

    public LaundryOrderWindow() {
        super(new BorderLayout(6, 6));
        setBorder(BorderFactory.createEmptyBorder(8, 8, 8, 8));
        add(buildForm(), BorderLayout.PAGE_START);
        orderList.setFont(new Font(Font.MONOSPACED, Font.PLAIN, 12));
        orderList.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
        orderList.setVisibleRowCount(6);
        JScrollPane scroller = new JScrollPane(orderList);
        scroller.setBorder(BorderFactory.createTitledBorder("Orders"));
        add(scroller, BorderLayout.CENTER);
        JPanel footer = new JPanel(new BorderLayout());
        footer.add(removeButton, BorderLayout.LINE_START);
        footer.add(totalLabel, BorderLayout.LINE_END);
        add(footer, BorderLayout.PAGE_END);
        removeButton.setEnabled(false);
        wireEvents();
    }

    private JPanel buildForm() {
        JPanel form = new JPanel(new GridBagLayout());
        GridBagConstraints c = new GridBagConstraints();
        c.insets = new Insets(2, 4, 2, 4);
        c.anchor = GridBagConstraints.LINE_START;
        String[] labels = {"Customer", "Phone", "Service", "Weight (kg)"};
        java.awt.Component[] fields = {nameField, phoneField, serviceBox, weightField};
        for (int row = 0; row < labels.length; row++) {
            c.gridx = 0;
            c.gridy = row;
            form.add(new JLabel(labels[row]), c);
            c.gridx = 1;
            form.add(fields[row], c);
```

Kode 10.19 (Lanjutan) *LaundryOrderWindow.java: jendela pencatatan pesanan Profitina Laundry dalam Swing, dengan kelas-kelas modelnya dan mode --selftest.*

```
        }
        c.gridx = 2;
        c.gridy = 3;
        form.add(expressBox, c);
        c.gridx = 1;
        c.gridy = 4;
        c.gridwidth = 2;
        form.add(priceLabel, c);
        JPanel buttons = new JPanel(new FlowLayout(FlowLayout.LEFT, 0, 0));
        buttons.add(addButton);
        buttons.add(clearButton);
        c.gridy = 5;
        form.add(buttons, c);
        c.gridx = 0;
        c.gridy = 6;
        c.gridwidth = 3;
        form.add(statusLabel, c);
        return form;
    }

    private void wireEvents() {
        weightField.getDocument().addDocumentListener(new DocumentListener() {
            @Override public void insertUpdate(DocumentEvent e) { updatePrice(); }
            @Override public void removeUpdate(DocumentEvent e) { updatePrice(); }
            @Override public void changedUpdate(DocumentEvent e) { updatePrice(); }
        });
        serviceBox.addActionListener(e -> updatePrice());
        expressBox.addActionListener(e -> updatePrice());
        addButton.addActionListener(e -> addOrder());
        weightField.addActionListener(e -> addOrder());  // Enter di field berat
        clearButton.addActionListener(e -> clearForm());
        orderList.addListSelectionListener(
                e -> removeButton.setEnabled(orderList.getSelectedIndex() >= 0));
        removeButton.addActionListener(e -> removeSelected());
    }

    private LaundryService service() {
        return (LaundryService) serviceBox.getSelectedItem();
    }
```

Kode 10.19 (Lanjutan) *LaundryOrderWindow.java: jendela pencatatan pesanan Profitina Laundry dalam Swing, dengan kelas-kelas modelnya dan mode --selftest.*

```
    void updatePrice() {
        try {
            double kg = OrderForm.parseKg(weightField.getText());
            Quote q = OrderForm.quote(service(), kg, expressBox.isSelected());
            priceLabel.setText(String.format(Locale.US, "Price: %s for %.1f kg, ready in %d
h",
                    rupiah(q.price()), q.chargeableKg(), q.hours()));
        } catch (FormError e) {
            priceLabel.setText("Price: -");
        }
    }

    void addOrder() {
        try {
            OrderLine line = OrderForm.build(book.nextNumber(), nameField.getText(),
                    phoneField.getText(), service(), weightField.getText(),
                    expressBox.isSelected());
            book.add(line);
            listModel.addElement(line.display());
            clearForm();
            say(OK, "Added " + line.number() + " for " + line.customer() + ".");
            nameField.requestFocusInWindow();
        } catch (FormError e) {
            say(ERROR, e.getMessage());
            switch (e.field) {
                case "name" -> nameField.requestFocusInWindow();
                case "phone" -> phoneField.requestFocusInWindow();
                default -> weightField.requestFocusInWindow();
            }
        }
        refreshTotal();
    }

    void clearForm() {
        nameField.setText("");
        phoneField.setText("");
        serviceBox.setSelectedIndex(0);
        weightField.setText("");
        expressBox.setSelected(false);
        priceLabel.setText("Price: -");
        say(OK, " ");
    }
```

Kode 10.19 (Lanjutan) *LaundryOrderWindow.java: jendela pencatatan pesanan Profitina Laundry dalam Swing, dengan kelas-kelas modelnya dan mode --selftest.*

```
    void removeSelected() {
        int i = orderList.getSelectedIndex();
        if (i < 0) {
            return;
        }
        OrderLine gone = book.remove(i);
        listModel.remove(i);
        say(OK, "Removed " + gone.number() + ".");
        refreshTotal();
    }

    private void say(Color color, String message) {
        statusLabel.setForeground(color);
        statusLabel.setText(message);
    }

    private void refreshTotal() {
        int n = book.lines().size();
        totalLabel.setText(n + (n == 1 ? " order" : " orders")
                + "    Total: " + rupiah(book.total()));
    }

    static String rupiah(long amount) {
        return String.format(Locale.US, "Rp%,d", amount);
    }

    public static void main(String[] args) throws Exception {
        if (args.length > 0 && args[0].equals("--selftest")) {
            System.setProperty("java.awt.headless", "true");
            SwingUtilities.invokeAndWait(LaundryOrderWindow::selfTest);
        } else {
            SwingUtilities.invokeLater(() -> {
                JFrame.setDefaultLookAndFeelDecorated(true);
                JFrame frame = new JFrame("Profitina Laundry - Order Entry");
                frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                frame.setContentPane(new LaundryOrderWindow());
                frame.pack();
                frame.setLocationRelativeTo(null);  // tengahkan di layar
                frame.setVisible(true);
            });
        }
    }

    // ---------- Uji mandiri: kendalikan komponen sungguhan tanpa layar ----------
```

Kode 10.19 (Lanjutan) *LaundryOrderWindow.java: jendela pencatatan pesanan Profitina Laundry dalam Swing, dengan kelas-kelas modelnya dan mode --selftest.*

```
    void fill(String name, String phone, int service, String kg, boolean express) {
        nameField.setText(name);
        phoneField.setText(phone);
        serviceBox.setSelectedIndex(service);
        weightField.setText(kg);
        if (expressBox.isSelected() != express) {
            expressBox.doClick();
        }
    }

    void report(String step) {
        System.out.println("== " + step);
        System.out.println("   price:  " + priceLabel.getText());
        System.out.println("   status: " + statusLabel.getText());
    }

    static void selfTest() {
        LaundryOrderWindow w = new LaundryOrderWindow();
        w.addButton.doClick();
        w.report("1. Add with an empty form");
        w.fill("Siti Aminah", "0812-3456-7890", 0, "4,2", false);
        w.report("2. Siti, Wash & Fold, 4,2 kg (typed, not yet added)");
        w.addButton.doClick();
        w.report("   ...Add order clicked");
        w.fill("Budi Santoso", "081298765432", 2, "2.6", true);
        w.report("3. Budi, Wash, Dry & Iron, 2.6 kg, express (typed)");
        w.addButton.doClick();
        w.report("   ...Add order clicked");
        w.fill("Ayu Lestari", "12345", 1, "1.3", false);
        w.addButton.doClick();
        w.report("4. Ayu with phone 12345");
        for (String kg : new String[] {"abc", "0", "75"}) {
            w.fill("Ayu Lestari", "0857 1111 2222", 1, kg, false);
            w.addButton.doClick();
            w.report("5. Ayu with weight " + kg);
        }
        w.fill("Ayu Lestari", "0857 1111 2222", 1, "1.3", false);
        w.addButton.doClick();
        w.report("6. Ayu, Iron Only, 1.3 kg");
        System.out.println("   list:");
        for (int i = 0; i < w.listModel.size(); i++) {
            System.out.println("     " + w.listModel.get(i));
        }
        System.out.println("   " + w.totalLabel.getText());
        w.orderList.setSelectedIndex(1);
        System.out.println("   remove enabled after selecting row 2: "
```

Kode 10.19 (Lanjutan) *LaundryOrderWindow.java: jendela pencatatan pesanan Profitina Laundry dalam Swing, dengan kelas-kelas modelnya dan mode --selftest.*

```
                + w.removeButton.isEnabled());
        w.removeButton.doClick();
        w.report("7. Remove the selected order");
        for (int i = 0; i < w.listModel.size(); i++) {
            System.out.println("     " + w.listModel.get(i));
        }
        System.out.println("   " + w.totalLabel.getText());
        System.out.println("   remove enabled: " + w.removeButton.isEnabled());
    }
}
```

Mari kita telusuri view-nya, karena modelnya hanya memakai gagasan dari bab-bab sebelumnya.

Membangun jendela. Konstruktornya memakai `BorderLayout` dengan celah 6 piksel: formulir di `PAGE_START`, daftar yang dapat digulung di `CENTER` (jadi daftar itulah yang membesar), kaki di `PAGE_END` — pohon **Gambar 10.3**. `buildForm` menata field-field berlabel dengan perulangan atas dua larik sejajar, lalu menempatkan kotak centang, harga, panel `FlowLayout` bersarang berisi tombol-tombol, dan label status. `JList` menampilkan sebuah `DefaultListModel<String>`: menambah ke atau menghapus dari model langsung memperbarui daftar di layar.

Menghubungkan peristiwa. Seluruh pengawatan ada di `wireEvents`, satu baris per reaksi, sehingga perilaku jendela dapat dibaca di satu tempat:

- **Dokumen** field berat — model di balik setiap komponen teks Swing — menerima sebuah `DocumentListener`, yang diberi tahu tentang setiap penyisipan dan penghapusan teks, ketukan demi ketukan. Kelas anonimnya harus mengimplementasikan ketiga metode antarmuka itu; masing-masing sekadar memanggil `updatePrice()`. (Action listener pada field hanya akan terpicu oleh Enter, terlambat untuk harga yang langsung diperbarui.)
- Memilih layanan dan mencentang *Express* memicu peristiwa aksi; lambda mengarahkan keduanya ke `updatePrice()`.
- *Add order*, dan Enter di field berat, memanggil `addOrder()`; *Clear* memanggil `clearForm()`.
- Sebuah `ListSelectionListener` pada daftar mengaktifkan *Remove selected* tepat ketika ada baris yang terpilih; tombol itu memanggil `removeSelected()`.

Para penangan. `updatePrice()` mengurai berat dan memformat jawaban `OrderForm.quote`, atau menampilkan tanda pisah bila `parseKg` melempar eksepsi. `addOrder()` membungkus `OrderForm.build` dalam `try`/`catch`: bila berhasil, ia menambahkan baris ke buku dan ke model daftar, mengosongkan formulir, dan melapor dengan warna hijau; bila gagal, ia menampilkan pesannya dengan warna merah dan memakai `field` milik `FormError` untuk menaruh kursor di tempat kekeliruan. Setiap penangan hanya makan waktu beberapa mikrodetik, sehingga EDT tidak pernah tertahan. `removeSelected()` menghapus indeks yang sama dari buku dan dari model daftar, menjaga keduanya tetap selaras.

Memulai. `main` mengikuti Subbab 10.4: `invokeLater`, frame berdekorasi, `EXIT_ON_CLOSE`, panel sebagai content pane, `pack()`, dan `setLocationRelativeTo(null)`, yang meletakkan jendela di tengah layar.

10.12.5 Uji mandiri

Uji mandiri mengendalikan komponen sungguhan melalui sesi yang sudah ditulis skripnya: ia mengetik ke field dengan `setText`, memilih layanan dengan `setSelectedIndex`, mencentang kotak dengan `doClick()`, menekan tombol dengan `doClick()`, dan memilih baris dengan `setSelectedIndex`. Setiap tindakan itu memicu listener yang sama dengan yang akan dipicu pengguna. `java LaundryOrderWindow --selftest` mencetak:

```
== 1. Add with an empty form
   price:  Price: -
   status: Customer name is required.
== 2. Siti, Wash & Fold, 4,2 kg (typed, not yet added)
   price:  Price: Rp31,500 for 4.5 kg, ready in 48 h
   status: Customer name is required.
==    ...Add order clicked
   price:  Price: -
   status: Added ORD-0001 for Siti Aminah.
== 3. Budi, Wash, Dry & Iron, 2.6 kg, express (typed)
   price:  Price: Rp45,000 for 3.0 kg, ready in 18 h
   status: Added ORD-0001 for Siti Aminah.
==    ...Add order clicked
   price:  Price: -
   status: Added ORD-0002 for Budi Santoso.
== 4. Ayu with phone 12345
   price:  Price: Rp10,000 for 2.0 kg, ready in 24 h
   status: Phone must be 10 to 13 digits starting with 0.
== 5. Ayu with weight abc
   price:  Price: -
   status: Weight must be a number, e.g. 4.5
== 5. Ayu with weight 0
   price:  Price: -
   status: Weight must be more than 0 kg.
== 5. Ayu with weight 75
   price:  Price: -
   status: Weight must be at most 50 kg.
== 6. Ayu, Iron Only, 1.3 kg
   price:  Price: -
   status: Added ORD-0003 for Ayu Lestari.
   list:
     ORD-0001  Siti Aminah    WF       4.5 kg    Rp31,500  48 h
     ORD-0002  Budi Santoso   EX-WDI   3.0 kg    Rp45,000  18 h
     ORD-0003  Ayu Lestari    IR       2.0 kg    Rp10,000  24 h
   3 orders   Total: Rp86,500
   remove enabled after selecting row 2: true
== 7. Remove the selected order
   price:  Price: -
   status: Removed ORD-0002.
     ORD-0001  Siti Aminah    WF       4.5 kg    Rp31,500  48 h
     ORD-0003  Ayu Lestari    IR       2.0 kg    Rp10,000  24 h
   2 orders   Total: Rp41,500
   remove enabled: false
```

Ikuti keluarannya terhadap kebutuhan. (1) Formulir kosong gagal pada aturan pertama, nama. (2) Begitu data Siti diketik — sebelum *Add order* — `DocumentListener` sudah mengubah `4,2` menjadi Rp31,500 untuk 4.5 kg; statusnya masih menampilkan galat lama. *Add order* menerima pesanan, mengosongkan formulir, dan melaporkan keberhasilan. (3) Pesanan kilat Budi menampilkan Rp45,000 dan 18 jam selagi diketik. (4) Telepon `12345` ditolak, sementara harga langsung untuk Ayu sudah tampil. (5) Tiga berat yang keliru mendapat tiga pesan yang berbeda. (6) Pesanan yang sudah dibetulkan menjadi ORD-0003, dan totalnya Rp86,500. (7) Memilih baris 2 mengaktifkan *Remove selected*; menghapus ORD-0002 menyisakan $86{,}500 - 45{,}000 = 41{,}500$ dan menonaktifkan tombol itu lagi.

10.12.6 Program Tkinter

Kode 10.20 adalah program yang sama dalam Tkinter. Modelnya berancangan sama baris demi baris, dengan idiom Python: sebuah `Enum` yang anggotanya membawa tupel nilai, `dataclass` beku untuk `Quote` dan `OrderLine`, serta fungsi biasa `quote`, `parse_kg`, dan `build_order` menggantikan metode statis `OrderForm`.

Kode 10.20 `laundry_order_window.py`: jendela pencatatan pesanan yang sama dalam Tkinter, dengan model yang sama dan uji mandiri yang sama.

```
"""laundry_order_window.py: jendela pencatatan pesanan Profitina Laundry dalam Tkinter."""
import math
import re
import sys
import tkinter as tk
from dataclasses import dataclass
from enum import Enum
from tkinter import ttk

# ---------- Model: Python biasa, tanpa Tkinter ----------


def rupiah(amount):
    return f"Rp{amount:,}"


class LaundryService(Enum):
    WASH_FOLD = ("WF", "Wash & Fold", 7000, 3.0, 48)
    IRON_ONLY = ("IR", "Iron Only", 5000, 2.0, 24)
    WASH_DRY_IRON = ("WDI", "Wash, Dry & Iron", 10000, 3.0, 72)

    def __init__(self, code, label, rate, min_kg, hours):
        self.code = code
        self.label = label
        self.rate = rate      # rupiah per kilogram
        self.min_kg = min_kg  # berat minimum yang ditagih
        self.hours = hours    # lama pengerjaan normal

    def __str__(self):  # yang ditampilkan combo box
        return f"{self.label} ({rupiah(self.rate)}/kg)"


@dataclass(frozen=True)
class Quote:
    chargeable_kg: float
    price: int
    hours: int
```

Kode 10.20 (Lanjutan) *laundry_order_window.py: jendela pencatatan pesanan yang sama dalam Tkinter, dengan model yang sama dan uji mandiri yang sama.*

```
@dataclass(frozen=True)
class OrderLine:
    number: str
    customer: str
    phone: str
    service: LaundryService
    express: bool
    quote: Quote

    def code(self):
        return ("EX-" if self.express else "") + self.service.code

    def display(self):
        q = self.quote
        return (f"{self.number}  {self.customer:<14.14} {self.code():<6} "
                f"{q.chargeable_kg:5.1f} kg {rupiah(q.price):>11} {q.hours:3d} h")


class FormError(ValueError):  # pesan untuk pengguna, dan field yang harus dibetulkan
    def __init__(self, field, message):
        super().__init__(message)
        self.field = field


def quote(s, kg, express):
    half_kg = math.ceil(2 * max(kg, s.min_kg) - 1e-9)
    price = s.rate * half_kg // 2
    hours = s.hours
    if express:
        price = 3 * price // 2
        hours = max(6, hours // 4)
    return Quote(half_kg / 2, price, hours)


def parse_kg(text):
    t = text.strip().replace(",", ".")  # accept 4,2 as well as 4.2
    if not t:
        raise FormError("weight", "Weight is required.")
    if not re.fullmatch(r"\d+(\.\d+)?", t):
        raise FormError("weight", "Weight must be a number, e.g. 4.5")
    kg = float(t)
    if kg <= 0:
        raise FormError("weight", "Weight must be more than 0 kg.")
    if kg > 50:
        raise FormError("weight", "Weight must be at most 50 kg.")
    return kg
```

Kode 10.20 (Lanjutan) *laundry_order_window.py: jendela pencatatan pesanan yang sama dalam Tkinter, dengan model yang sama dan uji mandiri yang sama.*

```
def build_order(number, name, phone, s, kg_text, express):
    n = name.strip()
    if not n:
        raise FormError("name", "Customer name is required.")
    if len(n) > 40:
        raise FormError("name", "Customer name must be at most 40 characters.")
    p = re.sub(r"[\s-]", "", phone)
    if not re.fullmatch(r"0\d{9,12}", p):
        raise FormError("phone", "Phone must be 10 to 13 digits starting with 0.")
    kg = parse_kg(kg_text)
    return OrderLine(number, n, p, s, express, quote(s, kg, express))


class OrderBook:
    def __init__(self):
        self._lines = []
        self._next = 1

    def next_number(self):
        return f"ORD-{self._next:04d}"

    def add(self, line):
        self._lines.append(line)
        self._next += 1

    def remove(self, index):
        return self._lines.pop(index)

    def lines(self):
        return tuple(self._lines)

    def total(self):
        return sum(line.quote.price for line in self._lines)


# ---------- View: widget Tkinter dan callback-nya ----------

ERROR = "#aa0000"
OK = "#006e00"
SERVICES = list(LaundryService)
```

Kode 10.20 (Lanjutan) *laundry_order_window.py: jendela pencatatan pesanan yang sama dalam Tkinter, dengan model yang sama dan uji mandiri yang sama.*

```
class LaundryOrderWindow(tk.Frame):
    def __init__(self, master):
        super().__init__(master, padx=8, pady=8)
        self.book = OrderBook()
        self.name_var = tk.StringVar()
        self.phone_var = tk.StringVar()
        self.service_var = tk.StringVar(value=str(SERVICES[0]))
        self.weight_var = tk.StringVar()
        self.express_var = tk.BooleanVar(value=False)
        self.build_form()
        box = tk.LabelFrame(self, text="Orders")
        box.pack(fill=tk.BOTH, expand=True, pady=6)
        self.order_list = tk.Listbox(box, height=6, width=58, font="TkFixedFont",
                                     selectmode=tk.SINGLE, exportselection=False)
        bar = tk.Scrollbar(box, command=self.order_list.yview)
        self.order_list.config(yscrollcommand=bar.set)
        bar.pack(side=tk.RIGHT, fill=tk.Y)
        self.order_list.pack(side=tk.LEFT, fill=tk.BOTH, expand=True)
        footer = tk.Frame(self)
        footer.pack(fill=tk.X)
        self.remove_button = tk.Button(footer, text="Remove selected",
                                       command=self.remove_selected, state=tk.DISABLED)
        self.remove_button.pack(side=tk.LEFT)
        self.total_label = tk.Label(footer, text="0 orders   Total: Rp0")
        self.total_label.pack(side=tk.RIGHT)
        self.wire_events()

    def build_form(self):
        form = tk.Frame(self)
        form.pack(fill=tk.X)
        self.name_entry = tk.Entry(form, width=20, textvariable=self.name_var)
        self.phone_entry = tk.Entry(form, width=20, textvariable=self.phone_var)
        self.service_box = ttk.Combobox(form, width=28, state="readonly",
                                        textvariable=self.service_var,
                                        values=[str(s) for s in SERVICES])
        self.weight_entry = tk.Entry(form, width=8, textvariable=self.weight_var)
        labels = ["Customer", "Phone", "Service", "Weight (kg)"]
        fields = [self.name_entry, self.phone_entry, self.service_box, self.weight_entry]
        for row, (text, field) in enumerate(zip(labels, fields)):
            tk.Label(form, text=text).grid(row=row, column=0, sticky="w", padx=4, pady=2)
            field.grid(row=row, column=1, sticky="w", padx=4, pady=2)
        self.express_check = tk.Checkbutton(form, text="Express (+50%, 4x faster)",
                                            variable=self.express_var,
                                            command=self.update_price)
        self.express_check.grid(row=3, column=2, sticky="w")
        self.price_label = tk.Label(form, text="Price: -")
```

Kode 10.20 (Lanjutan) *laundry_order_window.py: jendela pencatatan pesanan yang sama dalam Tkinter, dengan model yang sama dan uji mandiri yang sama.*

```
        self.price_label.grid(row=4, column=1, columnspan=2, sticky="w", padx=4)
        buttons = tk.Frame(form)
        buttons.grid(row=5, column=1, columnspan=2, sticky="w", padx=4, pady=2)
        self.add_button = tk.Button(buttons, text="Add order", command=self.add_order)
        self.clear_button = tk.Button(buttons, text="Clear", command=self.clear_form)
        self.add_button.pack(side=tk.LEFT)
        self.clear_button.pack(side=tk.LEFT)
        self.status_label = tk.Label(form, text=" ")
        self.status_label.grid(row=6, column=0, columnspan=3, sticky="w", padx=4)

    def wire_events(self):
        # Trace pada variabel terpicu setiap kali berubah, seperti DocumentListener.
        self.weight_var.trace_add("write", lambda *args: self.update_price())
        self.service_var.trace_add("write", lambda *args: self.update_price())
        self.weight_entry.bind("<Return>", lambda event: self.add_order())
        self.order_list.bind("<<ListboxSelect>>", self.on_select)

    def service(self):
        return SERVICES[[str(s) for s in SERVICES].index(self.service_var.get())]

    def update_price(self):
        try:
            kg = parse_kg(self.weight_var.get())
        except FormError:
            self.price_label.config(text="Price: -")
            return
        q = quote(self.service(), kg, self.express_var.get())
        self.price_label.config(
            text=f"Price: {rupiah(q.price)} for {q.chargeable_kg:.1f} kg, ready in
{q.hours} h")

    def add_order(self):
        try:
            line = build_order(self.book.next_number(), self.name_var.get(),
                               self.phone_var.get(), self.service(),
                               self.weight_var.get(), self.express_var.get())
        except FormError as e:
            self.say(ERROR, str(e))
            widget = {"name": self.name_entry, "phone": self.phone_entry}.get(
                e.field, self.weight_entry)
            widget.focus_set()
        else:
            self.book.add(line)
            self.order_list.insert(tk.END, line.display())
            self.clear_form()
            self.say(OK, f"Added {line.number} for {line.customer}.")
            self.name_entry.focus_set()
```

Kode 10.20 (Lanjutan) *laundry_order_window.py: jendela pencatatan pesanan yang sama dalam Tkinter, dengan model yang sama dan uji mandiri yang sama.*

```
        self.refresh_total()

    def clear_form(self):
        self.name_var.set("")
        self.phone_var.set("")
        self.service_var.set(str(SERVICES[0]))
        self.weight_var.set("")
        self.express_var.set(False)
        self.price_label.config(text="Price: -")
        self.say(OK, " ")

    def on_select(self, event):
        chosen = self.order_list.curselection()
        self.remove_button.config(state=tk.NORMAL if chosen else tk.DISABLED)

    def remove_selected(self):
        chosen = self.order_list.curselection()
        if not chosen:
            return
        i = chosen[0]
        gone = self.book.remove(i)
        self.order_list.delete(i)
        self.say(OK, f"Removed {gone.number}.")
        self.on_select(None)
        self.refresh_total()

    def say(self, color, message):
        self.status_label.config(fg=color, text=message)

    def refresh_total(self):
        n = len(self.book.lines())
        self.total_label.config(
            text=f"{n} {'order' if n == 1 else 'orders'}   Total:
{rupiah(self.book.total())}")


# ---------- Uji mandiri: kendalikan widget sungguhan tanpa menampilkannya ----------

def fill(w, name, phone, service, kg, express):
    w.name_var.set(name)
    w.phone_var.set(phone)
    w.service_box.current(service)
    w.weight_var.set(kg)
    if w.express_var.get() != express:
        w.express_check.invoke()
```

Kode 10.20 (Lanjutan) *laundry_order_window.py: jendela pencatatan pesanan yang sama dalam Tkinter, dengan model yang sama dan uji mandiri yang sama.*

```
def report(w, step):
    print("==", step)
    print("   price: ", w.price_label.cget("text"))
    print("   status:", w.status_label.cget("text"))


def self_test():
    root = tk.Tk()
    root.withdraw()
    w = LaundryOrderWindow(root)
    w.pack(fill=tk.BOTH, expand=True)
    w.add_button.invoke()
    report(w, "1. Add with an empty form")
    fill(w, "Siti Aminah", "0812-3456-7890", 0, "4,2", False)
    report(w, "2. Siti, Wash & Fold, 4,2 kg (typed, not yet added)")
    w.add_button.invoke()
    report(w, "   ...Add order clicked")
    fill(w, "Budi Santoso", "081298765432", 2, "2.6", True)
    report(w, "3. Budi, Wash, Dry & Iron, 2.6 kg, express (typed)")
    w.add_button.invoke()
    report(w, "   ...Add order clicked")
    fill(w, "Ayu Lestari", "12345", 1, "1.3", False)
    w.add_button.invoke()
    report(w, "4. Ayu with phone 12345")
    for kg in ("abc", "0", "75"):
        fill(w, "Ayu Lestari", "0857 1111 2222", 1, kg, False)
        w.add_button.invoke()
        report(w, f"5. Ayu with weight {kg}")
    fill(w, "Ayu Lestari", "0857 1111 2222", 1, "1.3", False)
    w.add_button.invoke()
    report(w, "6. Ayu, Iron Only, 1.3 kg")
    print("   list:")
    for row in w.order_list.get(0, tk.END):
        print("    ", row)
    print("  ", w.total_label.cget("text"))
    w.order_list.selection_set(1)
    w.on_select(None)  # yang dipanggil Tk setelah klik pada baris 2
    print("   remove enabled after selecting row 2:",
          str(w.remove_button.cget("state") == tk.NORMAL).lower())
    w.remove_button.invoke()
    report(w, "7. Remove the selected order")
    for row in w.order_list.get(0, tk.END):
        print("    ", row)
    print("  ", w.total_label.cget("text"))
    print("   remove enabled:", str(w.remove_button.cget("state") == tk.NORMAL).lower())
    root.destroy()
```

Kode 10.20 (Lanjutan) *laundry_order_window.py: jendela pencatatan pesanan yang sama dalam Tkinter, dengan model yang sama dan uji mandiri yang sama.*

```
def main():
    if "--selftest" in sys.argv:
        self_test()
        return
    root = tk.Tk()
    root.title("Profitina Laundry - Order Entry")
    LaundryOrderWindow(root).pack(fill=tk.BOTH, expand=True)
    root.mainloop()


if __name__ == "__main__":
    main()
```

`xvfb-run -a python3 laundry_order_window.py --selftest` mencetak baris-baris yang *persis* sama dengan versi Java, karakter demi karakter; hal ini sudah diperiksa dengan `diff`:

```
== 1. Add with an empty form
   price:  Price: -
   status: Customer name is required.
== 2. Siti, Wash & Fold, 4,2 kg (typed, not yet added)
   price:  Price: Rp31,500 for 4.5 kg, ready in 48 h
   status: Customer name is required.
==    ...Add order clicked
   price:  Price: -
   status: Added ORD-0001 for Siti Aminah.
== 3. Budi, Wash, Dry & Iron, 2.6 kg, express (typed)
   price:  Price: Rp45,000 for 3.0 kg, ready in 18 h
   status: Added ORD-0001 for Siti Aminah.
==    ...Add order clicked
   price:  Price: -
   status: Added ORD-0002 for Budi Santoso.
== 4. Ayu with phone 12345
   price:  Price: Rp10,000 for 2.0 kg, ready in 24 h
   status: Phone must be 10 to 13 digits starting with 0.
== 5. Ayu with weight abc
   price:  Price: -
   status: Weight must be a number, e.g. 4.5
== 5. Ayu with weight 0
   price:  Price: -
   status: Weight must be more than 0 kg.
== 5. Ayu with weight 75
   price:  Price: -
   status: Weight must be at most 50 kg.
== 6. Ayu, Iron Only, 1.3 kg
   price:  Price: -
   status: Added ORD-0003 for Ayu Lestari.
   list:
     ORD-0001  Siti Aminah    WF       4.5 kg    Rp31,500  48 h
     ORD-0002  Budi Santoso   EX-WDI   3.0 kg    Rp45,000  18 h
     ORD-0003  Ayu Lestari    IR       2.0 kg    Rp10,000  24 h
   3 orders   Total: Rp86,500
   remove enabled after selecting row 2: true
== 7. Remove the selected order
   price:  Price: -
   status: Removed ORD-0002.
     ORD-0001  Siti Aminah    WF       4.5 kg    Rp31,500  48 h
     ORD-0003  Ayu Lestari    IR       2.0 kg    Rp10,000  24 h
   2 orders   Total: Rp41,500
   remove enabled: false
```

Beberapa hal khas Tkinter layak disinggung.

- **Variabel kendali menggerakkan harga langsung.** Setiap entry dihubungkan ke sebuah `StringVar` dengan `textvariable=`, dan combo box ke `service_var`. `weight_var.trace_add("write", …)` memanggil sebuah fungsi setiap kali variabel itu ditulisi — entah oleh pengguna yang mengetik, entah oleh program yang memanggil `set` — dan inilah padanan `DocumentListener` di Tkinter. Callback *trace* menerima tiga argumen yang tidak kita perlukan, maka `lambda *args: self.update_price()`.
- **Combo box menyimpan teks.** Sebuah `ttk.Combobox` menampung string, jadi `service()` mencari anggota enum yang `str()`-nya sama dengan teks yang dipilih. `state="readonly"` mencegah pengguna mengetik layanan yang tidak ada.
- **List box.** Sebuah `Listbox` menyimpan sendiri baris-baris teksnya (tidak ada model daftar terpisah); `insert(tk.END, row)` menambah di akhir, `delete(i)` menghapus, dan `curselection()` mengembalikan tupel indeks yang terpilih. `exportselection=False` mempertahankan pilihan ketika pengguna mengeklik field lain. Bilah gulungnya dihubungkan ke dua arah: `command` milik bilah menggulung daftar, dan `yscrollcommand` milik daftar menggerakkan bilah.
- **Geometri.** Formulir memakai `grid`, daftar dan kaki memakai `pack`, dan keduanya tidak pernah berbagi induk, sebagaimana dituntut Subbab 10.8.5.
- **Galat memakai `else`.** Di `add_order`, bentuk `try`/`except`/`else` menjaga jalur keberhasilan di luar blok `try`, sehingga eksepsi tak terduga di kode keberhasilan tidak dikira galat validasi.

10.13 Profitina dalam Praktik

Produk-produk Profitina adalah aplikasi web, bukan jendela desktop, tetapi peramban menjalankan persis model bab ini. Sebuah halaman web memiliki satu thread GUI; kode JavaScript mendaftarkan listener dengan `addEventListener`, gelung peristiwa peramban menyalurkan klik dan penekanan tombol kepadanya, dan listener yang berhitung terlalu lama membekukan halaman, persis seperti `actionPerformed` yang lambat membekukan jendela Swing. Gagasannya berpindah satu lawan satu.

Pengiriman optimistis adalah aturan thread dalam praktik. Ketika pengguna mengirim tulisan di komunitas waktu-nyata Profitina, kotak tulis langsung kosong dan tulisannya langsung tampil; permintaan jaringan berjalan di latar, dan bila pengiriman gagal, teksnya dikembalikan sehingga tidak ada yang hilang. Itulah bentuk **Kode 10.17**: penangan klik memperbarui tampilan dan kembali dalam beberapa milidetik, pekerjaan lambat (di sini, perjalanan pulang-pergi jaringan) berjalan di tempat lain, dan hasilnya kembali ke thread GUI untuk mengonfirmasi atau membatalkan — seperti `done()` pada `SwingWorker`.

Umpan langsung adalah antrean peristiwa. Umpan komunitas diperbarui dengan **Server-Sent Events** (SSE): server mendorong pesan kecil setiap kali sesuatu terjadi, dan gelung peristiwa halaman mengantarkan setiap pesan ke listener yang menyisipkan tulisan baru. Halaman tidak pernah bertanya "adakah yang baru?" di dalam perulangan; ia mendaftarkan listener lalu menunggu, dan itu persis **Pseudokode 10.1** dengan server sebagai satu sumber peristiwa lagi.

Dasbor adalah view di atas model. Dasbor Profitina mengumpulkan notifikasi, kanal yang diikuti, peringkat, dan aktivitas dengan satu permintaan, dan menampilkan setiap butir sebagai sesuatu yang dapat diklik menuju bagiannya. Kode halaman menampilkan apa yang dikatakan model di server, dan setiap klik adalah penangan kecil; penilaian dan aturannya tinggal di server, tempat keduanya dapat diuji tanpa peramban sama sekali — pemisahan yang sama dengan **Gambar 10.20**. Dan sesuai prinsip rumah bahwa aplikasi memberi saran dan penggunalah yang

memutuskan, jendela seperti formulir laundry menampilkan harga dan menjelaskan setiap penolakan, tetapi orang di meja layananlah yang menekan *Add order*.

10.14 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya

1. Melakukan pekerjaan lambat di listener. Membaca berkas, memanggil server, atau tidur di dalam `actionPerformed` atau callback Tk membekukan seluruh jendela. *Hindari:* jaga penangan tetap singkat; pindahkan pekerjaan lambat ke `SwingWorker` atau ke thread dengan antrean dan `after` (Subbab 10.11).

2. Menyentuh komponen dari thread lain. Menyetel teks label dari thread pekerja mungkin berhasil seratus kali lalu merusak tampilan pada kali keseratus satu. Tkinter sering langsung menolak dengan `RuntimeError: main thread is not in main loop`. *Hindari:* di Swing, ubah komponen hanya di EDT (`done()`, `process()`, `invokeLater`); di Tkinter, hanya di thread `mainloop`, melalui antrean yang diperiksa dengan `after`.

3. Lupa `setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE)`. Menutup jendela hanya menyembunyikannya; program terus berjalan tanpa jendela, dan terminal tampak macet. *Hindari:* tulislah baris itu untuk setiap jendela utama (Subbab 10.4).

4. Lupa `setVisible(true)`, atau memanggilnya sebelum menambahkan komponen. Tidak ada yang muncul, atau komponen yang ditambahkan sesudahnya baru tampak setelah ukuran jendela diubah. *Hindari:* bangun semuanya, lalu `pack()`, lalu `setVisible(true)`, dalam urutan itu.

5. Menggambar di luar `paintComponent`. Menggambar dengan `getGraphics()` di dalam listener tampak berhasil, dan gambarnya lenyap begitu jendela dilukis ulang (Subbab 10.10.6). *Hindari:* simpan apa yang harus digambar di field, gambarlah di `paintComponent`, dan panggil `repaint()` setelah mengubah field.

6. Lupa `super.paintComponent(g)`. Gambar lama tertinggal di layar di bawah gambar baru, atau muncul sampah. *Hindari:* jadikan itu baris pertama setiap `paintComponent`.

7. Memanggil `paintComponent` sendiri. Anda tidak memiliki objek `Graphics` untuk layar, dan melukis adalah tugas gelung peristiwa. *Hindari:* panggil `repaint()`, yang meminta pelukisan.

8. Mengira listener berjalan di tempat ia didaftarkan. `button.addActionListener(e -> total += 10)` tidak menambahkan 10; ia mengatur agar 10 ditambahkan *setiap kali tombol diklik*. *Hindari:* bacalah `add…Listener` dan `bind` sebagai "ingat ini untuk nanti".

9. Jebakan late binding dalam perulangan Python. `command=lambda: self.bump(n)` di dalam perulangan membuat setiap tombol memakai nilai terakhir `n` (Subbab 10.10.5). *Hindari:* `lambda n=n: self.bump(n)` atau `functools.partial(self.bump, n)`.

10. Memanggil callback alih-alih memberikannya. Di Tkinter, `Button(..., command=self.add_order())` *memanggil* `add_order` sekali saat jendela sedang dibangun dan memberikan hasilnya, `None`, sebagai perintah, sehingga tombolnya tidak berbuat apa-apa. *Hindari:* berikan fungsinya tanpa tanda kurung, `command=self.add_order`, atau bungkus pemanggilan berargumen di dalam lambda.

11. Mencampur `pack` dan `grid` pada induk yang sama. Tk 8.6 melempar `TclError: cannot use geometry manager grid inside … which already has slaves managed by pack`, dan versi yang lebih lama sekadar macet. *Hindari:* pakai satu geometry manager per induk; sarangkan frame bila Anda memerlukan keduanya.

12. Posisi mutlak. `setLayout(null)` dengan `setBounds`, atau `place(x=…, y=…)`, menghasilkan jendela yang rusak dengan fon, bahasa, atau ukuran jendela yang lain. *Hindari:* pakai layout manager, dengan menyarangkan panel seperlunya (Subbab 10.8).

13. Menaruh aturan di dalam listener. Rumus harga atau aturan validasi yang ditulis di dalam `actionPerformed` hanya dapat diuji dengan mengeklik. *Hindari:* simpan aturan di kelas model yang tidak mengimpor apa pun dari GUI, dan biarkan listener memanggilnya (Subbab 10.12.3).

10.15 Ringkasan Bab

- Program GUI bersifat **berbasis peristiwa**: ia membangun jendela, mendaftarkan **penangan peristiwa** (*listener* di Java, *callback* di Python), lalu menyerahkan kendali ke **gelung peristiwa** toolkit, yang menunggu peristiwa dan memanggil para penangan — **pembalikan kendali**.
- GUI Java bertumpu pada **AWT** (jendela asli, peristiwa, `Graphics`, layout manager) dan **Swing** (komponen ringan `J…` yang menggambar dirinya sendiri). Toolkit standar Python adalah **Tkinter**, lapisan Python di atas **Tcl/Tk**.
- Jendela adalah sebuah **pohon** komponen. `JFrame` menampung **content pane**; panel (`JPanel`, `tk.Frame`) adalah kontainer, dan setiap bagian jendela sungguhan paling baik ditulis sebagai kelas yang memperluas salah satunya.
- **Gelung peristiwa** (**Pseudokode 10.1**) menunggu peristiwa, mencari sasarannya, memanggil listener yang terdaftar untuknya, dan menggambar ulang saat senggang. Selama sebuah listener berjalan, tidak ada hal lain yang terjadi.
- Untuk mendengarkan di Java: **deklarasikan** (`implements ActionListener`), **daftarkan** (`addActionListener`), **definisikan** (`actionPerformed`). Di Tkinter: `command=` atau `bind("<Sequence>", callback)`. Objek peristiwa menyatakan apa yang terjadi (`getSource()`, `getPoint()`; `event.x`, `event.widget`).
- **Layout manager** (`FlowLayout`, `BorderLayout`, `BoxLayout`, `GridBagLayout`) dan **geometry manager** (`pack`, `grid`) menempatkan komponen menurut aturan, sehingga jendela tetap utuh dengan fon dan ukuran lain; bangun jendela yang rumit dengan menyarangkan.
- **Melukis**: *override* `paintComponent`, panggil `super.paintComponent(g)` lebih dulu, gambar dari field, dan panggil `repaint()` untuk meminta pembaruan (mode langsung). `Canvas` Tk menyimpan item dan menggambarnya ulang sendiri (mode tertahan). Warna adalah tripel RGB, sebanyak 256^3 (Persamaan (10.1)).
- **Kelas dalam** menyimpan referensi tersembunyi ke objek luarnya; **kelas anonim** dan **adaptor** mengimplementasikan listener di tempat; **lambda** dan **referensi metode** adalah bentuk ringkas untuk antarmuka bermetode satu dan hanya menangkap variabel lokal yang **effectively final**, dengan cara disalin. **Closure** Python menangkap variabel melalui sel dan mengikat **terlambat**; perbaiki perulangan dengan `lambda n=n:` atau `partial`.
- **Aturan thread**: sentuh GUI hanya di thread-nya sendiri (EDT milik Swing, thread `mainloop` milik Tk) dan jangan pernah menahan thread itu. Pakai `invokeLater`/`invokeAndWait` dan `SwingWorker` di Swing; thread pekerja, `queue.Queue`, dan `after` di Tkinter.
- **Mode uji mandiri** (`--selftest`) membangun dan mengendalikan komponen sungguhan tanpa menampilkannya: mode *headless* di Swing, akar yang disembunyikan di bawah Xvfb di Tkinter.
- Jendela pencatatan pesanan laundry memisahkan **model** yang bebas toolkit (`LaundryService`, `Quote`, `OrderLine`, `FormError`, `OrderForm`, `OrderBook`) dari **view**; versi Swing dan Tkinter-nya menghasilkan keluaran uji mandiri yang identik.

10.16 Soal Latihan

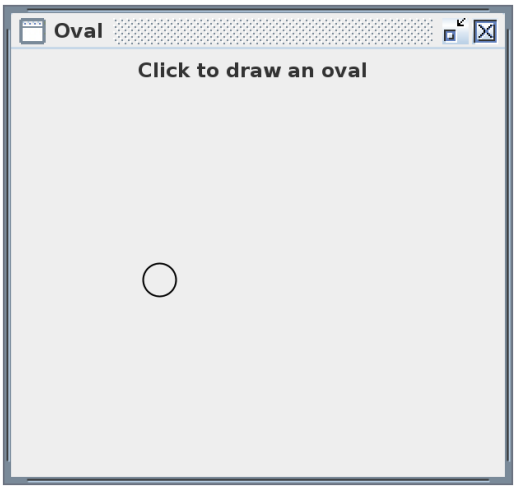
Soal Latihan 10.1 (menelusuri gelung peristiwa). Di **Kode 10.3**, tambahkan listener kedua pada tombol, didaftarkan *setelah* yang pertama, yang mencetak `button says: clicked`. Ramalkan keluaran lengkapnya sebelum menjalankan program. Lalu pindahkan baris `queue.add(new Event("totalLabel", …))` sehingga peristiwa gambar ulang diantrekan *sebelum* `total` dinaikkan. Apa yang ditampilkan "layar" sekarang, dan aturan umum apa tentang listener dan permintaan gambar ulang yang diperlihatkan hal ini?

Soal Latihan 10.2 (BorderLayout dengan tangan). Sebuah panel berukuran 400 × 200 piksel memakai `BorderLayout`. Komponen `PAGE_START`-nya menyukai tinggi 30, komponen `PAGE_END` tinggi 20, komponen `LINE_START` lebar 80, dan komponen `LINE_END` lebar 60; komponen `CENTER` menyukai 10 × 10. Hitunglah batas (x, y, w, h) kelima komponen. Periksa jawaban Anda dengan metode `report` dari **Kode 10.9**.

Soal Latihan 10.3 (ukur, jangan menduga). Kelima tombol **Kode 10.9** memiliki lebar yang disukai 94, 94, 94, 187, dan 42, dan `FlowLayout` menyisakan celah 5 piksel di antara komponen dan di kedua ujung baris. (a) Hitunglah lebar panel terkecil yang, menurut deskripsi itu, memuat kelimanya dalam satu baris. (b) Temukan lebar terkecil yang sebenarnya dengan percobaan, mengubah lebar yang diberikan kepada `report`. (c) Jelaskan perbedaannya.

Soal Latihan 10.4 (log baris-baris). Ubahlah **Kode 10.5** dan **Kode 10.6** sehingga setiap Enter menambahkan teks yang diketik sebagai baris baru sebuah log (`JTextArea` di Swing, widget `Text` atau `Listbox` di Tkinter), mengosongkan field untuk baris berikutnya, dan mengabaikan masukan kosong. Perluas uji mandirinya untuk mengetik tiga baris, salah satunya kosong, lalu mencetak log.

Soal Latihan 10.5 (oval di tempat Anda mengeklik). Tulislah, dalam kedua bahasa, panel yang menampilkan label *Click to draw an oval* dan menggambar lingkaran berdiameter 20 yang *berpusat* di titik tempat pengguna terakhir mengeklik, seperti di **Gambar 10.21**. Sebelum klik pertama, pusat lingkaran ada di (50,50).



Gambar 10.21 Program oval setelah sebuah klik: lingkaran kecil digambar di sekeliling titik yang diklik.

Soal Latihan 10.6 (penjepitan). Dengan Persamaan (10.2) dan penalaran Persamaan (10.3), berapa kali klik *Darken* membawa kotak **Kode 10.11** dari putih ke hitam bila langkahnya 15, bukan 10? Bila 7? Untuk langkah berapa saja antara 1 dan 20 klik terakhirnya *tidak* dijepit, yaitu tepat mendarat di 0?

Soal Latihan 10.7 (dari `getSource` ke lambda). Tulis ulang **Kode 10.11** sehingga panel tidak lagi mengimplementasikan antarmuka listener apa pun: setiap tombol mendapat lambda sendiri, dan klik tetikus ditangani oleh sebuah `MouseAdapter` anonim. Metode mana yang hilang dari kelas? Metode kelas lama mana yang tadinya dapat dipanggil kelas lain tetapi kini tidak lagi?

Soal Latihan 10.8 (late binding). Ramalkan apa yang dicetak program Python ini, jelaskan mengapa, dan perbaiki dengan dua cara yang berbeda.

```
fs = [lambda: i * 10 for i in range(3)]
print([f() for f in fs])
```

Soal Latihan 10.9 (effectively final). `count` adalah field kelas; variabel lainnya adalah variabel lokal metode `wire`. Baris mana dari (1)–(5) yang dapat dikompilasi? Untuk setiap baris yang tidak, jelaskan alasannya dalam satu kalimat.

```
int a = 1;
int b = 2;
b = 3;
final int[] c = {0};
String d = "x";
button.addActionListener(e -> System.out.println(a));  // (1)
button.addActionListener(e -> System.out.println(b));  // (2)
button.addActionListener(e -> c[0]++);                 // (3)
button.addActionListener(e -> d = d + "!");            // (4)
button.addActionListener(e -> count++);                // (5)
```

Soal Latihan 10.10 (aturan thread). Listener berikut dimaksudkan untuk mengunduh daftar harga hari ini dan menampilkan berapa layanan yang dimuatnya. Temukan dua pelanggaran aturan thread, jelaskan apa yang akan dialami pengguna akibat masing-masing, dan betulkan kodenya.

```
refresh.addActionListener(e -> {
    status.setText("Downloading...");
    String text = download("https://example.com/prices.csv");  // makan waktu 2-5 detik
    new Thread(() -> status.setText(text.lines().count() + " services")).start();
});
```

Soal Latihan 10.11 (papan mesin). Laundry memiliki empat mesin cuci, M1–M4. Tulislah sebuah panel, dalam kedua bahasa, yang menggambarnya sebagai empat lingkaran berdiameter 60 dalam satu baris, hijau bila bebas dan merah bila sibuk, dengan namanya di bawah; klik di dalam lingkaran membalik keadaan mesin itu, dan label di atas menampilkan *Busy: k of 4*. Klik yang berada di dalam persegi pembatas lingkaran tetapi di luar lingkarannya harus diabaikan. Berikan setiap program sebuah `--selftest`.

Soal Latihan 10.12 (anggota). Anggota mendapat potongan 10%, dihitung seperti di Bab 7: harga P menjadi $P - \lfloor P/10 \rfloor$. Tambahkan kotak centang *Member* ke jendela laundry sehingga potongan berlaku untuk pesanan yang sedang dimasukkan, harga langsungnya memperlihatkannya, dan daftar menandai pesanan anggota. Sebutkan dengan tepat kelas model dan metode view mana yang berubah, dalam kedua bahasa, dan hitung harga pesanan Siti di **Gambar 10.18** bila ia anggota.

Soal Latihan 10.13 (menguji model tanpa jendela). Tulislah kelas uji `unittest` untuk model **Kode 10.20** dengan sedikitnya empat pengujian: berat minimum, aturan kilat, koma desimal, dan galat validasi yang menyebut field yang benar. Perlihatkan bahwa pengujian berjalan di mesin tanpa layar, dan jelaskan mengapa hal itu mungkin walaupun modulnya mengimpor `tkinter`.

10.17 Penyelesaian Soal Latihan

Penyelesaian 10.1. Listener berjalan sesuai urutan pendaftarannya, jadi setiap klik kini mencetak baris listener kedua tepat setelah listener pertama berjalan, dan peristiwa gambar ulang, yang diantrekan listener pertama, disalurkan pada putaran gelung berikutnya:

```
loop: text -> weightField
loop: click -> addButton
    button says: clicked
loop: repaint -> totalLabel
    screen shows Rp28000
loop: text -> weightField
loop: click -> addButton
    button says: clicked
loop: repaint -> totalLabel
    screen shows Rp49000
loop: close -> window
loop ended; total = Rp49000
```

Bila peristiwa gambar ulang diantrekan sebelum total dinaikkan, peristiwa itu membawa teks `"Rp" + total` yang dihitung saat ia diciptakan — total yang *lama* — dan layar selalu terlambat satu klik:

```
loop: text -> weightField
loop: click -> addButton
loop: repaint -> totalLabel
    screen shows Rp0
loop: text -> weightField
loop: click -> addButton
loop: repaint -> totalLabel
    screen shows Rp28000
loop: close -> window
loop ended; total = Rp49000
```

Di toolkit sungguhan kekeliruan ini tidak mungkin terjadi, karena `repaint()` tidak membawa data: `paintComponent` membaca field *ketika ia berjalan*, setelah listener selesai, sehingga urutan `repaint()` dan pembaruan field di dalam satu listener tidak berpengaruh. Aturannya: permintaan gambar ulang sebaiknya menyatakan *bahwa* sesuatu berubah, bukan *menjadi apa* ia berubah; kode lukislah yang membaca keadaan terkini.

Penyelesaian 10.2. Tepi-tepi ditempatkan lebih dulu. `PAGE_START`: (0,0,400,30), selebar penuh pada tinggi yang disukainya. `PAGE_END`: (0,180,400,20), di bawah. Tinggi di antara keduanya $200 - 30 - 20 = 150$. `LINE_START`: (0,30,80,150). `LINE_END`: $(400 - 60, 30, 60, 150) = (340, 30, 60, 150)$. `CENTER`: semua sisanya, $(80, 30, 400 - 80 - 60, 150) = (80, 30, 260, 150)$; ukuran yang disukainya, 10 × 10, diabaikan. Menjalankan `report` pada panel seperti itu mencetak:

```
Border in 400 x 200:
  T                             x=  0 y=  0 w=400 h= 30
  B                             x=  0 y=180 w=400 h= 20
  L                             x=  0 y= 30 w= 80 h=150
  R                             x=340 y= 30 w= 60 h=150
  C                             x= 80 y= 30 w=260 h=150
```

Penyelesaian 10.3. (a) Barisnya terdiri atas $94 + 94 + 94 + 187 + 42 = 511$ piksel tombol ditambah empat celah di antaranya, $4 \times 5 = 20$, yaitu 531, ditambah celah 5 di setiap ujung: $531 + 10 = 541$. (b) Percobaan berkata lain: pada lebar 536 kelima tombol masih muat (baris mulai di $x = 3$), dan baru pada 535 tombol *5* pindah ke baris kedua:

```
FlowLayout in 536 x 70:
  Button 1                      x=  3 y=  5 w= 94 h= 25
  Button 2                      x=102 y=  5 w= 94 h= 25
  Button 3                      x=201 y=  5 w= 94 h= 25
  Long-Named Button 4           x=300 y=  5 w=187 h= 25
  5                             x=492 y=  5 w= 42 h= 25
FlowLayout in 535 x 70:
  Button 1                      x= 25 y=  5 w= 94 h= 25
  Button 2                      x=124 y=  5 w= 94 h= 25
  Button 3                      x=223 y=  5 w= 94 h= 25
  Long-Named Button 4           x=322 y=  5 w=187 h= 25
  5                             x=246 y= 35 w= 42 h= 25
```

(c) `FlowLayout` mengurangi lebar dengan kedua celah ujung, $536 - 10 = 526$, lalu, untuk setiap komponen, memeriksa apakah *komponen itu sendiri* muat setelah bagian baris yang sudah terpakai — sebelum menambahkan celah di depannya. Setelah empat tombol, baris memakai $94 + 5 + 94 + 5 + 94 + 5 + 187 = 484$, dan $484 + 42 = 526$ muat, padahal bersama celahnya tombol itu memerlukan 531. Baris itu lalu diletakkan di tengah dalam 536 piksel, sehingga margin menyusut dari 5 menjadi 3 dan 2. Pelajarannya bukan rincian itu, melainkan caranya: dokumentasi layout manager menggambarkan maksudnya; hanya menjalankannya (seperti yang dilakukan `report`) yang memberi tahu Anda aritmetika persisnya.

Penyelesaian 10.4. Di Swing, tambahkan field `private final JTextArea log = new JTextArea(5, 20);`, taruh di dalam `JScrollPane` di bawah field, jadikan hanya-baca dengan `log.setEditable(false)`, dan ubah listener-nya:

```
public void actionPerformed(ActionEvent e) {
    String s = t.getText().strip();
    if (!s.isEmpty()) {
        log.append(s + "\n");
    }
    t.setText("");
}
```

Di Tkinter, buat `self.log = tk.Listbox(self, height=5)` dan tulis

```
def on_enter(self, event):
    s = self.t.get().strip()
    if s:
        self.log.insert(tk.END, s)
    self.t.delete(0, tk.END)
```

Uji mandirinya mengetik setiap baris dengan `setText` / `insert`, memicu listener dengan `postActionEvent()` / `on_enter(None)`, dan akhirnya mencetak `log.getText()` / `log.get(0, tk.END)`: dua baris, karena masukan kosong diabaikan.

Penyelesaian 10.5. Keadaannya adalah pusat (x, y); kode lukis mengubahnya menjadi kotak pembatas yang dibutuhkan `drawOval`, yang pojok kiri atasnya 10 piksel (jari-jarinya) di atas dan di kiri pusat:

```
public class OvalClick extends JPanel {
    private int x = 50, y = 50;  // pusat lingkaran

    public OvalClick() {
        setPreferredSize(new Dimension(350, 300));
        add(new JLabel("Click to draw an oval"));
        addMouseListener(new MouseAdapter() {
            @Override
            public void mouseClicked(MouseEvent e) {
                x = e.getX();
                y = e.getY();
                repaint();
            }
        });
    }

    @Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);
        g.drawOval(x - 10, y - 10, 20, 20);  // kotak pembatas di sekeliling pusat
    }
}
```

Di Tkinter, oval itu adalah item yang diciptakan sekali dan dipindahkan pada setiap klik:

```
self.oval = self.canvas.create_oval(40, 40, 60, 60)
self.canvas.bind("<Button-1>", self.on_click)

def on_click(self, event):
    self.canvas.coords(self.oval, event.x - 10, event.y - 10,
                       event.x + 10, event.y + 10)
```

Kekeliruan yang umum adalah `g.drawOval(e.getX(), e.getY(), 20, 20)`, yang menaruh *pojok* kotak pembatas di titik klik, sehingga lingkaran tampak di bawah kanan penunjuk.

Penyelesaian 10.6. Dari putih (255) ke hitam, banyaknya klik dengan langkah s adalah $\lceil 255/s \rceil$. Untuk $s = 15$: $255/15 = 17$ tepat, jadi 17 klik, dan klik terakhir mendarat di 0 tanpa dijepit. Untuk $s = 7$: $255/7 \approx 36.43$, jadi 37 klik; setelah 36 klik tingkatnya $255 - 252 = 3$, dan klik ke-37 dijepit dari -4 menjadi 0. Klik terakhir tidak dijepit tepat bila s membagi 255. Karena $255 = 3 \times 5 \times 17$, langkah antara 1 dan 20 yang membaginya adalah 1, 3, 5, 15, dan 17.

Penyelesaian 10.7. Konstruktornya menjadi

```
darken.addActionListener(e -> shade(-10));
lighten.addActionListener(e -> shade(+10));
addMouseListener(new MouseAdapter() {
    @Override
    public void mouseClicked(MouseEvent e) {
        if (BOX.contains(e.getPoint())) {
            shade(+10);
        }
    }
});
```

dan kepala kelas kehilangan `implements ActionListener, MouseListener`. Enam metode hilang: `actionPerformed` (dengan uji-uji `getSource`-nya) dan kelima metode `MouseListener`, empat di antaranya kosong. Sebelum perubahan, kelas mana pun dapat memanggil `panel.actionPerformed(...)` atau `panel.mouseClicked(...)` dan mengubah kotak tanpa klik sungguhan, karena metode antarmuka bersifat publik; sesudahnya, satu-satunya cara mengubah kotak adalah melalui tombol dan tetikus. Listener telah menjadi rincian implementasi, sesuai tuntutan enkapsulasi Bab 7.

Penyelesaian 10.8. Program itu mencetak `[20, 20, 20]`. *List comprehension* menciptakan tiga lambda, yang semuanya merujuk ke variabel `i` yang sama; mereka baru mencarinya saat

dipanggil, dan saat itu *comprehension* sudah selesai dengan `i` bernilai 2. Dua perbaikan, keduanya mencetak `[0, 10, 20]`:

```
fs = [lambda i=i: i * 10 for i in range(3)]                      # argumen bawaan
fs = [functools.partial(lambda i: i * 10, i) for i in range(3)]  # partial
```

Cara ketiga, sering paling jelas dalam program sungguhan, adalah fungsi pabrik `def times_ten(i): return lambda: i * 10`, dipanggil sebagai `times_ten(i)`: setiap pemanggilan menciptakan sel baru.

Penyelesaian 10.9. Baris (1), (3), dan (5) dapat dikompilasi; (2) dan (4) tidak:

```
Capture.java:13: error: local variables referenced from a lambda expression must be final
or effectively final
        button.addActionListener(e -> System.out.println(b));  // (2)
                                                         ^
Capture.java:15: error: local variables referenced from a lambda expression must be final
or effectively final
        button.addActionListener(e -> d = d + "!");            // (4)
                                       ^
Capture.java:15: error: local variables referenced from a lambda expression must be final
or effectively final
        button.addActionListener(e -> d = d + "!");            // (4)
                                      ^
3 errors
```

(1)`a` diisi sekali: effectively final. (2) `b` diisi dua kali, jadi tidak effectively final, walaupun pengisian kedua terjadi sebelum lambda. (3) `c` bersifat final; lambda mengubah elemen larik, bukan variabelnya. (4) Lambda itu sendiri mengisi `d`, sesuatu yang tidak pernah boleh dilakukan lambda terhadap variabel lokal yang ditangkap (kompilator melaporkan operasi baca maupun operasi tulisnya). (5) `count` adalah field, dicapai melalui `this` yang ditangkap, jadi boleh dibaca dan diubah.

Penyelesaian 10.10. Pelanggaran pertama: `download` berjalan di dalam listener, di EDT, selama dua sampai lima detik; jendela membeku, dan bahkan teks *Downloading…* tidak pernah muncul, karena label baru dapat digambar ulang setelah listener kembali. Pelanggaran kedua: thread baru memanggil `status.setText`, menyentuh komponen Swing di luar EDT, yang sesekali dapat merusak tampilan. Versi yang benar mengunduh di pekerja dan memperbarui label di `done()`:

```
refresh.addActionListener(e -> {
    status.setText("Downloading...");
    refresh.setEnabled(false);
    new SwingWorker<String, Void>() {
        @Override
        protected String doInBackground() throws Exception {
            return download("https://example.com/prices.csv");
        }

        @Override
        protected void done() {
            try {
                status.setText(get().lines().count() + " services");
            } catch (Exception ex) {
                status.setText("Download failed: " + ex.getMessage());
            }
            refresh.setEnabled(true);
        }
    }.execute();
});
```

Menonaktifkan tombol selama unduhan berjalan mencegah unduhan kedua yang tumpang-tindih, dan `catch` mengubah kegagalan jaringan menjadi pesan, bukan kebisuan.

Penyelesaian 10.11. **Kode 10.21** dan **Kode 10.22** adalah kedua programnya. Titik (x, y) terletak di dalam lingkaran berpusat (c_x, c_y) dan berjari-jari ρ tepat bila jaraknya dari pusat paling besar ρ; mengkuadratkan kedua ruas agar tidak memerlukan akar kuadrat menghasilkan uji

$$(x - c_x)^2 + (y - c_y)^2 \le \rho^2 \tag{10.8}$$

yang diterapkan `machineAt` pada setiap mesin secara bergiliran, dengan $\rho = 30$.

Kode 10.21 `MachineBoard.java`: empat mesin dilukis sebagai lingkaran; klik di dalam lingkaran, diuji dengan Persamaan (10.8), membalik keadaan mesin itu.

```
import java.awt.Color;
import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.awt.image.BufferedImage;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.SwingUtilities;

public class MachineBoard extends JPanel {
    static final int N = 4, SIZE = 60, GAP = 20, TOP = 40;
    private final boolean[] busy = new boolean[N];
    private final JLabel status = new JLabel();

    public MachineBoard() {
        setPreferredSize(new Dimension(N * (SIZE + GAP) + GAP, TOP + SIZE + 40));
        add(status);
        updateStatus();
        addMouseListener(new MouseAdapter() {
            @Override
            public void mouseClicked(MouseEvent e) {
                int m = machineAt(e.getX(), e.getY());
                if (m >= 0) {
                    busy[m] = !busy[m];
                    updateStatus();
                    repaint();
                }
            }
        });
    }

    static int left(int m) {
        return GAP + m * (SIZE + GAP);
    }
```

Kode 10.21 (Lanjutan) *MachineBoard.java: empat mesin dilukis sebagai lingkaran; klik di dalam lingkaran, diuji dengan Persamaan (10.8), membalik keadaan mesin itu.*

```
    static int machineAt(int x, int y) {  // lingkaran mana yang memuat (x, y)?
        for (int m = 0; m < N; m++) {
            double cx = left(m) + SIZE / 2.0, cy = TOP + SIZE / 2.0;
            double dx = x - cx, dy = y - cy;
            if (dx * dx + dy * dy <= (SIZE / 2.0) * (SIZE / 2.0)) {
                return m;
            }
        }
        return -1;
    }

    private void updateStatus() {
        int n = 0;
        for (boolean b : busy) {
            n += b ? 1 : 0;
        }
        status.setText("Busy: " + n + " of " + N);
    }

    @Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);
        for (int m = 0; m < N; m++) {
            g.setColor(busy[m] ? new Color(200, 40, 40) : new Color(40, 160, 70));
            g.fillOval(left(m), TOP, SIZE, SIZE);
            g.setColor(Color.BLACK);
            g.drawOval(left(m), TOP, SIZE, SIZE);
            g.drawString("M" + (m + 1), left(m) + SIZE / 2 - 8, TOP + SIZE + 16);
        }
    }

    public static void main(String[] args) throws Exception {
        if (args.length > 0 && args[0].equals("--selftest")) {
            System.setProperty("java.awt.headless", "true");
            SwingUtilities.invokeAndWait(MachineBoard::selfTest);
        } else {
            SwingUtilities.invokeLater(() -> {
                JFrame.setDefaultLookAndFeelDecorated(true);
                JFrame frame = new JFrame("Machine board");
                frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                frame.add(new MachineBoard());
                frame.pack();
                frame.setVisible(true);
            });
        }
    }
```

Kode 10.21 (Lanjutan) *MachineBoard.java: empat mesin dilukis sebagai lingkaran; klik di dalam lingkaran, diuji dengan Persamaan (10.8), membalik keadaan mesin itu.*

```
    void click(int x, int y) {
        dispatchEvent(new MouseEvent(this, MouseEvent.MOUSE_CLICKED,
                System.currentTimeMillis(), 0, x, y, 1, false, MouseEvent.BUTTON1));
    }

    static void selfTest() {
        MachineBoard b = new MachineBoard();
        System.out.println("start:             " + b.status.getText());
        b.click(50, 70);    // pusat M1
        b.click(210, 70);   // pusat M3
        System.out.println("click M1, M3:      " + b.status.getText());
        b.click(20, 40);    // di dalam persegi M1 tetapi di luar lingkarannya
        System.out.println("click M1's corner: " + b.status.getText());
        b.click(50, 70);
        System.out.println("click M1 again:    " + b.status.getText());
        b.setSize(b.getPreferredSize());
        BufferedImage img = new BufferedImage(b.getWidth(), b.getHeight(),
                BufferedImage.TYPE_INT_RGB);
        b.paint(img.getGraphics());
        for (int m = 0; m < N; m++) {
            System.out.printf("M%d center pixel: #%06x%n", m + 1,
                    img.getRGB(left(m) + SIZE / 2, TOP + SIZE / 2) & 0xFFFFFF);
        }
    }
}
```

```
start:             Busy: 0 of 4
click M1, M3:      Busy: 2 of 4
click M1's corner: Busy: 2 of 4
click M1 again:    Busy: 1 of 4
M1 center pixel: #28a046
M2 center pixel: #28a046
M3 center pixel: #c82828
M4 center pixel: #28a046
```

Klik di (20,40) berada di dalam persegi pembatas M1, yang membentang dari (20,40) sampai (80,100), tetapi jaraknya dari pusat (50,70) adalah $\sqrt{30^2 + 30^2} \approx 42.4 > 30$, sehingga Persamaan (10.8) menolaknya dan tidak ada yang berubah. Piksel di keempat pusat memastikan bahwa pada akhirnya hanya M3 yang dilukis merah.

Kode 10.22 `machine_board.py`: papan yang sama di kanvas Tkinter, tempat `tag_bind` pada setiap oval melakukan uji kenanya.

```
"""machine_board.py: klik mesin cuci untuk menandainya sibuk atau bebas."""
import sys
import tkinter as tk

N, SIZE, GAP, TOP = 4, 60, 20, 10
FREE, BUSY = "#28a046", "#c82828"


class MachineBoard(tk.Frame):
    def __init__(self, master):
        super().__init__(master)
        self.busy = [False] * N
        self.status = tk.Label(self)
        self.status.pack()
        self.canvas = tk.Canvas(self, width=N * (SIZE + GAP) + GAP,
                                height=TOP + SIZE + 30, highlightthickness=0)
        self.canvas.pack()
        self.circles = []
        for m in range(N):
            x = GAP + m * (SIZE + GAP)
            c = self.canvas.create_oval(x, TOP, x + SIZE, TOP + SIZE,
                                        fill=FREE, outline="black")
            self.canvas.create_text(x + SIZE / 2, TOP + SIZE + 14, text=f"M{m + 1}")
            # Tk yang menguji "apakah tetikus di atas oval ini?" untuk kita.
            self.canvas.tag_bind(c, "<Button-1>", lambda event, m=m: self.toggle(m))
            self.circles.append(c)
        self.update_status()

    def toggle(self, m):
        self.busy[m] = not self.busy[m]
        self.canvas.itemconfig(self.circles[m], fill=BUSY if self.busy[m] else FREE)
        self.update_status()

    def update_status(self):
        self.status.config(text=f"Busy: {sum(self.busy)} of {N}")
```

Kode 10.22 (Lanjutan) *machine_board.py: papan yang sama di kanvas Tkinter, tempat tag_bind pada setiap oval melakukan uji kenanya.*

```
def self_test():
    root = tk.Tk()
    root.withdraw()
    b = MachineBoard(root)
    b.pack()
    print("start:            ", b.status.cget("text"))
    b.toggle(0)  # yang dipanggil Tk untuk klik pada M1
    b.toggle(2)
    print("click M1, M3:     ", b.status.cget("text"))
    b.toggle(0)
    print("click M1 again:   ", b.status.cget("text"))
    for m, c in enumerate(b.circles):
        print(f"M{m + 1} fill: {b.canvas.itemcget(c, 'fill')}")
    root.destroy()


def main():
    if "--selftest" in sys.argv:
        self_test()
        return
    root = tk.Tk()
    root.title("Machine board")
    MachineBoard(root).pack(padx=6, pady=6)
    root.mainloop()


if __name__ == "__main__":
    main()
```

```
start:            Busy: 0 of 4
click M1, M3:     Busy: 2 of 4
click M1 again:   Busy: 1 of 4
M1 fill: #28a046
M2 fill: #28a046
M3 fill: #c82828
M4 fill: #28a046
```

Di Tkinter, `tag_bind` pada setiap oval membuat Tk yang melakukan uji kena, dan Tk menguji bentuk oval yang sebenarnya, bukan kotak pembatasnya, sehingga klik di pojok diabaikan tanpa kode kita sama sekali. `lambda event, m=m: self.toggle(m)` memakai obat argumen bawaan dari Subbab 10.10.5; tanpa `m=m`, setiap lingkaran akan membalik M4.

Penyelesaian 10.12. Model berubah lebih dulu, dan view mengikuti.

- **Model.** `quote` mendapat parameter `member` dan, setelah aturan kilat, menerapkan `price = price - price / 10` (Java, pembagian bilangan bulat) atau `price -= price // 10` (Python). `OrderLine` mendapat field `member`, dan `display()` menambahkan penanda seperti `M` untuk pesanan anggota. `OrderForm.build` / `build_order` meneruskan penanda itu. Tidak ada hal lain di model yang berubah, dan `OrderBook.total()` sama sekali tidak perlu diubah, karena ia menjumlahkan harga yang sudah memuat potongan.
- **View.** Tambahkan `memberBox` (sebuah `JCheckBox`) atau `member_var` dengan sebuah `Checkbutton`, di samping *Express*; daftarkan `e -> updatePrice()` padanya (Tkinter: `command=self.update_price`); berikan `memberBox.isSelected()` kepada `quote` di `updatePrice` dan kepada `build` di `addOrder`; kembalikan nilainya di `clearForm`.

Pesanan Siti sebagai anggota: $P = 31{,}500$, potongan $\lfloor 31{,}500/10 \rfloor = 3{,}150$, harga 28,350. Perhatikan urutan aturannya: untuk pesanan kilat seorang anggota, potongan berlaku pada harga

kilat, $\lfloor 3P/2 \rfloor - \lfloor \lfloor 3P/2 \rfloor / 10 \rfloor$, sama seperti `Order` di Bab 7 menerapkan potongan anggota pada total yang sudah memuat biaya tambahan kilat.

Penyelesaian 10.13. Sebuah berkas uji `test_model.py` di samping program:

```
import unittest
from laundry_order_window import (FormError, LaundryService, build_order,
                                  parse_kg, quote)


class ModelTest(unittest.TestCase):
    def test_minimum_and_rounding(self):
        q = quote(LaundryService.IRON_ONLY, 1.3, False)
        self.assertEqual((q.chargeable_kg, q.price, q.hours), (2.0, 10000, 24))

    def test_express(self):
        q = quote(LaundryService.WASH_DRY_IRON, 2.6, True)
        self.assertEqual((q.price, q.hours), (45000, 18))

    def test_decimal_comma(self):
        self.assertEqual(parse_kg(" 4,2 "), 4.2)

    def test_bad_phone_names_the_field(self):
        with self.assertRaises(FormError) as ctx:
            build_order("ORD-0001", "Ayu", "12345", LaundryService.WASH_FOLD, "3", False)
        self.assertEqual(ctx.exception.field, "phone")


if __name__ == "__main__":
    unittest.main()
```

Bila dijalankan dengan variabel tampilan dihapus (`env -u DISPLAY python3 test_model.py` di Linux), ia melaporkan:

```
Ran 4 tests in 0.000s

OK
```

Mengimpor `laundry_order_window` berarti mengimpor `tkinter`, yang hanya memuat pustakanya; sebuah tampilan baru diperlukan ketika `tk.Tk()` dipanggil, dan fungsi-fungsi model tidak pernah memanggilnya. Inilah imbalan praktis Subbab 10.12.3: karena aturan-aturannya tinggal di kelas yang bebas toolkit, aturan itu dapat diuji di mana saja, dengan cepat, memakai perkakas Bab 6. Hal yang sama berlaku di Java, tempat pengujian JUnit dapat memanggil `OrderForm.quote` dan `OrderForm.build` tanpa setelan `java.awt.headless` sama sekali.

Daftar Pustaka

[1] Oracle, “Creating a GUI with Swing,” *The Java Tutorials*. [Online]. Available: https://docs.oracle.com/javase/tutorial/uiswing/

[2] OpenJFX project, “JavaFX 27 Highlights.” [Online]. Available: https://openjfx.io/highlights/27/ (accessed Sep. 2026).

[3] Oracle, “The Arrival of Java 27,” *Inside.java*, Sep. 15, 2026. [Online]. Available: https://inside.java/2026/09/15/jdk-27-available/

[4] OpenJDK, “JEP 398: Deprecate the Applet API for removal,” 2021. [Online]. Available: https://openjdk.org/jeps/398

[5] P. Race, “JEP 504: Remove the Applet API,” OpenJDK, 2025 (delivered in JDK 26, March 2026). [Online]. Available: https://openjdk.org/jeps/504

[6] J. K. Ousterhout and K. Jones, *Tcl and the Tk Toolkit*, 2nd ed. Upper Saddle River, NJ, USA: Addison-Wesley, 2009.

[7] Python Software Foundation, “tkinter — Python interface to Tcl/Tk,” *The Python Standard Library*, version 3.14. [Online]. Available: https://docs.python.org/3/library/tkinter.html

[8] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Reading, MA, USA: Addison-Wesley, 1994.

[9] Oracle, “A visual guide to layout managers,” *The Java Tutorials*. [Online]. Available: https://docs.oracle.com/javase/tutorial/uiswing/layout/visual.html

[10] A. Fowler, “Painting in AWT and Swing,” Sun Microsystems technical article, 2000. [Online]. Available: https://www.oracle.com/java/technologies/painting.html

[11] J. Gosling, B. Joy, G. Steele, G. Bracha, A. Buckley, D. Smith, and G. Bierman, *The Java Language Specification, Java SE 21 Edition*. Oracle America, 2023, §8.1.3 (inner classes), §15.9.5 (anonymous classes), §15.13 (method references), §15.27 (lambda expressions). [Online]. Available: https://docs.oracle.com/javase/specs/jls/se21/html/

[12] L. Ramalho, *Fluent Python*, 2nd ed. Sebastopol, CA, USA: O’Reilly Media, 2022, ch. 9.

[13] Python Software Foundation, “Why do lambdas defined in a loop with different values all return the same result?,” *Programming FAQ*. [Online]. Available: https://docs.python.org/3/faq/programming.html

[14] J. Bloch, *Effective Java*, 3rd ed. Boston, MA, USA: Addison-Wesley, 2018, Items 24 and 42.

[15] B. Goetz, T. Peierls, J. Bloch, J. Bowbeer, D. Holmes, and D. Lea, *Java Concurrency in Practice*. Upper Saddle River, NJ, USA: Addison-Wesley, 2006, ch. 9.

[16] Oracle, “Concurrency in Swing,” *The Java Tutorials*. [Online]. Available: https://docs.oracle.com/javase/tutorial/uiswing/concurrency/

[17] J. Nielsen, *Usability Engineering*. Boston, MA, USA: Academic Press, 1993, ch. 5.

[18] G. E. Krasner and S. T. Pope, “A cookbook for using the model-view-controller user interface paradigm in Smalltalk-80,” *Journal of Object-Oriented Programming*, vol. 1, no. 3, pp. 26–49, 1988.

Bab 11
Pola Model–View–Controller

Pada akhir bab ini Anda akan sanggup menjelaskan mengapa program berjendela sebaiknya dipecah menjadi sebuah **model** yang menyimpan data beserta aturannya, satu atau lebih **view** (tampilan) yang memperlihatkan data itu, dan sebuah **controller** (pengendali) yang mengubah tindakan pengguna menjadi operasi terhadap model; menggambar panah-panah di antara ketiganya dan menyebutkan siapa boleh mengenal siapa; mengimplementasikan pola **Observer** (pengamat) yang memungkinkan sebuah model mengumumkan perubahannya tanpa tahu siapa yang mendengarkan, dengan `PropertyChangeSupport` di Java dan sebuah list berisi *callable* di Python; membangun ulang sebuah pencacah kecil dan papan tombol ponsel dengan teks prediktif sebagai program MVC yang bersih, dalam Swing dan dalam Tkinter; memberi setiap program semacam itu mode `--selftest` yang menjalankan model dan controller tanpa jendela sama sekali; menguji model dengan *unit test* biasa, tanpa pernah menyentuh satu tombol pun; serta membangun aplikasi meja depan yang lengkap untuk Profitina Laundry, dengan model pesanan, view formulir, view daftar, view ringkasan, dan sebuah controller, berikut pengujiannya. Anda juga akan berkenalan dengan dua keturunan MVC yang paling terkenal, Model–View–Presenter dan Model–View–ViewModel, serta memahami apa yang diubah masing-masing dan mengapa.

11.1 Di Mana Kita Berhenti

Bab 10 memberi program-program kita jendela. Kita membangun *frame* dan panel di Swing maupun Tkinter, menata label, kolom teks, dan tombol di atasnya dengan *layout manager*, dan mempelajari gagasan inti setiap program grafis: setelah jendela selesai dibangun, program tidak lagi menentukan apa yang terjadi berikutnya. Ia menunggu. Pengguna mengeklik, mengetik, atau menutup jendela; *toolkit* mengubah gestur itu menjadi sebuah *event* (peristiwa), lalu memanggil sepotong kecil kode kita — *listener* di Java, *callback* di Python — yang sudah kita daftarkan sebelumnya. Kita menulis *listener* itu sebagai kelas dalam (*inner class*), sebagai lambda, dan sebagai *closure*, dan kita melihat bahwa masing-masing berjalan di satu-satunya *thread* peristiwa milik *toolkit*, sehingga harus cepat selesai.

Program-program Bab 10 kecil, dan wajar bila masing-masing ditulis sebagai satu kelas: kelas itu membangun jendela, dan *listener*-nya mengerjakan apa pun kegunaan tombolnya. Gaya itu indah untuk dua puluh baris. Gaya itu berhenti bekerja di sekitar dua ratus baris, dan bab ini dimulai dengan menyaksikan saat ia berhenti. Obatnya adalah cara membagi program grafis menjadi tiga jenis objek, masing-masing dengan satu tugas. Pembagian itu disebut **Model–View–Controller**, disingkat MVC. Umurnya lebih dari empat puluh tahun, ia adalah leluhur cara hampir semua aplikasi desktop, web, dan ponsel diorganisasi saat ini, dan — bagian yang paling penting bagi pemrogram yang bekerja — dialah yang membuat program berjendela dapat *diuji*.

Di bawah MVC terdapat gagasan yang lebih kecil yang akan kita jumpai di sepanjang jalan: pola **Observer**, tempat satu objek menyimpan daftar objek lain dan memberi tahu mereka semua setiap kali ia berubah. Anda sebenarnya sudah memakainya tanpa mengenal namanya: setiap `addActionListener` di Bab 10 mendaftarkan seorang pengamat pada sebuah tombol. Di bab ini kita menulis subjek kita sendiri, supaya *data* kita dapat memberi tahu *jendela* kita bahwa sesuatu telah terjadi.

> **WAWASAN KUNCI:** Jendela adalah tempat yang buruk untuk menyimpan kebenaran. Letakkan data dan aturannya di dalam objek yang tidak tahu apa-apa tentang jendela (model), biarkan sebanyak apa pun jendela menggambar objek itu (view), dan biarkan objek ketiga menerjemahkan klik menjadi permintaan kepada model (controller). Seluruh isi bab ini adalah penjabaran kalimat tersebut.

11.2 Sebuah Kisah: Meja Depan yang Tahu Terlalu Banyak

Meja depan Profitina Laundry sudah memiliki sebuah program kecil di komputernya sejak toko itu dibuka. Seorang pelanggan membawa sekantong pakaian; petugas meja depan mengetik nama pelanggan, memilih layanan — Wash & Fold, Wash, Dry & Iron, atau Iron Only — mengetik berat yang tertera di timbangan, mencentang *Express* bila pelanggan sedang terburu-buru, lalu mengeklik **Add order**. Pesanan itu muncul di sebuah daftar bersama harganya. Ketika pakaian berpindah dari mesin cuci ke meja lipat, petugas memilih pesanan itu dan mengeklik **Advance status**; kemudian pesanan menjadi *Ready*; akhirnya pelanggan mengambilnya dan statusnya menjadi *Picked up*.

Versi pertama program itu ditulis dalam satu sore oleh seorang mahasiswa yang bersemangat, sebagai satu kelas bernama `FrontDeskWindow`. Kelas itu subkelas `JFrame`. Konstruktornya membangun kolom teks dan tombol. *Listener* tombol **Add order** mengerjakan segala hal yang tercakup dalam “menambah pesanan”, kira-kira seperti ini:

```
add.addActionListener(e -> {
    String name = customerField.getText().trim();
    if (name.isEmpty()) { statusLabel.setText("Name?"); return; }
    double kg = Double.parseDouble(weightField.getText());
    double rate = serviceBox.getSelectedIndex() == 0 ? 7000
                : serviceBox.getSelectedIndex() == 1 ? 10000 : 5000;
    long price = Math.round(Math.ceil(kg * 2) / 2 * rate);
    if (expressBox.isSelected()) price = price * 3 / 2;
    tableModel.addRow(new Object[] {++count, name, kg, price, "Received"});
    total += price;
    totalLabel.setText("Total: Rp" + total);
});
```

Program itu bekerja. Kemudian, dalam kurun satu tahun, empat hal terjadi.

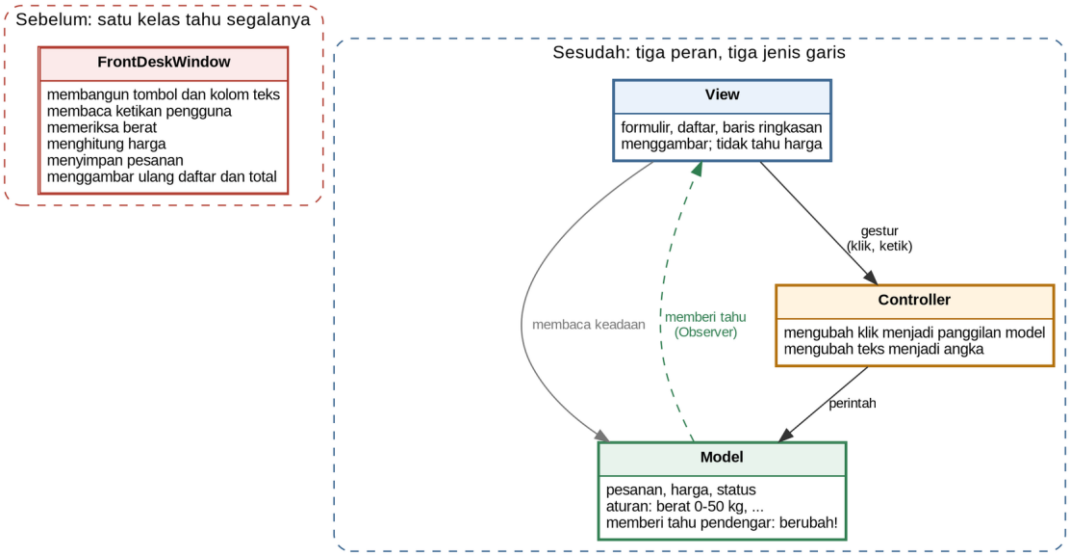
Pertama, pemilik menginginkan layar kedua. Sebuah televisi dipasang di ruang tunggu, dan pemilik ingin televisi itu menampilkan pesanan yang siap diambil, dengan huruf besar, supaya pelanggan dapat melihat sekilas apakah perlu mengantre. Di mana pesanan-pesanan itu berada? Mereka hanya ada sebagai baris-baris `DefaultTableModel` di dalam jendela meja depan, sebagai larik `Object`. Harganya berupa angka di satu sel tabel; statusnya berupa string di sel lain. Untuk menampilkan pesanan yang siap di tempat lain, layar baru harus merogoh ke dalam jendela lama dan membaca sel-sel tabelnya — atau pesanan-pesanan itu harus disalin, lalu kedua salinan itu mulai berbeda begitu ada orang yang lupa memperbarui salah satunya.

Kedua, aturan harga berubah. Setiap layanan mendapat berat minimum (tiga kilogram untuk cuci, dua untuk setrika), karena mencuci setengah kilogram menghabiskan air dan listrik hampir sebanyak mencuci tiga kilogram. Aturan itu milik bisnis, tetapi kodenya tinggal di dalam *listener* tombol, bersebelahan dengan panggilan `getText()` dan baris-baris tabel. Mahasiswa yang mengubahnya harus membaca kode *widget* untuk menemukan tiga baris aritmetika, dan sebuah tanda kurung yang salah tempat di tiga baris itu ditemukan oleh seorang pelanggan, bukan oleh pemrogramnya.

Ketiga, ada yang meminta pengujian. "Bisakah kita memeriksa secara otomatis bahwa Wash & Fold 4,2 kg tetap berharga Rp31,500 setelah setiap perubahan?" Dengan rancangan pertama, satu-satunya cara menghitung harga adalah membangun seluruh jendela, mengetik ke kolom-kolom teksnya, dan mengeklik tombolnya. Itu memerlukan layar. *Server* pembangun yang menjalankan pengujian tidak punya layar. Maka tidak ada pengujian, dan setiap perubahan diuji dengan tangan, dengan mengeklik, bila ada yang ingat.

Keempat, binatu itu menginginkan versi web untuk cabang keduanya. Tak satu baris pun dari kelas itu dapat dipakai ulang, karena setiap barisnya terjalin dengan Swing.

Keempat kisah ini tidak aneh. Itulah kehidupan sehari-hari program apa pun yang benar-benar dipakai orang. Dan penyebabnya satu: kelas `FrontDeskWindow` mengerjakan enam tugas berbeda, yang terdaftar di sisi kiri **Gambar 11.1**. Ia membangun *widget*, membaca ketikan pengguna, memeriksa berat, menghitung harga, menyimpan pesanan, dan menggambar ulang daftar serta total. Tugas-tugas yang berubah karena alasan yang berbeda — rupa jendela, aturan bisnis, penyimpanan data — dilas menjadi satu *listener*.



Gambar 11.1 Meja depan yang sama sebelum dan sesudah MVC. Kiri: satu kelas jendela mengerjakan enam tugas. Kanan: model menyimpan pesanan dan aturannya, view menggambar, dan controller menerjemahkan gestur menjadi panggilan ke model; panah putus-putus adalah pemberitahuan Observer, cara model memberi tahu view bahwa sesuatu telah berubah.

Separuh kanan **Gambar 11.1** memperlihatkan program yang sama setelah dibongkar mengikuti garis-garis yang ditarik MVC. **Model** menyimpan pesanan, harga dan statusnya, serta aturan-aturan tentang semua itu, dan ia dapat memberi tahu siapa saja yang berminat bahwa dirinya telah berubah. Ia tidak mengimpor satu kelas Swing atau Tkinter pun. **View** menggambar: sebuah formulir, sebuah daftar, sebaris total. Ia tidak memuat aturan harga sama sekali; ia menampilkan angka apa pun yang dilaporkan model. **Controller** mendengarkan tombol-tombol view, mengubah teks yang diketik pengguna menjadi nilai yang layak, lalu memanggil model. Kini setiap kisah hanya menyentuh satu bagian: televisi adalah view kedua dari model yang sama; aturan berat minimum adalah perubahan pada model saja; pengujian memanggil model secara langsung, tanpa jendela; dan versi web mempertahankan model serta mengganti view dan controller. Sisa bab ini memperlihatkan, dalam kedua bahasa, cara menulis program dengan cara itu.

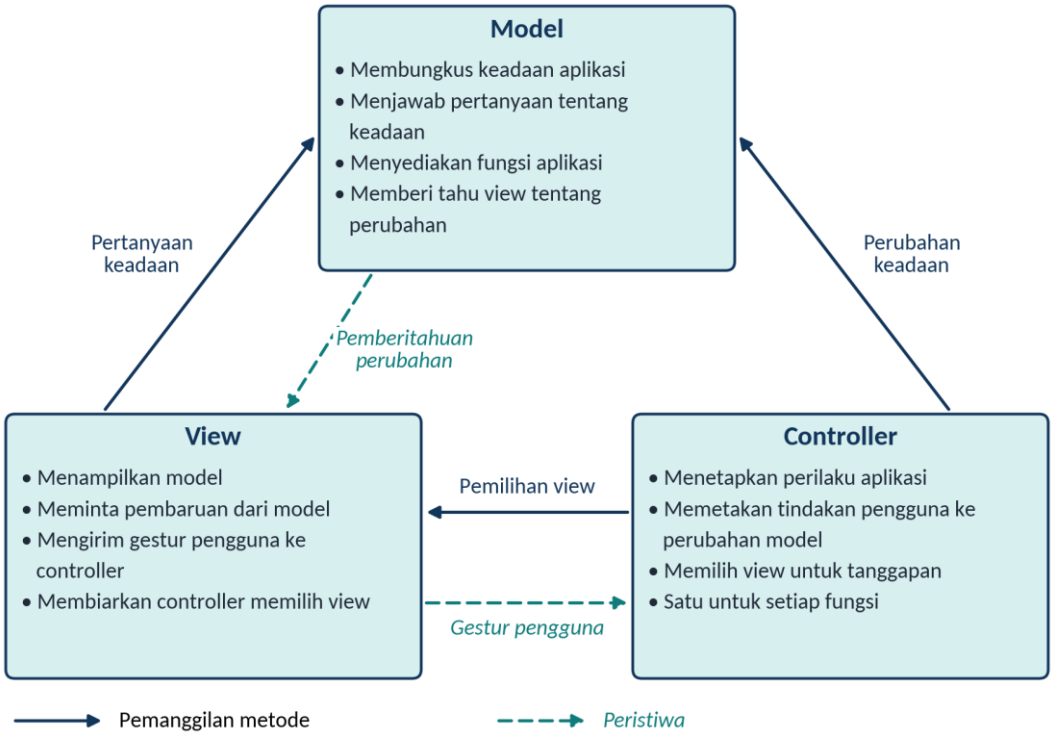
11.3 Tiga Peran: Model, View, Controller

Pola MVC memisahkan bagian aplikasi yang menyimpan dan mengatur data dari bagian yang menyajikannya, dan keduanya dari bagian yang menangani tindakan pengguna. Ketiga peran itu didefinisikan sebagai berikut.

> **DEFINISI. Model** mewakili data aplikasi beserta aturan yang mengatur akses dan perubahan terhadap data itu; dalam perangkat lunak bisnis, model sering merupakan perkiraan, dalam bentuk kode, dari sebuah proses di dunia nyata. **View** menampilkan isi sebuah model: ia menentukan dengan tepat bagaimana data disajikan, dan ketika data berubah, view memperbarui penyajiannya. **Controller** menerjemahkan interaksi pengguna dengan view menjadi tindakan yang dikerjakan model.

Perhatikan apa yang *tidak* disebut oleh setiap definisi. Definisi model tidak menyebut piksel, fon, atau tombol. Definisi view tidak menyebut apa *arti* data itu — view yang menampilkan harga sebuah pesanan tidak tahu bagaimana harga itu dihitung. Definisi controller tidak menyebut penyimpanan atau penggambaran data; controller adalah penerjemah antara dua bahasa, bahasa gestur ("tombol berlabel *Add order* diklik ketika kolom berat berisi `4,2`") dan bahasa bisnis ("tambahkan pesanan Wash & Fold seberat 4,2 kg untuk Sari").

Gambar 11.2 adalah gambar klasik ketiga peran itu beserta pesan-pesan di antaranya, digambar ulang dari gambar yang dipakai dokumentasi Java ketika menjelaskan cara merancang aplikasi Java dengan MVC [1]. Panah utuh adalah *pemanggilan metode*: satu objek memanggil metode objek lain. Panah putus-putus adalah *peristiwa*: satu objek mengumumkan bahwa sesuatu terjadi, tanpa tahu sebelumnya siapa yang akan bereaksi.



Gambar 11.2 Diagram MVC klasik. Model membungkus keadaan aplikasi, menjawab pertanyaan tentang keadaan itu, menyediakan fungsi aplikasi, dan memberi tahu view tentang perubahan; view menampilkan model dan mengirim gestur pengguna ke controller; controller memetakan tindakan pengguna menjadi perubahan model dan memilih view untuk tanggapannya.

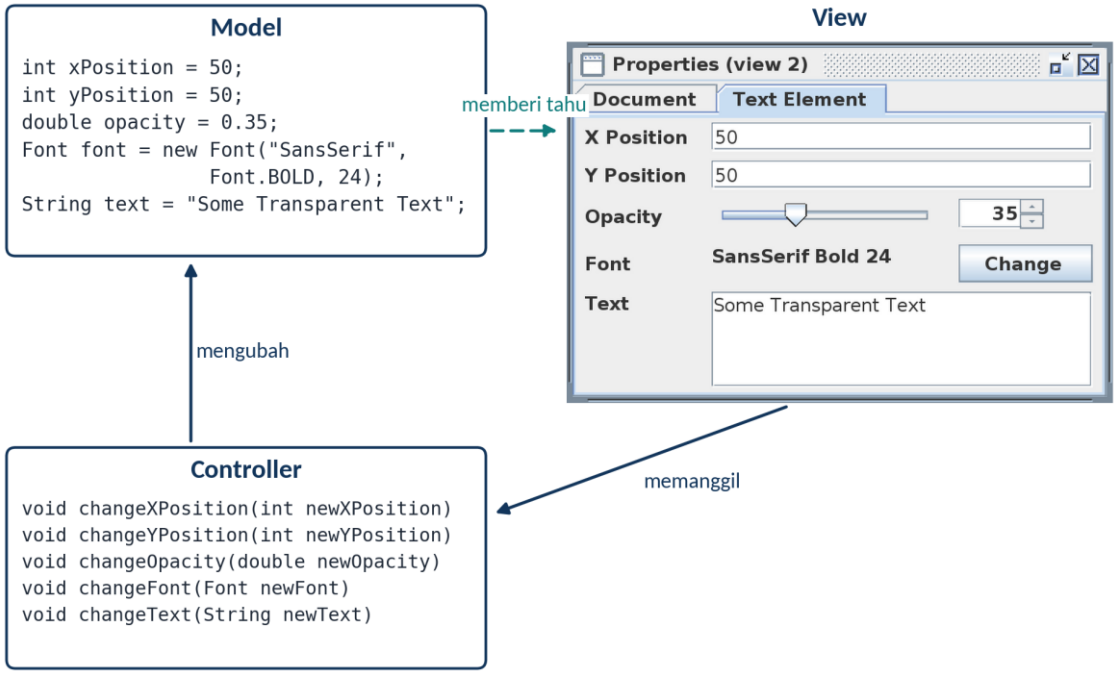
Ikutilah panah-panah di gambar itu sekali, seolah-olah Anda adalah sebuah klik. Pengguna menekan tombol di **view**; view tidak menindaklanjuti klik itu sendiri, tetapi meneruskannya

sebagai *gestur pengguna* kepada **controller**. Controller menentukan arti gestur itu dan memanggil metode **model** yang mengubah keadaannya (*state change*). Model berubah, dan karena view harus tetap mutakhir, model mengirim *pemberitahuan perubahan* — sebuah peristiwa — kepada setiap view yang telah mendaftarkan minatnya. Setiap view lalu menanyakan kepada model data yang ia perlukan (*state query*) dan menggambar ulang dirinya. Controller juga dapat memilih view *mana* yang ditampilkan berikutnya (*view selection*), yang penting dalam aplikasi dengan beberapa layar, seperti *wizard* atau situs web.

Model dalam siklus ini pasif dalam satu hal penting: ia tidak pernah memanggil sebuah view dengan namanya. Ia hanya mengumumkan "aku telah berubah" kepada daftar pendengar yang diberikan kepadanya. Itulah sebabnya panah dari model ke view putus-putus. Mekanisme pemberitahuan itu — pola Observer di Subbab 11.5 — yang memungkinkan model tetap tidak mengenal view, dan ia adalah bahan teknis terpenting dari MVC.

11.3.1 Model, view, dan controller yang dapat ditunjuk

Gambar 11.3 membuat ketiga peran itu konkret untuk sebuah program gambar mungil yang menempatkan sebaris teks semitransparan di sebuah halaman. Modelnya lima variabel: posisi x, posisi y, opasitas, fon, dan teks itu sendiri. View-nya adalah jendela *Properties* dengan tab *Text Element*, tempat kelima nilai itu ditampilkan dalam kolom teks, penggeser, *spinner*, dan area teks. Controller-nya adalah sekumpulan metode, satu untuk setiap properti: `changeXPosition`, `changeYPosition`, `changeOpacity`, `changeFont`, dan `changeText`.

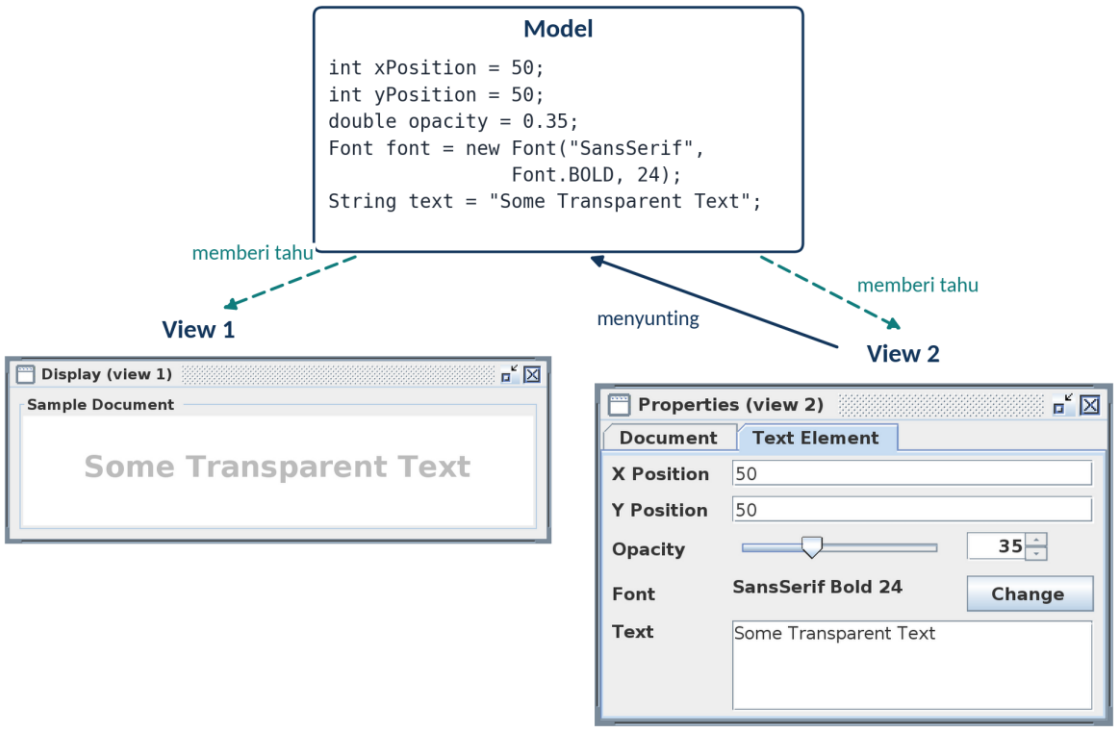


Gambar 11.3 Model, view, dan controller sebuah penyunting dokumen kecil. Modelnya lima nilai biasa; view-nya jendela properti yang menampilkan nilai-nilai itu; controller-nya menyediakan satu metode untuk setiap properti, yang dipanggil view ketika pengguna menyunting sebuah kolom.

Ketika pengguna menarik penggeser opasitas dari 35 ke 50, *listener* penggeser itu memanggil `changeOpacity(0.50)`; controller mengatur opasitas model; model memberi tahu view-view-nya; dan view menggambar ulang penggeser dan *spinner* pada angka 50. Perhatikan bahwa *spinner* di sebelah penggeser ikut berubah, padahal pengguna tidak menyentuhnya. Ia berubah *karena model berubah*, bukan karena penggeser menyuruhnya. Itulah manfaat praktis pertama MVC: dua

widget yang menampilkan nilai yang sama tidak mungkin berselisih, karena tak satu pun dari keduanya memiliki nilai itu.

Manfaat kedua adalah yang diinginkan pemilik binatu: **satu model dapat memiliki banyak view**. **Gambar 11.4** menambahkan jendela kedua untuk kelima nilai yang sama. Jendela *Display* menggambar teks itu di halaman, abu-abu dan semitransparan, di posisi (50, 50); jendela *Properties* menampilkan nilai-nilai yang sama sebagai angka. Ubahlah opasitas di jendela properti, dan teks di halaman memudar; ketik kalimat baru di area teks, dan halaman menampilkannya. Tak satu pun dari kedua jendela tahu bahwa yang lain ada. Masing-masing hanya mengenal model.



Gambar 11.4 Satu model, dua view. Jendela *Display* menggambar teks sebagaimana akan tampak di halaman, dan jendela *Properties* menampilkan nilai-nilai yang sama sebagai angka; keduanya dijaga tetap mutakhir oleh pemberitahuan perubahan dari model.

Hal yang sama dapat kita katakan dengan sebuah rumus kecil. Pada setiap saat t model berada dalam suatu keadaan S_t — nilai semua *field*-nya. Idealnya, sebuah view tidak lain adalah fungsi R (dari *render*, menggambar) dari keadaan model ke apa yang tampak di layar:

$$V_t = R(S_t) \tag{11.1}$$

Persamaan (11.1) adalah tujuan rancangan, dan tujuan yang berguna: gambar di layar *dihitung dari* model dan tidak menyimpan informasi miliknya sendiri. Dua view adalah dua fungsi, R_1 dan R_2, dari S_t yang sama, sehingga keduanya tidak mungkin saling bertentangan; dan pengujian model tidak memerlukan view sama sekali, karena ia memeriksa S_t secara langsung. Ketika sebuah program melanggar Persamaan (11.1) — ketika sebuah kolom teks memegang nilai yang tidak ada di tempat lain — Anda telah menemukan tempat di mana kelak dua layar akan berselisih.

11.3.2 Siapa boleh mengenal siapa

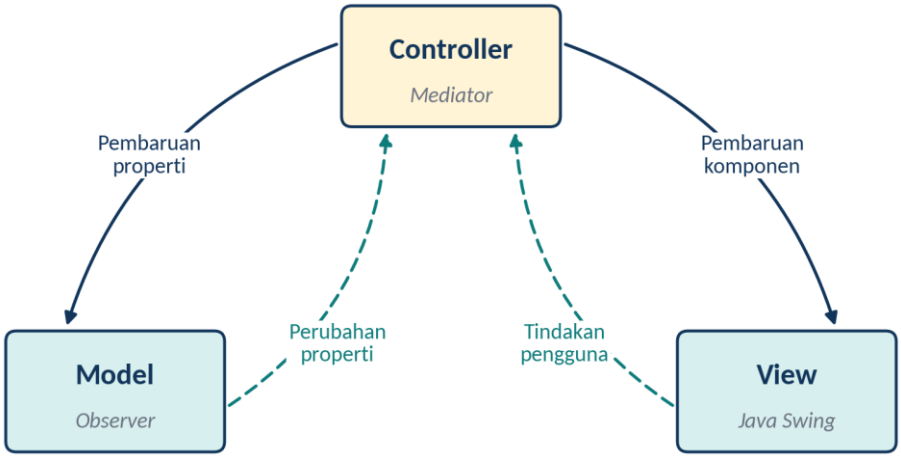
Ketiga peran itu hanya berguna bila garis-garis di antara mereka dihormati. **Tabel 11.1** merangkum aturannya. "Mengenal" di sini berarti "memegang referensi ke, dan memanggil metode dari"; di Java hal itu tampak sebagai `import` atau tipe sebuah *field*, di Python sebagai `import` atau atribut.

Tabel 11.1 Tanggung jawab dan ketergantungan yang diizinkan bagi ketiga peran.

Peran	Memegang	Boleh memanggil	Tidak boleh tahu tentang
Model	data dan aturan bisnis	tidak ada apa pun di luar dirinya, kecuali pendengar yang diberikan kepadanya	view, controller, *toolkit* GUI
View	*widget*, dan cara menggambar model	model (pertanyaan hanya-baca) atau controller	cara data dihitung atau disimpan
Controller	tidak memegang data sendiri, kecuali rangkaian sambungan	model (perintah) dan view (pembaruan)	cara view menggambar, cara model menyimpan

Sel terpenting dari **Tabel 11.1** adalah sel terakhir di baris pertama: **model tidak boleh mengenal *toolkit* GUI**. Di Java, kelas model tidak boleh mengimpor apa pun dari `javax.swing` atau `java.awt`. Di Python, modul model tidak boleh mengimpor `tkinter`. Aturan yang satu ini mudah diperiksa (cari impor-impor itu di dalam berkasnya) dan sekaligus menjamin tiga hal baik: model dapat diuji tanpa layar, dapat dipakai ulang di bawah antarmuka pengguna yang lain, dan tidak mungkin secara tak sengaja bergantung pada urutan penggambaran ulang *widget*.

Aturan untuk view dan controller lebih lentur, dan program yang berbeda menariknya di tempat yang berbeda. Dalam susunan klasik **Gambar 11.2**, view memegang referensi ke model dan menanyainya langsung ketika diberi tahu. Dalam susunan lain yang juga umum, diperlihatkan di **Gambar 11.5**, controller duduk di tengah sebagai **mediator** (perantara): peristiwa perubahan properti dari model menuju controller, dan controller memperbarui komponen-komponen view; view melaporkan tindakan pengguna kepada controller; model dan view tidak pernah bertemu. Label-label gambar itu menyebut bahan-bahan yang dipakai di Java: model diamati melalui perubahan properti, dan view dibangun dengan Swing.



Gambar 11.5 Controller sebagai mediator. Tindakan pengguna bergerak dari view ke controller, yang memperbarui properti model; perubahan properti model bergerak kembali ke controller, yang memperbarui komponen view. Model dan view tidak pernah saling merujuk.

Kedua susunan itu sama-sama MVC, dan bab ini memakai keduanya: papan tombol ponsel di Subbab 11.7 membiarkan view-view-nya mengamati model secara langsung, sedangkan meja

depan binatu di Subbab 11.9 menyalurkan segalanya melalui controller perantara, yang membuat controller sedikit lebih besar dan view sama sekali tidak mengenal model.

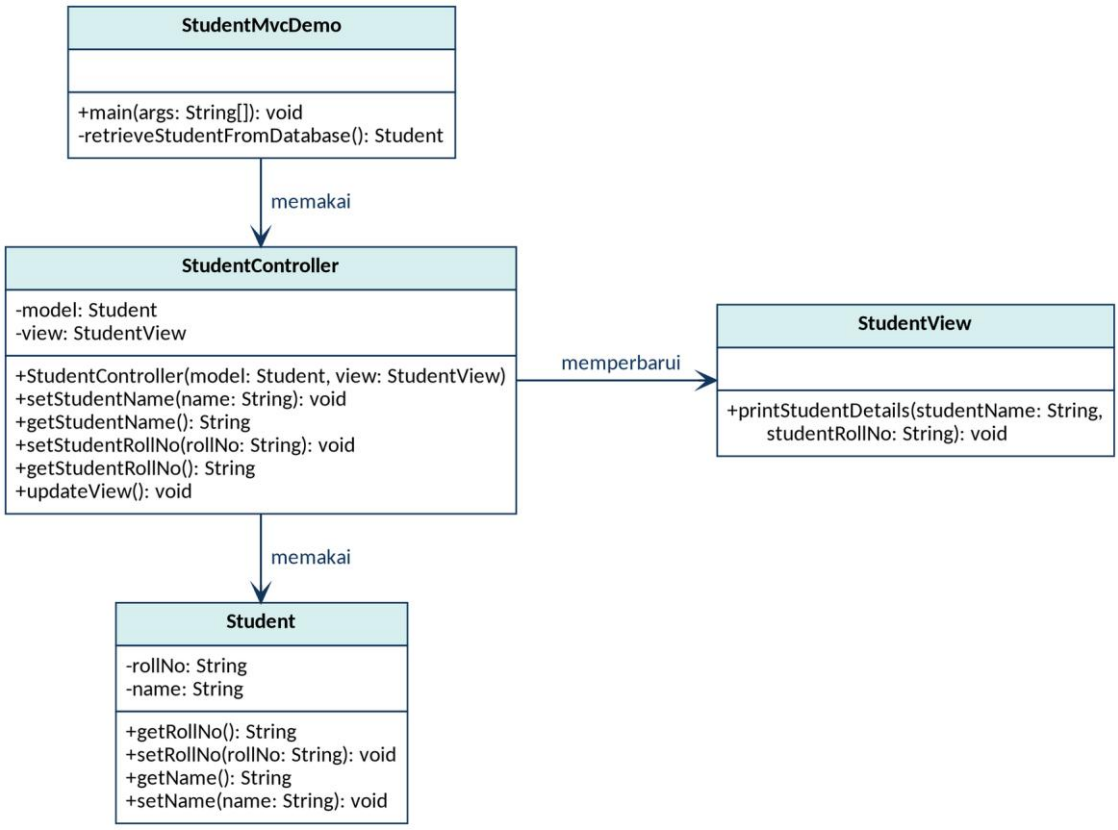
11.3.3 Asal-usul gagasan ini

MVC dideskripsikan pada tahun 1979 oleh Trygve Reenskaug, ilmuwan komputer Norwegia yang bekerja di Xerox PARC pada Smalltalk, bahasa tempat jendela, menu, dan tetikus pertama kali dipadukan dengan cara yang kini kita anggap lumrah [2]. Masalahnya sama dengan masalah kisah kita: pengguna ingin melihat dan memanipulasi informasi yang sama dengan beberapa cara sekaligus, dan program harus menjaga agar semua cara itu tetap konsisten. Catatannya menamai bagian-bagian itu *model*, *view*, dan *controller*, dan gagasan itu ditanamkan ke dalam pustaka kelas Smalltalk-80, yang kemudian diuraikan dalam artikel "buku resep" yang terkenal [3]. Pola Observer yang menghubungkan model dengan view adalah salah satu pola yang didata pada tahun 1994 dalam *Design Patterns* karya Gamma, Helm, Johnson, dan Vlissides [4], yang juga menguraikan MVC sebagai gabungan Observer dengan dua pola lain.

Dalam rancangan Smalltalk yang asli, controller menangani tetikus dan papan ketik secara mentah; Swing dan Tkinter sudah mengubah masukan mentah menjadi klik tombol, sehingga controller masa kini menerima "tombol *Add order* diklik" alih-alih "tetikus ditekan di piksel (412, 37)". Swing sendiri dibangun di atas varian MVC: setiap `JTable` memiliki `TableModel` terpisah, setiap `JList` sebuah `ListModel`, dan setiap `JButton` sebuah `ButtonModel`, sementara view dan controller setiap *widget* digabung menjadi satu "UI delegate" [5]. Anda sudah memakainya di Bab 10 setiap kali mengisi `DefaultTableModel` alih-alih tabelnya sendiri.

11.4 Program MVC Pertama: Data Mahasiswa

Sebelum berjumpa dengan jendela, peristiwa, dan pengamat, mari kita lihat ketiga peran itu dalam program terkecil yang dapat memuatnya. Tidak ada GUI sama sekali di dalamnya, dan itu sendiri sebuah pelajaran: MVC berbicara tentang *tanggung jawab*, bukan tentang Swing. Contohnya sudah terkenal [6]: data seorang mahasiswa dengan nama dan nomor urut (*roll number*, nomor mahasiswa di daftar kelas), sebuah view yang mencetak data itu ke konsol, dan sebuah controller yang menghubungkan keduanya. **Gambar 11.6** adalah diagram kelasnya.



Gambar 11.6 Diagram kelas contoh mahasiswa. `StudentController` memegang sebuah model `Student` dan sebuah `StudentView`; ia memperbarui model melalui *setter* dan memperbarui view dengan `updateView()`; kelas demo `StudentMvcDemo` memakai controller dan mengambil data pertama dari basis data (tiruan).

Bacalah diagram itu dari bawah. `Student`, modelnya, memiliki dua *field* privat, `rollNo` dan `name`, masing-masing dengan *getter* dan *setter*; tidak ada yang lain. `StudentView` memiliki satu metode, `printStudentDetails`, yang mencetak apa yang diberikan kepadanya. `StudentController` memegang satu model dan satu view (tanda `-` di depan `model` dan `view` berarti *private*); ia meneruskan `setStudentName` dan *setter* lainnya ke model, dan metodenya `updateView` menanyakan nilai-nilai model lalu menyerahkannya kepada view. Panah-panah menyatakan siapa *memakai* siapa: demo memakai controller, controller memakai model dan memperbarui view. Tidak ada panah dari `Student` ke mana pun. Model tidak mengenal siapa-siapa.

Kode 11.1 adalah programnya dalam Java, satu berkas berisi kelas publik `StudentMvcDemo` dan tiga kelas *package-private*.

Kode 11.1 `StudentMvcDemo.java`: program MVC terkecil. `Student` adalah model, `StudentView` view, dan `StudentController` controller.

```
public class StudentMvcDemo {
    public static void main(String[] args) {
        // Ambil data mahasiswa dari "basis data".
        Student model = retrieveStudentFromDatabase();
        // View yang menuliskan data mahasiswa ke konsol.
        StudentView view = new StudentView();
        StudentController controller = new StudentController(model, view);
        controller.updateView();
        // Ubah model melalui controller, lalu tampilkan lagi.
        controller.setStudentName("John");
        controller.updateView();
    }

    private static Student retrieveStudentFromDatabase() {
        Student student = new Student();
        student.setName("Robert");
        student.setRollNo("10");
        return student;
    }
}

// Model: data beserta aturan untuk membaca dan mengubahnya.
class Student {
    private String rollNo;
    private String name;

    public String getRollNo() {
        return rollNo;
    }

    public void setRollNo(String rollNo) {
        this.rollNo = rollNo;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}
```

Kode 11.1 (Lanjutan) *StudentMvcDemo.java: program MVC terkecil. Student adalah model, StudentView view, dan StudentController controller.*

```
// View: tahu cara menampilkan mahasiswa, dan tidak lebih.
class StudentView {
    public void printStudentDetails(String studentName, String studentRollNo) {
        System.out.println("Student: ");
        System.out.println("Name: " + studentName);
        System.out.println("Roll No: " + studentRollNo);
    }
}

// Controller: satu-satunya kelas yang mengenal model sekaligus view.
class StudentController {
    private final Student model;
    private final StudentView view;

    public StudentController(Student model, StudentView view) {
        this.model = model;
        this.view = view;
    }

    public void setStudentName(String name) {
        model.setName(name);
    }

    public String getStudentName() {
        return model.getName();
    }

    public void setStudentRollNo(String rollNo) {
        model.setRollNo(rollNo);
    }

    public String getStudentRollNo() {
        return model.getRollNo();
    }

    public void updateView() {
        view.printStudentDetails(model.getName(), model.getRollNo());
    }
}
```

Kompilasi dan jalankan di folder `java` bab ini dengan `javac StudentMvcDemo.java` dan `java StudentMvcDemo`:

```
Student:
Name: Robert
Roll No: 10
Student:
Name: John
Roll No: 10
```

Telusurilah `main` baris demi baris, karena setiap program MVC memiliki empat langkah yang sama dengan urutan yang sama.

- **Bangun model.** `retrieveStudentFromDatabase()` mewakili pembacaan data dari basis data; ia membuat sebuah `Student`, mengatur namanya menjadi `"Robert"` dan nomor urutnya menjadi `"10"`, lalu mengembalikannya. Dalam program sungguhan ini akan berupa *query*; intinya, model dibuat *lebih dulu*, dan terlepas dari view mana pun.
- **Bangun view.** `new StudentView()` membuat objek yang keahliannya hanya mencetak. Ia tidak memegang data mahasiswa.

- **Bangun controller, dan berikan keduanya kepadanya.** `new StudentController(model, view)` adalah saat ketiganya bertemu. Konstruktor menyimpan kedua referensi itu di *field* `final`, sehingga sambungannya tidak dapat diubah sesudahnya.
- **Biarkan controller mengemudi.** `controller.updateView()` membaca model dan menyuruh view mencetak; hasilnya tiga baris pertama keluaran. `controller.setStudentName("John")` mengubah model — dan tidak ada yang lain: tidak ada baris yang tercetak, karena tidak ada yang meminta view menggambar ulang. `updateView()` yang kedua mencetak keadaan baru, yaitu tiga baris terakhir.

Langkah yang sunyi di tengah itu patut diperhatikan. Setelah `setStudentName("John")`, model berkata *John* sementara hal terakhir yang tercetak berkata *Robert*. Untuk sesaat view ketinggalan zaman, dan ia tetap ketinggalan sampai ada yang ingat memanggil `updateView()`. Dalam program sekecil ini `main` ingat. Dalam aplikasi sungguhan, dengan puluhan tempat yang mengubah model, cepat atau lambat ada yang lupa. Pola Observer di subbab berikutnya menghapus kebutuhan untuk ingat: model sendiri akan mengumumkan setiap perubahan.

(Satu catatan kecil tentang keluaran: baris pertama setiap blok adalah `Student:` dengan spasi setelah titik dua, karena `println("Student: ")` aslinya memuat spasi itu. Spasi tak terlihat di ujung baris adalah penyebab klasik gagalnya pembandingan keluaran; buku ini menyalin keluaran program dengan persis.)

Kode 11.2 adalah program yang sama dalam Python. Modelnya memakai *property* (Bab 7) alih-alih metode *getter* dan *setter*, sehingga controller menulis `self._model.name = name` di tempat Java menulis `model.setName(name)`.

Kode 11.2 `student_mvc_demo.py`: data mahasiswa dalam Python, dengan *field* model disajikan sebagai *property*.

```
"""student_mvc_demo.py: program MVC terkecil, data mahasiswa di konsol."""


class Student:
    """Model: data beserta aturan untuk membaca dan mengubahnya."""

    def __init__(self, name="", roll_no=""):
        self._name = name
        self._roll_no = roll_no

    @property
    def name(self):
        return self._name

    @name.setter
    def name(self, value):
        self._name = value

    @property
    def roll_no(self):
        return self._roll_no

    @roll_no.setter
    def roll_no(self, value):
        self._roll_no = value


class StudentView:
    """View: tahu cara menampilkan mahasiswa, dan tidak lebih."""

    def print_student_details(self, student_name, student_roll_no):
        print("Student: ")
        print("Name: " + student_name)
        print("Roll No: " + student_roll_no)


class StudentController:
    """Controller: satu-satunya kelas yang mengenal model sekaligus view."""

    def __init__(self, model, view):
        self._model = model
        self._view = view

    def set_student_name(self, name):
        self._model.name = name
```

Kode 11.2 (Lanjutan) *student_mvc_demo.py: data mahasiswa dalam Python, dengan field model disajikan sebagai property.*

```
    def get_student_name(self):
        return self._model.name

    def set_student_roll_no(self, roll_no):
        self._model.roll_no = roll_no

    def get_student_roll_no(self):
        return self._model.roll_no

    def update_view(self):
        self._view.print_student_details(self._model.name, self._model.roll_no)


def retrieve_student_from_database():
    return Student(name="Robert", roll_no="10")


def main():
    # Ambil data mahasiswa dari "basis data".
    model = retrieve_student_from_database()
    # View yang menuliskan data mahasiswa ke konsol.
    view = StudentView()
    controller = StudentController(model, view)
    controller.update_view()
    # Ubah model melalui controller, lalu tampilkan lagi.
    controller.set_student_name("John")
    controller.update_view()


if __name__ == "__main__":
    main()
```

Jalankan dengan `python3 student_mvc_demo.py` (di Windows, `python student_mvc_demo.py` atau `py student_mvc_demo.py`). Keluarannya sama persis dengan keluaran Java:

```
Student:
Name: Robert
Roll No: 10
Student:
Name: John
Roll No: 10
```

`private String name` milik Java beserta `getName` dan `setName` menjadi `self._name` di Python dengan `@property` dan `@name.setter`; garis bawah di depan adalah konvensi Python untuk "privat" (Subbab 7.13). `retrieve_student_from_database` adalah fungsi biasa, karena Python tidak mengharuskan setiap fungsi tinggal di dalam kelas.

Tiga kelas untuk mencetak enam baris memang lebih dari yang diperlukan sebuah `print`; nilainya muncul begitu program bertambah besar. Soal Latihan 11.2 meminta Anda menambahkan view kedua — satu baris nilai yang dipisahkan koma, siap untuk lembar kerja — dan Anda akan mendapati bahwa Anda sedikit mengubah controller, menambah satu kelas, dan sama sekali tidak menyentuh model. Model seorang mahasiswa tidak peduli ada berapa cara untuk mencetaknya.

11.5 Pola Observer di Bawah MVC

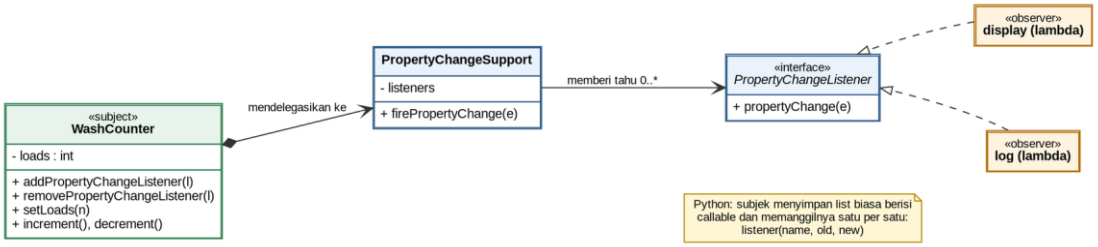
Program mahasiswa memiliki kelemahan yang sudah kita catat: setelah model berubah, view tetap basi sampai seseorang memanggil `updateView()`. Pola Observer menyembuhkannya dengan membalik hubungan itu. Alih-alih ada yang *menyuruh view* menggambar ulang, model

mengumumkan bahwa dirinya telah berubah, dan setiap pihak yang berminat — setiap view, sebuah log, layar kedua — bereaksi.

> **DEFINISI.** Dalam pola **Observer**, sebuah objek yang disebut **subjek** (atau *observable*) menyimpan daftar **pengamat** (*observer*, juga disebut *listener* atau pelanggan). Pengamat mendaftar pada subjek dengan memanggil metode seperti `addListener`, dan kelak boleh berhenti mendaftar. Setiap kali keadaan subjek berubah, subjek **memberi tahu** setiap pengamat yang terdaftar, biasanya dengan memanggil satu metode yang telah disepakati pada masing-masing. Subjek hanya tahu bahwa pengamatnya memiliki metode itu; ia tidak tahu kelas mereka atau apa yang mereka kerjakan [4].

Pola ini ada di mana-mana begitu Anda mencarinya. Sebuah `JButton` di Swing adalah subjek yang pengamatnya `ActionListener`; sebuah `StringVar` di Tkinter adalah subjek yang pengamatnya didaftarkan dengan `trace_add`; sebuah buletin berlangganan adalah subjek yang pengamatnya para pelanggannya. Bagian kontrak milik subjek kecil saja: simpan daftar, sediakan *tambah* dan *hapus*, dan panggil semua anggota daftar ketika sesuatu berubah. Bagian milik pengamat lebih kecil lagi: miliki metode yang akan dipanggil subjek.

Gambar 11.7 memperlihatkan pola itu sebagaimana akan kita tulis di Java. Subjeknya adalah `WashCounter`, yang mencacah berapa kali mesin cuci binatu dijalankan hari ini. Ia tidak mengelola pengamatnya sendiri; ia *mendelegasikan* tugas itu kepada objek pembantu dari pustaka standar, `java.beans.PropertyChangeSupport`, yang menyimpan daftar dan melakukan pemberitahuan. Para pengamat mengimplementasikan *interface* bermetode tunggal `PropertyChangeListener`; pengamat kita adalah dua lambda, sebuah tampilan (*display*) dan sebuah log.



Gambar 11.7 Pola Observer sebagaimana dipakai `ObserverDemo.java`. Subjek `WashCounter` mendelegasikan daftar pendengarnya kepada sebuah `PropertyChangeSupport`, yang memberi tahu sebanyak apa pun objek yang mengimplementasikan `PropertyChangeListener`; kedua pengamatnya adalah lambda. Di Python peran yang sama dimainkan oleh list biasa berisi *callable*.

11.5.1 Java: dari `Observable` ke `PropertyChangeSupport`

Java telah menawarkan dua implementasi siap pakai untuk sisi subjek. Yang lebih tua, kelas `java.util.Observable` beserta *interface* pasangannya `java.util.Observer`, berasal dari Java 1.0 dan dijumpai di banyak buku teks dan contoh perkuliahan yang lebih lama: sebuah model `extends Observable`, memanggil `setChanged()` dan `notifyObservers(value)` setelah setiap perubahan, dan sebuah view `implements Observer` serta menerima `update(Observable o, Object arg)`. Sejak Java 9 keduanya **usang** (*deprecated*) [7]. Alasannya patut dipelajari. `Observable` adalah sebuah *kelas*, sehingga model yang mewarisinya tidak dapat mewarisi kelas lain (Java hanya mengenal pewarisan tunggal, Bab 7). Metode `update`-nya mengirim nilai yang berubah sebagai `Object` telanjang, sehingga setiap pengamat harus melakukan *cast* sambil berharap. Ia tidak menyebut properti *mana* yang berubah, sehingga pengamat sebuah model dengan sepuluh *field* tidak dapat mengetahui mana yang bergerak. Dan urutan pemberitahuan serta keamanan *thread*-nya tidak pernah dispesifikasikan.

Pengganti yang dianjurkan adalah `java.beans.PropertyChangeSupport`, dan ia memperbaiki setiap masalah tersebut. Ia dipakai melalui *komposisi* — model memegangnya sebagai *field* — sehingga model bebas mewarisi kelas apa pun. Setiap pemberitahuan berupa `PropertyChangeEvent` yang membawa objek sumber, *nama* properti yang berubah, nilai lamanya, dan nilai barunya. Dan ia memiliki aturan bawaan yang menolong: bila nilai lama dan nilai baru sama, tidak ada peristiwa yang ditembakkan sama sekali. **Kode 11.3** menerapkannya.

Kode 11.3 `ObserverDemo.java`: sebuah subjek, `WashCounter`, yang mengumumkan setiap perubahan properti `loads`-nya melalui `PropertyChangeSupport`, dan dua pengamat yang tidak saling mengenal.

```
import java.beans.PropertyChangeEvent;
import java.beans.PropertyChangeListener;
import java.beans.PropertyChangeSupport;

public class ObserverDemo {
    public static void main(String[] args) {
        WashCounter counter = new WashCounter();

        // Dua pengamat. Tidak satu pun tahu bahwa yang lain ada.
        PropertyChangeListener display =
                e -> System.out.println("  [display] loads today: " + e.getNewValue());
        PropertyChangeListener log =
                e -> System.out.println("  [log] " + e.getPropertyName() + ": "
                        + e.getOldValue() + " -> " + e.getNewValue());

        counter.addPropertyChangeListener(display);
        counter.addPropertyChangeListener(log);

        System.out.println("increment()");
        counter.increment();
        System.out.println("increment()");
        counter.increment();
        System.out.println("setLoads(2)");
        counter.setLoads(2);  // nilai sama: tidak ada peristiwa yang ditembakkan
        counter.removePropertyChangeListener(log);
        System.out.println("decrement() after the log unsubscribed");
        counter.decrement();
        System.out.println("listeners now: " + counter.listenerCount());
        try {
            counter.setLoads(-1);
        } catch (IllegalArgumentException e) {
            System.out.println("setLoads(-1): " + e.getMessage());
        }
    }
}

// Subjek: ia menyimpan datanya dan mengumumkan setiap perubahan.
class WashCounter {
    private final PropertyChangeSupport support = new PropertyChangeSupport(this);
    private int loads;

    void addPropertyChangeListener(PropertyChangeListener listener) {
        support.addPropertyChangeListener(listener);
    }
```

Kode 11.3 (Lanjutan) *ObserverDemo.java: sebuah subjek, WashCounter, yang mengumumkan setiap perubahan properti loads-nya melalui PropertyChangeSupport, dan dua pengamat yang tidak saling mengenal.*

```
    void removePropertyChangeListener(PropertyChangeListener listener) {
        support.removePropertyChangeListener(listener);
    }

    int listenerCount() {
        return support.getPropertyChangeListeners().length;
    }

    int getLoads() {
        return loads;
    }

    void setLoads(int newLoads) {
        if (newLoads < 0) {
            throw new IllegalArgumentException("loads cannot be negative");
        }
        int oldLoads = loads;
        loads = newLoads;
        // Hanya menembakkan peristiwa bila nilai lama dan baru berbeda.
        support.firePropertyChange(
                new PropertyChangeEvent(this, "loads", oldLoads, newLoads));
    }

    void increment() {
        setLoads(loads + 1);
    }

    void decrement() {
        setLoads(loads - 1);
    }
}
```

Menjalankan `java ObserverDemo` mencetak:

```
increment()
  [display] loads today: 1
  [log] loads: 0 -> 1
increment()
  [display] loads today: 2
  [log] loads: 1 -> 2
setLoads(2)
decrement() after the log unsubscribed
  [display] loads today: 1
listeners now: 1
setLoads(-1): loads cannot be negative
```

Cocokkan keluaran dengan kode, peristiwa demi peristiwa.

- `increment()` yang pertama memanggil `setLoads(1)`. Metode itu memeriksa bahwa nilai baru tidak negatif, mengingat nilai lama 0, menyimpan nilai baru 1, dan memanggil `support.firePropertyChange(...)` dengan sebuah peristiwa yang berkata "*loads* berubah dari 0 menjadi 1". `PropertyChangeSupport` memanggil setiap pendengar terdaftar sesuai urutan pendaftarannya: mula-mula `display`, yang mencetak `[display] loads today: 1`, lalu `log`, yang mencetak `[log] loads: 0 -> 1`. Pencacah itu sendiri tidak mencetak apa-apa. Ia tidak tahu bahwa ada *display* atau log.
- `setLoads(2)` adalah yang menarik: ia tidak mencetak *apa pun* setelah judulnya. Pencacah memang memanggil `firePropertyChange`, tetapi nilai lama dan baru sama-sama 2, dan `PropertyChangeSupport` tidak merepotkan pendengarnya dengan perubahan yang bukan

perubahan. Aturan ini menghemat kerja dan — lebih penting lagi — mencegah satu keluarga *bug* utuh, yaitu dua objek yang terus-menerus saling memberi tahu "perubahan" tanpa akhir.

- `removePropertyChangeListener(log)` menghentikan langganan log, sehingga `decrement()` berikutnya hanya sampai ke *display*. Daftar subjek kini berisi satu pendengar, sebagaimana dikonfirmasi `listenerCount()`.
- `setLoads(-1)` melanggar aturan model, dan model menolaknya dengan `IllegalArgumentException` *sebelum* mengubah apa pun atau memberi tahu siapa pun. Model yang memberi tahu lebih dulu dan memvalidasi belakangan akan mengabarkan kepada setiap view sebuah nilai yang sebentar lagi ditolak.

Perhatikan bahwa `increment` dan `decrement` sama-sama melewati `setLoads`, sehingga hanya ada tepat satu tempat nilai berubah dan pengamat diberi tahu; jadikan itu kebiasaan. Dan karena `PropertyChangeListener` hanya memiliki satu metode abstrak, seorang pengamat dapat berupa lambda, `e -> ...`, persis seperti `ActionListener` di Bab 10.

11.5.2 Python: pengamat dalam selusin baris

Pustaka standar Python tidak memiliki kelas *observable* serbaguna untuk objek biasa — variabel Tkinter memilikinya, tetapi memerlukan Tkinter — dan memang tidak memerlukannya, karena pola ini hanya selusin baris Python biasa. Fungsi adalah objek, sehingga subjek dapat menyimpan list berisi *fungsi* dan memanggil masing-masing. **Kode 11.4** adalah pencacah yang sama.

Kode 11.4 `observer_demo.py`: pola Observer yang ditulis sendiri. Subjek menyimpan list berisi *callable* dan memanggil masing-masing dengan nama properti serta nilai lama dan barunya.

```
"""observer_demo.py: pola Observer yang ditulis sendiri dalam Python."""


class WashCounter:
    """Subjek: ia menyimpan datanya dan mengumumkan setiap perubahan."""

    def __init__(self):
        self._loads = 0
        self._listeners = []

    def add_listener(self, listener):
        self._listeners.append(listener)

    def remove_listener(self, listener):
        self._listeners.remove(listener)

    def listener_count(self):
        return len(self._listeners)

    @property
    def loads(self):
        return self._loads

    def set_loads(self, new_loads):
        if new_loads < 0:
            raise ValueError("loads cannot be negative")
        old_loads = self._loads
        self._loads = new_loads
        if old_loads != new_loads:  # umumkan perubahan sungguhan saja
            self._fire("loads", old_loads, new_loads)

    def increment(self):
        self.set_loads(self._loads + 1)

    def decrement(self):
        self.set_loads(self._loads - 1)

    def _fire(self, name, old, new):
        # Telusuri salinannya: pendengar bisa berhenti berlangganan saat sedang diberi
tahu.
        for listener in list(self._listeners):
            listener(name, old, new)


def main():
    counter = WashCounter()
```

Kode 11.4 (Lanjutan) *observer_demo.py: pola Observer yang ditulis sendiri. Subjek menyimpan list berisi callable dan memanggil masing-masing dengan nama properti serta nilai lama dan barunya.*

```
    # Dua pengamat. Tidak satu pun tahu bahwa yang lain ada.
    def display(name, old, new):
        print(f"  [display] loads today: {new}")

    def log(name, old, new):
        print(f"  [log] {name}: {old} -> {new}")

    counter.add_listener(display)
    counter.add_listener(log)

    print("increment()")
    counter.increment()
    print("increment()")
    counter.increment()
    print("set_loads(2)")
    counter.set_loads(2)  # nilai sama: tidak ada peristiwa yang ditembakkan
    counter.remove_listener(log)
    print("decrement() after the log unsubscribed")
    counter.decrement()
    print("listeners now:", counter.listener_count())
    try:
        counter.set_loads(-1)
    except ValueError as e:
        print("set_loads(-1):", e)


if __name__ == "__main__":
    main()
```

Jalankan dengan `python3 observer_demo.py`:

```
increment()
  [display] loads today: 1
  [log] loads: 0 -> 1
increment()
  [display] loads today: 2
  [log] loads: 1 -> 2
set_loads(2)
decrement() after the log unsubscribed
  [display] loads today: 1
listeners now: 1
set_loads(-1): loads cannot be negative
```

Keluarannya sama baris demi baris dengan keluaran Java, kecuali nama metode `set_loads`, karena kelas Python mengimplementasikan kontrak yang sama secara manual. `add_listener` dan `remove_listener` mengelola list biasa. `set_loads` memvalidasi, menyimpan, dan memanggil `_fire` hanya bila nilainya benar-benar berubah — aturan yang diberikan `PropertyChangeSupport` kepada Java secara cuma-cuma di sini menjadi satu baris `if old_loads != new_loads:`. Dan `_fire` menelusuri `list(self._listeners)`, sebuah *salinan* list, bukan list itu sendiri. Komentarnya menjelaskan alasannya: seorang pendengar dapat memutuskan, justru saat sedang diberi tahu, untuk berhenti berlangganan (bayangkan peringatan sekali pakai), dan menghapus elemen dari list ketika sebuah gelung `for` sedang menelusurinya akan diam-diam melompati elemen berikutnya. Soal Latihan 11.3 memperlihatkan hal itu terjadi di kedua bahasa; salinan itu hanya berharga beberapa nanodetik dan menghapus bahayanya. (`PropertyChangeSupport` melindungi dirinya dengan cara yang sama di dalamnya, satu alasan lagi untuk memakainya di Java alih-alih `ArrayList` buatan sendiri.)

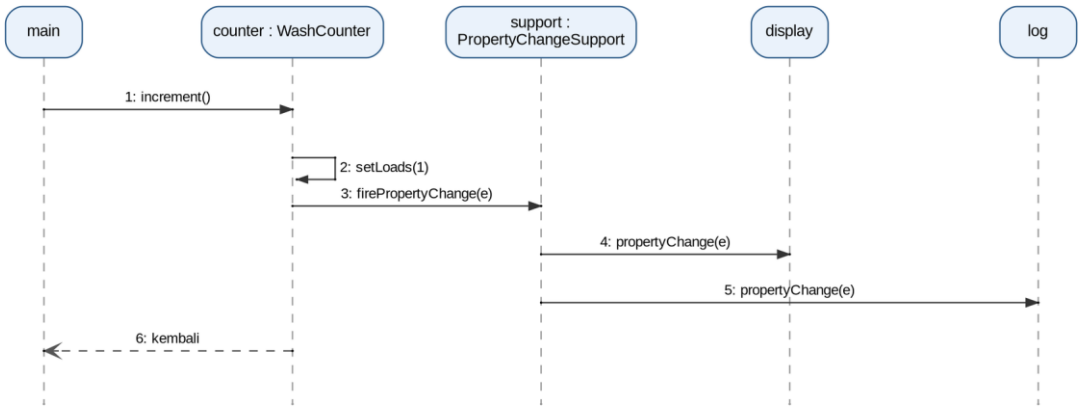
Tabel 11.2 menyandingkan kedua implementasi itu.

Tabel 11.2 Pola Observer dalam kedua bahasa.

Bahan	Java (`ObserverDemo.java`)	Python (`observer_demo.py`)
Tipe pengamat	*interface* `PropertyChangeListener`	*callable* apa pun: fungsi, lambda, metode terikat
Daftar pengamat	di dalam `PropertyChangeSupport`	`self._listeners`, sebuah `list` biasa
Berlangganan / berhenti	`addPropertyChangeListener`, `removePropertyChangeListener`	`add_listener`, `remove_listener`
Pemberitahuan	`firePropertyChange(event)`	`_fire(name, old, new)`
Yang diterima pengamat	sebuah `PropertyChangeEvent`: sumber, nama, lama, baru	tiga argumen `name, old, new`
Tanpa peristiwa bila tak ada yang berubah	bawaan	`if old_loads != new_loads:`
Aman bila pendengar berhenti berlangganan sendiri	bawaan	telusuri `list(self._listeners)`

11.5.3 Satu perubahan, banyak panggilan

Gambar 11.8 menggambar `increment()` pertama dari **Kode 11.3** sebagai **diagram sekuens** (*sequence diagram*). Diagram sekuens adalah gambar UML tentang *waktu*: setiap objek yang terlibat memiliki garis hidup (*lifeline*) tegak putus-putus, waktu mengalir ke bawah, dan setiap panah mendatar adalah satu pesan — satu pemanggilan metode — yang diberi nomor sesuai urutan terjadinya. Lengkungan yang kembali ke garis hidupnya sendiri adalah panggilan sebuah objek kepada dirinya sendiri; panah putus-putus adalah nilai kembali.



Gambar 11.8 Diagram sekuens satu `increment()`. Pencacah memanggil `setLoads(1)` miliknya sendiri, yang meminta objek pendukung menembakkan satu peristiwa; objek pendukung memanggil setiap pendengar sesuai urutan pendaftaran, dan baru setelah itu `increment()` kembali ke `main`.

Dua fakta dapat dibaca dari diagram itu. Pertama, pemberitahuan bersifat **sinkron**: pesan 4 dan 5 terjadi *di dalam* pesan 1. Panggilan `main` terhadap `increment()` tidak kembali sebelum setiap pengamat selesai bereaksi. Bila sebuah pengamat lambat — bila ia menggambar ulang tabel besar, atau menulis ke disk — setiap perubahan model ikut menjadi lambat. Kedua, jumlah panggilan bertambah seiring jumlah pengamat. Bila k pengamat terdaftar dan model berubah m kali, para pengamat dipanggil sebanyak

$$C = m \cdot k \tag{11.2}$$

kali seluruhnya. Persamaan (11.2) menjelaskan satu aturan rancangan setiap program MVC: *sebuah model sebaiknya memberi tahu sekali untuk setiap perubahan logis*, bukan sekali untuk setiap *field* yang disentuhnya. Bila menambah pesanan mengubah daftar pesanan, total berjalan, dan jumlah pesanan yang menunggu, lalu model menembakkan satu peristiwa untuk masing-masing, setiap view menggambar ulang tiga kali untuk satu klik — *m* menjadi tiga kali lipat tanpa guna. Karena itu model binatu di Subbab 11.9 menembakkan tepat satu peristiwa, `orderBookChanged`, per operasi, dan setiap view menghitung ulang semua yang ditampilkannya dari keadaan model saat itu.

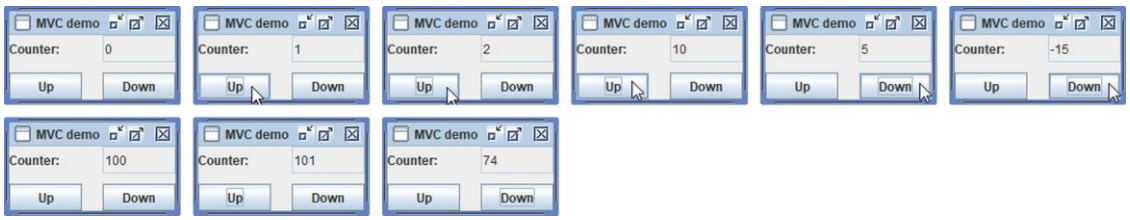
11.5.4 Didorong atau ditarik?

Kedua program di atas *mendorong* (*push*) data kepada pengamatnya: setiap peristiwa membawa nilai lama dan nilai baru, sehingga pengamat dapat bereaksi tanpa menanyakan apa pun kepada subjek. Alternatifnya adalah *menarik* (*pull*): peristiwa hanya berkata "aku berubah", lalu setiap pengamat menanyakan kepada subjek apa pun yang ia butuhkan. Contoh pencacah asli yang dibangun ulang di Subbab 11.6 memakai kedua gaya itu: `setValue`-nya memanggil `notifyObservers()` tanpa isi, sedangkan `incValue` memanggil `notifyObservers(value)` dengan nilai barunya.

Tak satu pun gaya selalu lebih baik. Mendorong nyaman ketika pengamat membutuhkan tepat nilai yang berubah, seperti *display* kita. Menarik lebih sederhana ketika modelnya besar dan view-view yang berbeda memerlukan bagian yang berbeda: model menembakkan satu peristiwa tanpa isi, dan setiap view membaca apa yang ia perlukan — view daftar membaca pesanan, view ringkasan membaca total. Menarik juga tidak mungkin menyampaikan informasi basi, karena setiap view membaca keadaan model *saat ini*, setelah seluruh perubahan selesai. Papan tombol dan meja binatu di bagian selanjutnya bab ini menarik; pencacah di subbab berikutnya mendorong satu bilangan bulat.

11.6 Pencacah, Dibangun Ulang: MVC dengan Jendela

Kini kita memiliki semua bagian untuk membangun program MVC sungguhan yang berjendela. Yang pertama sengaja dibuat mungil: sebuah pencacah dengan dua tombol, **Up** dan **Down**. Inilah contoh pertama klasik MVC berjendela, dan **Gambar 11.9** memperlihatkan versi aslinya sedang bekerja. Di baris atas program dimulai dari 0; mengeklik **Up** menghasilkan 1, 2, lalu, setelah beberapa klik lagi, 10, dan mengeklik **Down** berulang kali membawanya ke 5 dan terus turun sampai −15, karena tidak ada yang melarang nilai negatif. Di baris bawah program yang sama dimulai dengan nilai awal 100, naik ke 101 dan turun ke 74.



Gambar 11.9 Pencacah Up/Down sedang bekerja. Baris atas, dimulai dari 0: setelah klik pada *Up* kolomnya menampilkan 1, 2, dan 10; setelah klik pada *Down* ia menampilkan 5 dan −15. Baris bawah, dimulai dari 100: 101, lalu 74. Penunjuk tetikus memperlihatkan tombol yang terakhir diklik.

Versi asli program ini memiliki empat kelas, dengan gaya zamannya: `Model extends Observable` menyimpan bilangan bulat; `MyPanel extends JPanel` membangun label, kolom teks hanya-baca, dan kedua tombol dalam `GridLayout` dua baris; `View implements Observer, ActionListener`

membuat *frame*, mendaftarkan dirinya sebagai pengamat model *dan* sebagai pendengar kedua tombol, lalu di dalam `actionPerformed` memanggil `model.incValue()` atau `model.decValue()` bergantung pada *action command* tombolnya; dan `Controller` memiliki metode `initView(title, value)` yang meminta view membangun jendela. `Main` membuat ketiganya dan memanggil `controller.initView("MVC demo", 0)`; mengganti 0 dengan 100 menghasilkan baris bawah **Gambar 11.9**.

Versi itu bekerja, tetapi perhatikan di mana tugas-tugasnya berakhir. Kelas bernama `View` juga menangani klik tombol dan memanggil model — ia controller sekaligus view. Kelas bernama `Controller` nyaris tidak mengerjakan apa-apa. Dan modelnya mewarisi kelas yang sudah usang. Versi yang kita bangun ulang mempertahankan jendelanya persis seperti semula dan menempatkan setiap tugas di tempat yang semestinya.

- `CounterModel` menyimpan bilangan bulat dan daftar pengamat. Ia tidak memakai kelas GUI apa pun.
- `CounterView` adalah sebuah ***interface***: tiga hal yang dibutuhkan controller dari view *mana pun* — tampilkan sebuah nilai, beri tahu aku ketika *Up* diklik, beri tahu aku ketika *Down* diklik.
- `SwingCounterView` mengimplementasikan *interface* itu dengan jendela sungguhan. `TextCounterView` mengimplementasikannya dengan mencetak ke konsol, untuk uji mandiri.
- `CounterController` menyambungkan model dengan view: *Up* berarti `increment`, *Down* berarti `decrement`, dan setiap perubahan model ditampilkan oleh view.

Kode 11.5 adalah program Java-nya.

Kode 11.5 `CounterMvc.java`: pencacah Up/Down sebagai model, *interface* view dengan implementasi Swing dan implementasi teks, serta controller. Dengan `--selftest` ia berjalan tanpa jendela.

```
import java.awt.GridLayout;
import java.util.ArrayList;
import java.util.List;
import java.util.function.IntConsumer;
import javax.swing.BorderFactory;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JTextField;
import javax.swing.SwingUtilities;

public class CounterMvc {
    public static void main(String[] args) {
        boolean selftest = false;
        int start = 0;
        for (String arg : args) {
            if (arg.equals("--selftest")) {
                selftest = true;
            } else {
                start = Integer.parseInt(arg);
            }
        }
        CounterModel model = new CounterModel(start);
        if (selftest) {
            selfTest(model);
            return;
        }
        // Objek Swing dibuat di event dispatch thread.
        SwingUtilities.invokeLater(() -> {
            SwingCounterView view = new SwingCounterView("MVC demo");
            new CounterController(model, view);
            view.open();
        });
    }
```

Kode 11.5 (Lanjutan) *CounterMvc.java: pencacah Up/Down sebagai model, interface view dengan implementasi Swing dan implementasi teks, serta controller. Dengan --selftest ia berjalan tanpa jendela.*

```
    // Model dan controller bekerja, dengan view teks sebagai ganti jendela.
    static void selfTest(CounterModel model) {
        TextCounterView view = new TextCounterView();
        new CounterController(model, view);
        int start = model.getValue();
        view.click("Up");
        view.click("Up");
        view.click("Down");
        view.click("Up");
        int expected = start + 2;
        if (model.getValue() != expected
                || !view.shownText().equals(Integer.toString(expected))) {
            throw new AssertionError("selftest failed");
        }
        System.out.println("selftest passed: model holds " + model.getValue()
                + ", view shows " + view.shownText());
    }
}

// Model: sebuah bilangan bulat dan aturan untuk mengubahnya. Tanpa Swing sama sekali.
class CounterModel {
    private final List<IntConsumer> listeners = new ArrayList<>();
    private int value;

    CounterModel(int value) {
        this.value = value;
    }

    int getValue() {
        return value;
    }

    void addListener(IntConsumer listener) {
        listeners.add(listener);
    }

    void increment() {
        setValue(value + 1);
    }

    void decrement() {
        setValue(value - 1);
    }
```

Kode 11.5 (Lanjutan) *CounterMvc.java: pencacah Up/Down sebagai model, interface view dengan implementasi Swing dan implementasi teks, serta controller. Dengan --selftest ia berjalan tanpa jendela.*

```
    void setValue(int newValue) {
        value = newValue;
        for (IntConsumer listener : listeners) {
            listener.accept(value);  // beri tahu setiap pengamat
        }
    }
}

// Apa yang dibutuhkan controller dari view mana pun, berjendela atau tidak.
interface CounterView {
    void showValue(int value);

    void onUp(Runnable action);

    void onDown(Runnable action);
}

// Controller: mengubah gestur pengguna menjadi operasi model dan merangkai bagian-
bagiannya.
class CounterController {
    CounterController(CounterModel model, CounterView view) {
        view.onUp(model::increment);
        view.onDown(model::decrement);
        model.addListener(view::showValue);
        view.showValue(model.getValue());  // gambar pertama
    }
}

// View sungguhan: sebuah jendela Swing kecil.
class SwingCounterView implements CounterView {
    private final JFrame frame;
    private final JTextField counter = new JTextField(6);
    private final JButton up = new JButton("Up");
    private final JButton down = new JButton("Down");

    SwingCounterView(String title) {
        frame = new JFrame(title);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JPanel panel = new JPanel(new GridLayout(2, 2, 20, 10));
        panel.setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10));
        counter.setEditable(false);
        panel.add(new JLabel("Counter:"));
        panel.add(counter);
        panel.add(up);
        panel.add(down);
        frame.setContentPane(panel);
    }
```

Kode 11.5 (Lanjutan) *CounterMvc.java: pencacah Up/Down sebagai model, interface view dengan implementasi Swing dan implementasi teks, serta controller. Dengan --selftest ia berjalan tanpa jendela.*

```
    void open() {
        frame.pack();
        frame.setLocationRelativeTo(null);  // tengah layar
        frame.setVisible(true);
    }

    @Override
    public void showValue(int value) {
        counter.setText(Integer.toString(value));
    }

    @Override
    public void onUp(Runnable action) {
        up.addActionListener(e -> action.run());
    }

    @Override
    public void onDown(Runnable action) {
        down.addActionListener(e -> action.run());
    }
}

// View pengganti untuk uji mandiri: mencetak alih-alih menggambar.
class TextCounterView implements CounterView {
    private Runnable upAction = () -> { };
    private Runnable downAction = () -> { };
    private String shown = "";

    @Override
    public void showValue(int value) {
        shown = Integer.toString(value);
        System.out.println("  [view] Counter: " + shown);
    }

    @Override
    public void onUp(Runnable action) {
        upAction = action;
    }

    @Override
    public void onDown(Runnable action) {
        downAction = action;
    }
```

Kode 11.5 (Lanjutan) *CounterMvc.java: pencacah Up/Down sebagai model, interface view dengan implementasi Swing dan implementasi teks, serta controller. Dengan --selftest ia berjalan tanpa jendela.*

```
    void click(String button) {
        System.out.println("click " + button);
        if (button.equals("Up")) {
            upAction.run();
        } else {
            downAction.run();
        }
    }

    String shownText() {
        return shown;
    }
}
```

11.6.1 Membaca program

Mulailah dengan `CounterModel`. List-nya berisi `IntConsumer` — *interface* fungsional standar untuk "sesuatu yang menerima sebuah `int`" — sehingga pengamat adalah apa saja yang memiliki metode `void accept(int value)`. `setValue` menyimpan nilai baru dan memanggil `accept` pada setiap pengamat: inilah gaya mendorong, menyerahkan bilangan baru kepada setiap pengamat. `increment` dan `decrement` melewati `setValue`, sehingga hanya ada satu tempat nilai berubah dan satu tempat pengamat diberi tahu. Model sama sekali tidak tahu apakah dirinya sedang ditampilkan di jendela, dicetak di konsol, atau tidak ditampilkan sama sekali.

`CounterView` adalah kontrak antara controller dan view. Ia memiliki tiga metode. `showValue(int)` adalah cara view *menerima* data. `onUp(Runnable)` dan `onDown(Runnable)` adalah cara view *melaporkan gestur*: controller menyerahkan kepada view sepotong kode untuk dijalankan ketika tombol yang bersangkutan diklik. `Runnable` adalah *interface* standar untuk "sepotong kode yang tidak menerima apa-apa dan tidak mengembalikan apa-apa", sehingga `model::increment` (referensi metode, Bab 10) adalah `Runnable` yang sah.

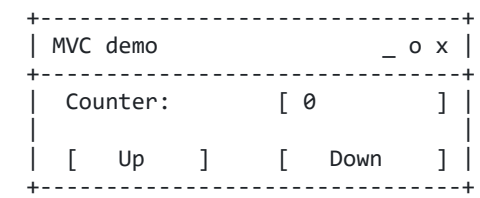
`CounterController` panjangnya empat baris, dan keempat baris itu adalah seluruh rangkaian sambungannya. `view.onUp(model::increment)` berkata: ketika pengguna mengeklik *Up*, naikkan model. `view.onDown(model::decrement)` berkata hal yang sama untuk *Down*. `model.addListener(view::showValue)` mendaftarkan view pada model, sehingga setiap perubahan model ditampilkan. Dan `view.showValue(model.getValue())` menggambar gambar pertama, karena model belum berubah sehingga belum memberi tahu siapa pun. Melupakan baris terakhir itu adalah kesalahan umum: jendela terbuka dengan kolom kosong, dan klik pertama memunculkan angka seolah-olah dengan sihir.

`SwingCounterView` adalah kode Swing dari Bab 10: sebuah `JPanel` dengan `GridLayout` 2 kali 2 yang memuat sebuah `JLabel`, sebuah `JTextField` hanya-baca, dan kedua `JButton`. `onUp`-nya mendaftarkan sebuah `ActionListener` yang sekadar menjalankan aksi dari controller: view tidak menentukan arti *Up*.

`main` membaca baris perintah. Sebuah angka menetapkan nilai awal; kata `--selftest` memilih uji mandiri. Untuk jalannya yang normal, ia membuat view dan controller di dalam `SwingUtilities.invokeLater`, karena Swing mengharuskan setiap *widget* dibuat dan disentuh di satu-satunya *event dispatch thread* miliknya, tidak pernah di *thread* `main` [8]; Bab 10 menjelaskan alasannya, dan Bab 14 kembali membahas *thread* secara mendalam.

11.6.2 Jendelanya

Jalankan `java CounterMvc` di komputer yang memiliki layar. Sebuah jendela kecil berjudul **MVC demo** muncul di tengah layar, berukuran sekitar 190 kali 80 piksel ditambah batang judulnya, tersusun sebagai kisi dua baris dan dua kolom:



Sel kiri atas adalah label **Counter:**; sel kanan atas adalah kolom teks yang menampilkan `0`, berlatar abu-abu karena tidak dapat disunting. Baris bawah memuat tombol **Up** dan **Down** yang sama lebar. Setiap klik pada **Up** menambah satu pada angka di kolom itu, setiap klik pada **Down**

menguranginya satu, persis seperti di **Gambar 11.9**. `java CounterMvc 100` dimulai dari 100. Menutup jendela mengakhiri program, karena `EXIT_ON_CLOSE`.

11.6.3 Uji mandiri

Komputer yang membangun buku ini tidak memiliki layar, begitu pula *server* yang menjalankan pengujian di kebanyakan proyek sungguhan. Di mesin seperti itu, `java CounterMvc` gagal begitu mencoba membuat sebuah `JFrame` (Subbab 11.13 memperlihatkan galat persisnya). Di sinilah pemisahan itu membuahkan hasil untuk pertama kalinya. Karena controller berbicara dengan view hanya melalui *interface* `CounterView`, kita dapat menyerahkan view *lain* kepadanya — yang mencetak alih-alih menggambar — dan menguji model serta controller sepenuhnya, tanpa jendela. Itulah yang dilakukan `--selftest`. `TextCounterView` menyimpan kedua aksi yang diberikan kepadanya dan menyediakan metode `click(button)` yang mencetak `click Up` atau `click Down` lalu menjalankan aksi yang bersangkutan, persis seperti yang akan dilakukan Swing ketika tombol sungguhan diklik. `showValue`-nya mencetak apa yang akan ditampilkan view sungguhan.

Menjalankan `java CounterMvc --selftest` mencetak:

```
  [view] Counter: 0
click Up
  [view] Counter: 1
click Up
  [view] Counter: 2
click Down
  [view] Counter: 1
click Up
  [view] Counter: 2
selftest passed: model holds 2, view shows 2
```

Baris pertama dicetak oleh konstruktor controller, yang menggambar gambar pertama. Setiap `click` kemudian menempuh seluruh jalur MVC: view menjalankan aksi controller, aksi itu memanggil model, model memberi tahu pengamatnya, dan pengamat itu — view teks — mencetak nilai baru. Di akhir, `selfTest` memeriksa bahwa model memegang 2 dan view menampilkan 2; bila salah satunya keliru, ia akan melempar `AssertionError` dan program berakhir dengan galat. `java CounterMvc --selftest 100` menjalankan klik-klik yang sama mulai dari 100 dan berakhir dengan `selftest passed: model holds 102, view shows 102`.

Apa yang dibuktikan uji mandiri, dan apa yang tidak? Ia membuktikan bahwa model mencacah dengan benar, bahwa controller memetakan *Up* ke *increment* dan *Down* ke *decrement*, bahwa pengamat terdaftar, dan bahwa gambar pertama digambar. Ia *tidak* membuktikan bahwa jendela Swing tampak benar: bahwa ejaan labelnya tepat, bahwa tombol-tombolnya cukup lebar, atau bahwa kolom teksnya hanya-baca. Fakta-fakta itu tinggal di `SwingCounterView`, selusin baris yang sengaja kita jaga bebas logika sehingga hanya sedikit yang bisa keliru di dalamnya. Inilah strategi umum untuk menguji program grafis: pindahkan setiap keputusan keluar dari view, uji keputusan-keputusan itu tanpa jendela, dan jaga view tetap begitu tipis sehingga membacanya saja sudah cukup [9].

Kode 11.6 adalah program yang sama dalam Python dengan Tkinter.

Kode 11.6 `counter_mvc.py`: pencacah dalam Python. `TkCounterView` mengimpor Tkinter di dalam konstruktornya, sehingga uji mandiri tetap berjalan bahkan di tempat Tkinter tidak terpasang.

```
"""counter_mvc.py: pencacah Up/Down sebagai model, view, dan controller."""
import sys


class CounterModel:
    """Model: sebuah bilangan bulat dan aturan untuk mengubahnya. Tanpa Tkinter sama
sekali."""

    def __init__(self, value=0):
        self._value = value
        self._listeners = []

    @property
    def value(self):
        return self._value

    def add_listener(self, listener):
        self._listeners.append(listener)

    def increment(self):
        self.set_value(self._value + 1)

    def decrement(self):
        self.set_value(self._value - 1)

    def set_value(self, new_value):
        self._value = new_value
        for listener in list(self._listeners):
            listener(self._value)  # beri tahu setiap pengamat


class CounterController:
    """Controller: mengubah gestur pengguna menjadi operasi model dan merangkai bagian-
bagiannya."""

    def __init__(self, model, view):
        view.on_up(model.increment)
        view.on_down(model.decrement)
        model.add_listener(view.show_value)
        view.show_value(model.value)  # gambar pertama


class TkCounterView:
    """View sungguhan: sebuah jendela Tkinter kecil."""
```

Kode 11.6 (Lanjutan) *counter_mvc.py: pencacah dalam Python. TkCounterView mengimpor Tkinter di dalam konstruktornya, sehingga uji mandiri tetap berjalan bahkan di tempat Tkinter tidak terpasang.*

```
    def __init__(self, title):
        import tkinter as tk  # diimpor di sini, agar uji mandiri tidak memerlukan Tkinter
        self._root = tk.Tk()
        self._root.title(title)
        self._text = tk.StringVar()
        tk.Label(self._root, text="Counter:").grid(
            row=0, column=0, sticky="w", padx=10, pady=5)
        tk.Entry(self._root, textvariable=self._text, width=6,
                 state="readonly").grid(row=0, column=1, padx=10, pady=5)
        self._up = tk.Button(self._root, text="Up", width=8)
        self._up.grid(row=1, column=0, padx=10, pady=5)
        self._down = tk.Button(self._root, text="Down", width=8)
        self._down.grid(row=1, column=1, padx=10, pady=5)

    def open(self):
        self._root.mainloop()

    def show_value(self, value):
        self._text.set(str(value))

    def on_up(self, action):
        self._up.config(command=action)

    def on_down(self, action):
        self._down.config(command=action)


class TextCounterView:
    """View pengganti untuk uji mandiri: mencetak alih-alih menggambar."""

    def __init__(self):
        self._up = lambda: None
        self._down = lambda: None
        self.shown = ""

    def show_value(self, value):
        self.shown = str(value)
        print("  [view] Counter: " + self.shown)

    def on_up(self, action):
        self._up = action

    def on_down(self, action):
        self._down = action
```

Kode 11.6 (Lanjutan) *counter_mvc.py: pencacah dalam Python. TkCounterView mengimpor Tkinter di dalam konstruktornya, sehingga uji mandiri tetap berjalan bahkan di tempat Tkinter tidak terpasang.*

```
    def click(self, button):
        print("click " + button)
        if button == "Up":
            self._up()
        else:
            self._down()


def self_test(model):
    """Model dan controller bekerja, dengan view teks sebagai ganti jendela."""
    view = TextCounterView()
    CounterController(model, view)
    start = model.value
    view.click("Up")
    view.click("Up")
    view.click("Down")
    view.click("Up")
    expected = start + 2
    assert model.value == expected and view.shown == str(expected), "selftest failed"
    print(f"selftest passed: model holds {model.value}, view shows {view.shown}")


def main(args):
    selftest = "--selftest" in args
    numbers = [a for a in args if a != "--selftest"]
    model = CounterModel(int(numbers[0]) if numbers else 0)
    if selftest:
        self_test(model)
        return
    view = TkCounterView("MVC demo")
    CounterController(model, view)
    view.open()


if __name__ == "__main__":
    main(sys.argv[1:])
```

`python3 counter_mvc.py --selftest` mencetak baris-baris yang persis sama dengan uji mandiri Java:

```
  [view] Counter: 0
click Up
  [view] Counter: 1
click Up
  [view] Counter: 2
click Down
  [view] Counter: 1
click Up
  [view] Counter: 2
selftest passed: model holds 2, view shows 2
```

Strukturnya adalah struktur Java, dengan sintaks Python yang lebih ringan. Pengamat model adalah *callable* apa pun yang menerima satu argumen. Controller menyerahkan metode terikat (*bound method*) — `model.increment`, `view.show_value` — yang merupakan padanan Python bagi referensi metode Java. Tidak ada *interface* `CounterView` di dalam kode, karena Python memeriksa pada saat program berjalan apakah sebuah objek memiliki metode yang diminta darinya (*duck typing*): apa saja yang memiliki `show_value`, `on_up`, dan `on_down` adalah view pencacah. (Subbab 11.9 memperlihatkan cara *menuliskan* kontrak semacam itu dengan `typing.Protocol` ketika program sudah cukup besar untuk memerlukannya.)

View Tkinter [10] membangun kisi dua kali dua yang sama dengan `grid(row=..., column=...)`. Angkanya ditampilkan dalam sebuah `Entry` ber-`state="readonly"` yang terhubung ke sebuah `StringVar`, sehingga `show_value` cukup memanggil `self._text.set(...)`. `on_up` mengatur `command` tombol, yang merupakan cara Tkinter mengatakan "panggil ini ketika diklik". Dan `open()` memanggil `mainloop()`, yang menyerahkan kendali kepada gelung peristiwa Tkinter sampai jendela ditutup.

Satu baris di view Tkinter tidak lazim: `import tkinter as tk` berada *di dalam* `__init__`, bukan di bagian atas berkas. Dengan begitu model, controller, dan uji mandiri tetap berjalan bahkan di mesin yang tidak memasang Tkinter — hal yang umum di *server*, dan berlaku bagi mesin yang menjalankan setiap program buku ini. (Di Linux, Tkinter sering menjadi paket tersendiri, `python3-tk` di Debian dan Ubuntu; pemasang dari python.org untuk Windows dan macOS sudah menyertakannya.) Tanpa Tkinter, `python3 counter_mvc.py` berhenti dengan `ModuleNotFoundError: No module named 'tkinter'`, tetapi `python3 counter_mvc.py --selftest` tetap berjalan. Jendela Tkinter memiliki tata letak yang sama dengan jendela Swing: **Counter:** dan sebuah kolom hanya-baca berisi `0` di baris pertama, **Up** dan **Down** di baris kedua.

11.7 Papan Tombol Ponsel: Teks Prediktif dalam MVC

Program berjendela kita yang kedua memiliki kisahnya sendiri. Sebelum ada layar sentuh, ponsel memiliki dua belas tombol, dan setiap tombol dari 2 sampai 9 memuat tiga atau empat huruf: 2 untuk *abc*, 3 untuk *def*, sampai 9 untuk *wxyz*. Dalam skema tertua, *multi-tap*, Anda menekan 4 dua kali untuk *h*, 3 dua kali untuk *e*, dan seterusnya — tiga belas tekanan untuk *hello*. Pada tahun 1990-an sebuah perusahaan kecil bernama Tegic Communications menciptakan **T9**, "*text on 9 keys*", tempat Anda menekan setiap tombol *sekali* per huruf dan ponsel menebak katanya dari sebuah kamus [11]. *hello* menjadi lima tekanan 4-3-5-5-6. Ketika beberapa kata berbagi tombol yang sama — 4-6-6-3 adalah *good*, tetapi juga *home*, *gone*, *hood*, *hoof*, dan *hone* — ponsel menampilkan satu, dan Anda menekan tombol *berikutnya*, biasanya bintang, untuk berpindah ke kata-kata lainnya.

Program papan tombol kita mengimplementasikan persis hal itu, dan rancangannya mengikuti rangkaian kelas asli untuk latihan ini: model bernama `DictionaryModel` yang menyimpan pesan, view `MessageView` yang menampilkannya di area teks, panel `KeypadController` dengan dua belas tombol, kelas pendengar `KeypadListener` yang meneruskan setiap tombol ke model, `MessageComponent` yang merakit bagian-bagiannya, dan `MessageGUI` dengan metode `main`. Pencarian di kamus berasal dari kelas bernama `PredictivePrototype`, yang mengubah sebuah kata menjadi urutan tombolnya dengan rangkaian panjang pernyataan `if`. Kita mempertahankan arsitektur itu dan membangun ulang setiap kelas dengan rapi.

11.7.1 Kata, tombol, dan tanda tangan

Model ini bertumpu pada satu fungsi. Misalkan κ (huruf Yunani *kappa*, dari *key*, tombol) memetakan setiap huruf ke tombolnya: $\kappa(a) = \kappa(b) = \kappa(c) = 2$, $\kappa(d) = \kappa(e) = \kappa(f) = 3$, dan seterusnya, dengan $\kappa(p) = \kappa(q) = \kappa(r) = \kappa(s) = 7$ dan $\kappa(w) = \kappa(x) = \kappa(y) = \kappa(z) = 9$. **Tanda tangan** (*signature*) sebuah kata $w = c_1 c_2 \ldots c_n$ yang terdiri atas n huruf adalah untaian tombol-tombolnya:

$$\sigma(c_1 c_2 \ldots c_n) = \kappa(c_1)\,\kappa(c_2) \ldots \kappa(c_n) \tag{11.3}$$

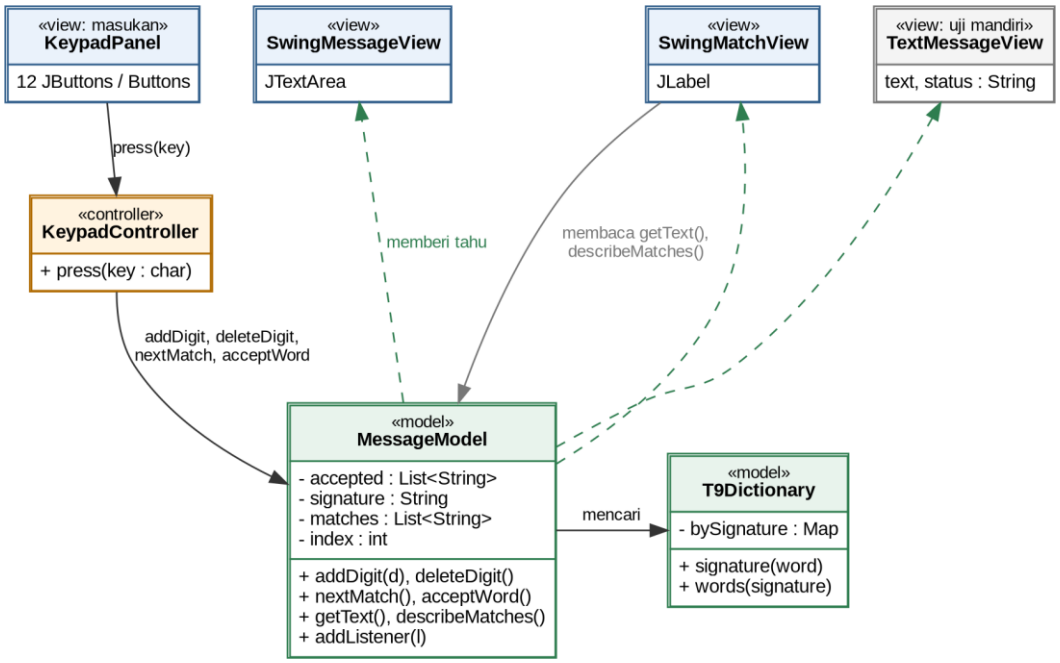
Sebagai contoh, menurut Persamaan (11.3), $\sigma(\text{hello}) = \kappa(h)\kappa(e)\kappa(l)\kappa(l)\kappa(o) = 4\,3\,5\,5\,6$, yaitu `43556`; $\sigma(\text{wash}) = 9\,2\,7\,4$; dan $\sigma(\text{good}) = \sigma(\text{home}) = 4663$. Fungsi σ tidak satu-satu — kata yang berbeda dapat berbagi tanda tangan — dan justru karena itulah pengguna kadang memerlukan tombol bintang. Tugas kamus adalah arah kebalikannya: diberi sebuah tanda tangan, kembalikan setiap kata yang memilikinya. Bila kamus memuat N kata, kita menghitung setiap tanda tangan sekali ketika kamus dibangun dan menyimpan kata-katanya di sebuah *map* dari tanda tangan ke list kata; setelah itu, mencari kata-kata untuk sebuah tanda tangan cukup dengan satu pencarian di *map*, sebesar apa pun N.

Model pesan lalu hanya memerlukan sedikit keadaan: kata-kata yang sudah diterima, tanda tangan yang sejauh ini diketik untuk kata yang sedang berjalan, list kata kamus yang cocok dengan tanda tangan itu, dan indeks kecocokan yang sedang ditampilkan. Empat operasi mengubahnya, dan controller memetakan setiap tombol ke salah satunya:

Tabel 11.3 Apa yang dilakukan setiap tombol papan tombol terhadap model pesan.

Tombol	Operasi	Pengaruh pada model
2 … 9	`addDigit(d)`	tambahkan *d* ke tanda tangan dan cari lagi; tampilkan kecocokan pertama
`*`	`nextMatch()`	tampilkan kecocokan berikutnya, kembali ke yang pertama setelah yang terakhir
0	`acceptWord()`	terima kata yang ditampilkan, seolah-olah spasi diketik, dan mulai kata baru
C	`deleteDigit()`	hapus angka terakhir tanda tangan dan cari lagi
1	—	tidak ada: tombol 1 tidak memuat huruf

Ketika tanda tangan yang sejauh ini diketik tidak cocok dengan kata apa pun — seperti yang terjadi di tengah *hello*, karena tak ada kata di kamus yang bertanda tangan `435` — model menampilkan angka-angkanya sendiri, sehingga pengguna dapat melihat bahwa tombol-tombolnya tercatat. **Gambar 11.10** memperlihatkan bagaimana kelas-kelasnya saling terkait.



Gambar 11.10 Kelas-kelas papan tombol prediktif. Panel tombol melaporkan setiap tekanan kepada controller, yang memanggil salah satu dari empat operasi model; model memakai kamus dan memberi tahu ketiga view-nya, yang membaca teks dan kecocokan saat ini dari model.

11.7.2 Programnya

Kode 11.7 adalah program Java-nya. Ia kode terpanjang sejauh ini, tetapi setiap kelas di dalamnya pendek dan hanya memiliki satu tugas.

Kode 11.7 `PredictiveKeypad.java`: papan tombol ponsel dengan teks prediktif. `T9Dictionary` dan `MessageModel` membentuk model, `KeypadController` controller, `SwingMessageView` dan `SwingMatchView` dua view dari model yang sama, dan `KeypadPanel` kedua belas tombol; `TextMessageView` menggantikan jendela dalam uji mandiri.

```
import java.awt.BorderLayout;
import java.awt.GridLayout;
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.util.ArrayList;
import java.util.Collection;
import java.util.List;
import java.util.Map;
import java.util.TreeMap;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import javax.swing.SwingUtilities;

public class PredictiveKeypad {
    static final List<String> BUILT_IN_WORDS = List.of(
            "a", "act", "and", "bat", "cat", "cod", "dry", "fold", "go", "gone",
            "good", "he", "hello", "hi", "home", "hone", "hood", "hoof", "i", "if",
            "in", "iron", "is", "it", "laundry", "mane", "me", "my", "name", "no",
            "oboe", "of", "on", "order", "ox", "ready", "shirt", "the", "tie", "to",
            "vie", "wash", "we", "world");

    public static void main(String[] args) throws IOException {
        boolean selftest = false;
        T9Dictionary dictionary = new T9Dictionary(BUILT_IN_WORDS);
        for (String arg : args) {
            if (arg.equals("--selftest")) {
                selftest = true;
            } else {
                dictionary = T9Dictionary.fromFile(Path.of(arg));  // misalnya daftar kata
            }
        }
        MessageModel model = new MessageModel(dictionary);
        KeypadController controller = new KeypadController(model);
        if (selftest) {
            selfTest(model, controller);
            return;
        }
        SwingUtilities.invokeLater(() -> {
            JFrame frame = new JFrame("Predictive Text");
            frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            JPanel screen = new JPanel(new BorderLayout());
```

Kode 11.7 (Lanjutan) *PredictiveKeypad.java: papan tombol ponsel dengan teks prediktif. T9Dictionary dan MessageModel membentuk model, KeypadController controller, SwingMessageView dan SwingMatchView dua view dari model yang sama, dan KeypadPanel kedua belas tombol; TextMessageView menggantikan jendela dalam uji mandiri.*

```
            screen.add(new SwingMessageView(model), BorderLayout.CENTER);
            screen.add(new SwingMatchView(model), BorderLayout.SOUTH);
            frame.add(screen, BorderLayout.CENTER);
            frame.add(new KeypadPanel(controller), BorderLayout.SOUTH);
            frame.setSize(240, 320);
            frame.setLocationRelativeTo(null);
            frame.setVisible(true);
        });
    }

    // Tekan tombol melalui controller; view teks menampilkan apa yang akan tampak di
jendela.
    static void selfTest(MessageModel model, KeypadController controller) {
        TextMessageView view = new TextMessageView(model);
        for (char key : "4355609675304663**092740143CC".toCharArray()) {
            controller.press(key);
            String line = String.format("key %c  %-26s %s",
                    key, "\"" + view.text() + "\"", view.status());
            System.out.println(line.stripTrailing());
        }
        if (!model.getText().equals("hello world home wash")) {
            throw new AssertionError("selftest failed: " + model.getText());
        }
        System.out.println("selftest passed: " + model.getWords().size() + " words");
    }
}

// Bagian dari model: kata apa saja yang dapat dimaksud oleh setiap urutan tombol.
class T9Dictionary {
    private static final String[] LETTERS =
            {"", "", "abc", "def", "ghi", "jkl", "mno", "pqrs", "tuv", "wxyz"};
    private final Map<String, List<String>> bySignature = new TreeMap<>();

    T9Dictionary(Collection<String> words) {
        for (String raw : words) {
            String word = raw.trim().toLowerCase();
            if (isValid(word)) {
                bySignature.computeIfAbsent(signature(word), k -> new
ArrayList<>()).add(word);
            }
        }
        for (List<String> list : bySignature.values()) {
            list.sort(null);  // alfabetis, agar urutannya dapat diramalkan
        }
    }
```

Kode 11.7 (Lanjutan) *PredictiveKeypad.java: papan tombol ponsel dengan teks prediktif. T9Dictionary dan MessageModel membentuk model, KeypadController controller, SwingMessageView dan SwingMatchView dua view dari model yang sama, dan KeypadPanel kedua belas tombol; TextMessageView menggantikan jendela dalam uji mandiri.*

```
    static T9Dictionary fromFile(Path path) throws IOException {
        return new T9Dictionary(Files.readAllLines(path));
    }

    static boolean isValid(String word) {
        return !word.isEmpty() && word.chars().allMatch(c -> c >= 'a' && c <= 'z');
    }

    static String lettersOf(char key) {
        return LETTERS[key - '0'];
    }

    static String signature(String word) {
        StringBuilder sig = new StringBuilder();
        for (char c : word.toCharArray()) {
            for (int key = 2; key <= 9; key++) {
                if (LETTERS[key].indexOf(c) >= 0) {
                    sig.append((char) ('0' + key));
                }
            }
        }
        return sig.toString();
    }

    List<String> words(String signature) {
        return List.copyOf(bySignature.getOrDefault(signature, List.of()));
    }
}

// Model: pesan yang sedang diketik. Ia tidak tahu apa-apa tentang tombol atau jendela.
class MessageModel {
    private final T9Dictionary dictionary;
    private final List<String> accepted = new ArrayList<>();
    private final List<Runnable> listeners = new ArrayList<>();
    private String signature = "";
    private List<String> matches = List.of();
    private int index;

    MessageModel(T9Dictionary dictionary) {
        this.dictionary = dictionary;
    }

    void addListener(Runnable listener) {
        listeners.add(listener);
    }
```

Kode 11.7 (Lanjutan) *PredictiveKeypad.java: papan tombol ponsel dengan teks prediktif. T9Dictionary dan MessageModel membentuk model, KeypadController controller, SwingMessageView dan SwingMatchView dua view dari model yang sama, dan KeypadPanel kedua belas tombol; TextMessageView menggantikan jendela dalam uji mandiri.*

```
    private void fireChanged() {
        for (Runnable listener : listeners) {
            listener.run();
        }
    }

    void addDigit(char digit) {
        if (digit < '2' || digit > '9') {
            throw new IllegalArgumentException("not a letter key: " + digit);
        }
        signature += digit;
        lookUp();
        fireChanged();
    }

    void deleteDigit() {
        if (!signature.isEmpty()) {
            signature = signature.substring(0, signature.length() - 1);
            lookUp();
            fireChanged();
        }
    }

    void nextMatch() {
        if (matches.size() > 1) {
            index = (index + 1) % matches.size();  // kembali memutar ke yang pertama
            fireChanged();
        }
    }

    void acceptWord() {
        if (!signature.isEmpty()) {
            accepted.add(currentWord());
            signature = "";
            lookUp();
            fireChanged();
        }
    }

    private void lookUp() {
        matches = dictionary.words(signature);
        index = 0;
    }
```

Kode 11.7 (Lanjutan) *PredictiveKeypad.java: papan tombol ponsel dengan teks prediktif. T9Dictionary dan MessageModel membentuk model, KeypadController controller, SwingMessageView dan SwingMatchView dua view dari model yang sama, dan KeypadPanel kedua belas tombol; TextMessageView menggantikan jendela dalam uji mandiri.*

```
    String getSignature() {
        return signature;
    }

    List<String> getMatches() {
        return matches;
    }

    int getMatchIndex() {
        return index;
    }

    // Urutan tombol yang tidak dikenal ditampilkan sebagai angkanya.
    String currentWord() {
        return matches.isEmpty() ? signature : matches.get(index);
    }

    List<String> getWords() {
        List<String> all = new ArrayList<>(accepted);
        if (!signature.isEmpty()) {
            all.add(currentWord());
        }
        return all;
    }

    String getText() {
        return String.join(" ", getWords());
    }

    String describeMatches() {
        if (signature.isEmpty()) {
            return "";
        }
        if (matches.isEmpty()) {
            return signature + " -> no word";
        }
        return signature + " -> " + currentWord()
                + " (" + (index + 1) + " of " + matches.size() + ")";
    }
}

// Controller: menerjemahkan tekanan tombol menjadi operasi model.
class KeypadController {
    private final MessageModel model;
```

Kode 11.7 (Lanjutan) *PredictiveKeypad.java: papan tombol ponsel dengan teks prediktif. T9Dictionary dan MessageModel membentuk model, KeypadController controller, SwingMessageView dan SwingMatchView dua view dari model yang sama, dan KeypadPanel kedua belas tombol; TextMessageView menggantikan jendela dalam uji mandiri.*

```
    KeypadController(MessageModel model) {
        this.model = model;
    }

    void press(char key) {
        switch (key) {
            case '0' -> model.acceptWord();
            case '*' -> model.nextMatch();
            case 'C' -> model.deleteDigit();
            case '1' -> { }  // tombol 1 tidak memuat huruf
            default -> model.addDigit(key);
        }
    }
}

// View 1: pesannya sendiri, dalam area teks hanya-baca.
class SwingMessageView extends JPanel {
    private final JTextArea message = new JTextArea(5, 20);

    SwingMessageView(MessageModel model) {
        super(new BorderLayout());
        message.setEditable(false);
        message.setLineWrap(true);
        message.setWrapStyleWord(true);
        message.setText("predictive text: enter text with 8 keys.");
        add(new JScrollPane(message), BorderLayout.CENTER);
        model.addListener(() -> message.setText(model.getText()));
    }
}

// View 2: urutan tombol dan kandidat yang sedang ditampilkan.
class SwingMatchView extends JLabel {
    SwingMatchView(MessageModel model) {
        super(" ");
        model.addListener(() -> setText(" " + model.describeMatches()));
    }
}
```

Kode 11.7 (Lanjutan) *PredictiveKeypad.java: papan tombol ponsel dengan teks prediktif. T9Dictionary dan MessageModel membentuk model, KeypadController controller, SwingMessageView dan SwingMatchView dua view dari model yang sama, dan KeypadPanel kedua belas tombol; TextMessageView menggantikan jendela dalam uji mandiri.*

```
// Papan tombol: dua belas tombol yang melaporkan tekanan ke controller.
class KeypadPanel extends JPanel {
    KeypadPanel(KeypadController controller) {
        super(new GridLayout(4, 3));
        for (char key : "123456789*0C".toCharArray()) {
            String letters = switch (key) {
                case '0' -> "_";
                case '*', 'C', '1' -> "";
                default -> T9Dictionary.lettersOf(key);
            };
            JButton button = new JButton(
                    "<html><center>" + key + "<br>" + letters + "</center></html>");
            button.addActionListener(e -> controller.press(key));
            add(button);
        }
    }
}

// View pengganti untuk uji mandiri: menggambar ke dalam string.
class TextMessageView {
    private String text = "predictive text: enter text with 8 keys.";
    private String status = "";

    TextMessageView(MessageModel model) {
        model.addListener(() -> {
            text = model.getText();
            status = model.describeMatches();
        });
    }

    String text() {
        return text;
    }

    String status() {
        return status;
    }
}
```

Mari kita membacanya kelas demi kelas, dari dasar arsitektur ke atas.

`T9Dictionary` adalah bagian dari model. Larik `LETTERS` menyimpan, di indeks *k*, huruf-huruf tombol *k* (tombol 0 dan 1 tidak memilikinya). `signature(word)` adalah Persamaan (11.3) dalam bentuk kode. `wordToSignature` yang asli memakai rangkaian `if`–`else` delapan cabang; tabel lebih pendek dan, yang lebih penting, ia adalah tabel yang *sama* yang memberi label tombol-tombol papan, sehingga label dan pencarian tidak mungkin berselisih. (Keduanya memang berselisih di versi asli: tombol 9 diberi label *vwxy*, sementara pencarian dengan benar memakai *wxyz*. Soal Latihan 11.6 menanyakan mengapa tidak ada pengujian model yang dapat menangkapnya.) Konstruktor membangun *map* dari tanda tangan ke kata dengan `computeIfAbsent`, yang membuat list kosong pada saat pertama kali sebuah tanda tangan dijumpai, lalu mengurutkan setiap list secara alfabetis agar urutan kecocokan dapat diramalkan. Versi asli memakai `HashSet`, yang urutannya bergantung pada kode *hash*. `isValid` hanya menyimpan kata yang tersusun dari huruf *a* sampai *z*. Dan `fromFile` membaca satu kata per baris dari berkas apa pun, misalnya `/usr/share/dict/words` di banyak sistem Linux dan macOS; program asli *mengharuskan* berkas itu dan tidak dapat berjalan tanpanya, sedangkan program kita membawa daftar bawaan kecil berisi 44 kata dan hanya memakai berkas bila Anda menyebutkannya di baris perintah.

`MessageModel` adalah jantung model. Ia menyimpan kata-kata yang sudah diterima, tanda tangan saat ini, kecocokan saat ini, dan indeks kecocokan yang ditampilkan, serta list pendengar bertipe `Runnable` — gaya menarik dari Subbab 11.5.4: pendengar hanya diberi tahu *bahwa* sesuatu berubah dan menanyakan kepada model apa yang ia perlukan. Setiap operasi dari keempat operasi di **Tabel 11.3** memeriksa apakah ada yang perlu dikerjakan, mengubah keadaan, dan memanggil `fireChanged()`. Pemeriksaan itu penting. `deleteDigit` pada tanda tangan kosong, `nextMatch` dengan kurang dari dua kecocokan, dan `acceptWord` ketika belum ada yang diketik tidak mengubah apa pun, sehingga tidak menembakkan apa pun: lagi-lagi Persamaan (11.2). Metode `press` yang asli memberi tahu setelah setiap tombol, bahkan ketika tidak ada yang berubah, dan tombol `'0'`-nya menambahkan kata kosong ke pesan bila ditekan dua kali, sehingga muncul spasi ganda. `addDigit` menolak apa pun selain `'2'` sampai `'9'` dengan `IllegalArgumentException`: model melindungi invariannya sendiri, apa pun yang dilakukan controller. Metode-metode pertanyaan — `currentWord`, `getWords`, `getText`, dan `describeMatches` — menghitung dari keadaan apa yang diperlukan view; `currentWord` menampilkan angka-angka bila tidak ada kecocokan.

`KeypadController` pendek saja: `press(key)` adalah tabel **Tabel 11.3** yang ditulis sebagai `switch`. Tugasnya menerjemahkan. Ia tahu bahwa tombol berlabel `0` berarti "terima katanya" pada papan tombol ini, dan itu adalah fakta tentang *antarmuka pengguna*, bukan tentang pesan — pada papan ketik lain, tombol spasi akan bermakna sama. Pengetahuan semacam itulah yang menjadi milik controller.

`SwingMessageView` dan `SwingMatchView` adalah dua view dari model yang sama. Yang pertama adalah `JPanel` yang memuat `JTextArea` hanya-baca berukuran 5 baris dan 20 kolom, dengan pemenggalan baris pada batas kata, di dalam sebuah `JScrollPane`; ia berlangganan pada model dengan lambda yang menyalin `model.getText()` ke area teks. Yang kedua adalah `JLabel` yang menampilkan `describeMatches()`, misalnya `4663 -> home (3 of 6)`, sehingga pengguna dapat melihat kandidat mana yang sedang ditampilkan. Tak satu view pun tahu tentang yang lain; keduanya dijaga tetap mutakhir oleh pemberitahuan yang sama — keadaan **Gambar 11.4** dalam bentuk mini.

`KeypadPanel` membangun kedua belas tombol dalam `GridLayout` empat baris dan tiga kolom, dengan urutan ponsel: 1 2 3, 4 5 6, 7 8 9, `*` 0 C. Teks setiap tombol adalah potongan HTML — `<html><center>2<br>abc</center></html>` — karena tombol Swing menerima HTML sederhana, cara termudah untuk menaruh huruf-hurufnya di baris kedua. Huruf-huruf itu berasal dari `T9Dictionary.lettersOf`. Pendengar setiap tombol memanggil `controller.press(key)`: panel adalah separuh masukan dari view, dan ia meneruskan gestur tanpa menafsirkannya.

`main` membangun kamus, model, dan controller, lalu menjalankan uji mandiri atau membangun jendela di *event dispatch thread*.

11.7.3 Jendelanya

Jalankan `java PredictiveKeypad`. Sebuah jendela berjudul **Predictive Text**, selebar 240 piksel dan setinggi 320, muncul di tengah layar:

```
+-----------------------------+
| Predictive Text       _ o x |
+-----------------------------+
| predictive text: enter text |
| with 8 keys.                |
|                             |
|                             |
|-----------------------------|
|                             |   <- baris kecocokan (mula-mula kosong)
+---------+---------+---------+
|    1    |    2    |    3    |
|         |   abc   |   def   |
+---------+---------+---------+
|    4    |    5    |    6    |
|   ghi   |   jkl   |   mno   |
+---------+---------+---------+
|    7    |    8    |    9    |
|  pqrs   |   tuv   |  wxyz   |
+---------+---------+---------+
|    *    |    0    |    C    |
|         |    _    |         |
+---------+---------+---------+
```

Bagian atas adalah area pesan, putih dan hanya-baca, yang menampilkan sapaan *predictive text: enter text with 8 keys.* sampai tombol pertama ditekan. Di bawahnya terbentang label satu baris untuk kecocokan saat ini, mula-mula kosong. Bagian bawah adalah papan tombol: dua belas tombol sama besar dalam empat baris, masing-masing menampilkan angkanya dan, di bawahnya, huruf-hurufnya; tombol 0 menampilkan garis bawah sebagai gambar spasi. Menekan 4, 3, 5, 5, 6 membuat area pesan menampilkan *i*, *he*, *435*, *4355*, dan akhirnya *hello*, sementara baris kecocokan menampilkan, misalnya, `43 -> he (1 of 2)` lalu `43556 -> hello (1 of 1)`. Menekan 0 menerima kata itu; menekan 4, 6, 6, 3 menampilkan *gone*, dan menekan `*` dua kali mengubahnya menjadi *good* lalu *home*, dengan baris kecocokan berbunyi `4663 -> home (3 of 6)`.

11.7.4 Uji mandiri

`java PredictiveKeypad --selftest` menekan tombol-tombol sebuah pesan utuh melalui controller — persis panggilan yang akan dilakukan tombol-tombolnya — sementara sebuah `TextMessageView` mencatat apa yang akan ditampilkan kedua view jendela. Setelah setiap tombol, uji mandiri mencetak tombol itu, teks pesan, dan baris kecocokan:

```
key 4  "i"                        4 -> i (1 of 1)
key 3  "he"                       43 -> he (1 of 2)
key 5  "435"                      435 -> no word
key 5  "4355"                     4355 -> no word
key 6  "hello"                    43556 -> hello (1 of 1)
key 0  "hello"
key 9  "hello 9"                  9 -> no word
key 6  "hello 96"                 96 -> no word
key 7  "hello 967"                967 -> no word
key 5  "hello 9675"               9675 -> no word
key 3  "hello world"              96753 -> world (1 of 1)
key 0  "hello world"
key 4  "hello world i"            4 -> i (1 of 1)
key 6  "hello world go"           46 -> go (1 of 2)
key 6  "hello world 466"          466 -> no word
key 3  "hello world gone"         4663 -> gone (1 of 6)
key *  "hello world good"         4663 -> good (2 of 6)
key *  "hello world home"         4663 -> home (3 of 6)
key 0  "hello world home"
key 9  "hello world home 9"       9 -> no word
key 2  "hello world home 92"      92 -> no word
key 7  "hello world home 927"     927 -> no word
key 4  "hello world home wash"    9274 -> wash (1 of 1)
key 0  "hello world home wash"
key 1  "hello world home wash"
key 4  "hello world home wash i"  4 -> i (1 of 1)
key 3  "hello world home wash he" 43 -> he (1 of 2)
key C  "hello world home wash i"  4 -> i (1 of 1)
key C  "hello world home wash"
selftest passed: 4 words
```

Bacalah keluaran itu seperti film dari layar. Lima tombol pertama mengeja *hello*, dan keadaan-keadaan antaranya memperlihatkan kamus sedang bekerja: setelah `4` satu-satunya kecocokan adalah kata satu huruf *i*; setelah `43` ada dua kecocokan, *he* dan *if*, dan yang pertama menurut abjad ditampilkan; `435` dan `4355` tidak cocok dengan apa pun, sehingga angka-angkanya yang ditampilkan; `43556` adalah *hello*. Tombol `0` menerima kata itu, dan baris kecocokan menjadi kosong karena kata baru telah dimulai. *world* mengikuti pola yang sama melalui lima tombol. Lalu `4663` menghasilkan *gone*, yang pertama dari enam kecocokan, dan dua tekanan `*` berpindah ke *good* dan *home*. Setelah `0` dan `9274` — *wash* — dan `0` lagi, tombol `1` tidak mengubah apa pun, sebagaimana dijanjikan **Tabel 11.3**. Empat tombol terakhir memperlihatkan penghapusan: `4` dan `3` memulai *he*, dan dua tekanan `C` menghapus kedua angka itu lagi, menyisakan pesan *hello world home wash*. Uji mandiri memeriksa teks akhir itu dan mencetak putusannya.

Uji mandiri ini menemukan *bug* sungguhan ketika bab ini sedang ditulis: versi pertama untaian tombolnya berisi `466` di tempat yang seharusnya `4663`, dan pemeriksaan penutupnya gagal dengan `selftest failed: hello world 466 wash`. Namun ia tidak dapat melihat label tombol yang salah eja; ia memeriksa *logika* program, dalam satu detik, setelah setiap perubahan.

11.7.5 Papan tombol yang sama dalam Python

Kode 11.8 adalah versi Python-nya. Model dan controller-nya terjemahan baris demi baris; jendelanya dibangun oleh satu kelas, `TkKeypadWindow`, yang memuat kedua view dan papan tombol.

Kode 11.8 `predictive_keypad.py`: papan tombol prediktif dalam Python. Model dan controller-nya sepadan dengan kelas-kelas Java; `TkKeypadWindow` membangun kotak pesan, baris kecocokan, dan kedua belas tombol dengan Tkinter.

```
"""predictive_keypad.py: papan tombol ponsel dengan teks prediktif, sebagai MVC."""
import sys

BUILT_IN_WORDS = [
    "a", "act", "and", "bat", "cat", "cod", "dry", "fold", "go", "gone",
    "good", "he", "hello", "hi", "home", "hone", "hood", "hoof", "i", "if",
    "in", "iron", "is", "it", "laundry", "mane", "me", "my", "name", "no",
    "oboe", "of", "on", "order", "ox", "ready", "shirt", "the", "tie", "to",
    "vie", "wash", "we", "world"]

LETTERS = ["", "", "abc", "def", "ghi", "jkl", "mno", "pqrs", "tuv", "wxyz"]


class T9Dictionary:
    """Bagian dari model: kata apa saja yang dapat dimaksud oleh setiap urutan tombol."""

    def __init__(self, words):
        self._by_signature = {}
        for raw in words:
            word = raw.strip().lower()
            if self.is_valid(word):
                self._by_signature.setdefault(self.signature(word), []).append(word)
        for matches in self._by_signature.values():
            matches.sort()  # alfabetis, agar urutannya dapat diramalkan

    @classmethod
    def from_file(cls, path):
        with open(path, encoding="utf-8") as f:
            return cls(f)

    @staticmethod
    def is_valid(word):
        return word != "" and all("a" <= c <= "z" for c in word)

    @staticmethod
    def letters_of(key):
        return LETTERS[int(key)]

    @staticmethod
    def signature(word):
        return "".join(str(key) for c in word
                       for key in range(2, 10) if c in LETTERS[key])

    def words(self, signature):
        return list(self._by_signature.get(signature, []))
```

Kode 11.8 (Lanjutan) *predictive_keypad.py: papan tombol prediktif dalam Python. Model dan controller-nya sepadan dengan kelas-kelas Java; TkKeypadWindow membangun kotak pesan, baris kecocokan, dan kedua belas tombol dengan Tkinter.*

```
class MessageModel:
    """Model: pesan yang sedang diketik. Ia tidak tahu apa-apa tentang tombol atau
jendela."""

    def __init__(self, dictionary):
        self._dictionary = dictionary
        self._accepted = []
        self._listeners = []
        self.signature = ""
        self.matches = []
        self.index = 0

    def add_listener(self, listener):
        self._listeners.append(listener)

    def _fire_changed(self):
        for listener in list(self._listeners):
            listener()

    def add_digit(self, digit):
        if len(digit) != 1 or digit not in "23456789":
            raise ValueError("not a letter key: " + digit)
        self.signature += digit
        self._look_up()
        self._fire_changed()

    def delete_digit(self):
        if self.signature:
            self.signature = self.signature[:-1]
            self._look_up()
            self._fire_changed()

    def next_match(self):
        if len(self.matches) > 1:
            self.index = (self.index + 1) % len(self.matches)  # kembali memutar
            self._fire_changed()

    def accept_word(self):
        if self.signature:
            self._accepted.append(self.current_word())
            self.signature = ""
            self._look_up()
            self._fire_changed()
```

Kode 11.8 (Lanjutan) *predictive_keypad.py: papan tombol prediktif dalam Python. Model dan controller-nya sepadan dengan kelas-kelas Java; TkKeypadWindow membangun kotak pesan, baris kecocokan, dan kedua belas tombol dengan Tkinter.*

```
    def _look_up(self):
        self.matches = self._dictionary.words(self.signature)
        self.index = 0

    def current_word(self):
        # Urutan tombol yang tidak dikenal ditampilkan sebagai angkanya.
        return self.matches[self.index] if self.matches else self.signature

    def words(self):
        return self._accepted + ([self.current_word()] if self.signature else [])

    def text(self):
        return " ".join(self.words())

    def describe_matches(self):
        if not self.signature:
            return ""
        if not self.matches:
            return self.signature + " -> no word"
        return (f"{self.signature} -> {self.current_word()} "
                f"({self.index + 1} of {len(self.matches)})")


class KeypadController:
    """Controller: menerjemahkan tekanan tombol menjadi operasi model."""

    def __init__(self, model):
        self._model = model

    def press(self, key):
        if key == "0":
            self._model.accept_word()
        elif key == "*":
            self._model.next_match()
        elif key == "C":
            self._model.delete_digit()
        elif key == "1":
            pass  # tombol 1 tidak memuat huruf
        else:
            self._model.add_digit(key)


class TkKeypadWindow:
    """Kedua view dan papan tombol, dalam satu jendela Tkinter."""
```

Kode 11.8 (Lanjutan) *predictive_keypad.py: papan tombol prediktif dalam Python. Model dan controller-nya sepadan dengan kelas-kelas Java; TkKeypadWindow membangun kotak pesan, baris kecocokan, dan kedua belas tombol dengan Tkinter.*

```
    def __init__(self, model, controller):
        import tkinter as tk  # diimpor di sini, agar uji mandiri tidak memerlukan Tkinter
        self._model = model
        self._root = tk.Tk()
        self._root.title("Predictive Text")
        self._root.geometry("240x320")
        # View 1: pesannya sendiri, dalam kotak teks hanya-baca.
        self._message = tk.Text(self._root, height=5, width=20, wrap="word")
        self._message.insert("1.0", "predictive text: enter text with 8 keys.")
        self._message.config(state="disabled")
        self._message.pack(fill="both", expand=True)
        # View 2: urutan tombol dan kandidat yang sedang ditampilkan.
        self._status = tk.Label(self._root, text=" ", anchor="w")
        self._status.pack(fill="x")
        # Papan tombol: dua belas tombol yang melaporkan tekanan ke controller.
        pad = tk.Frame(self._root)
        pad.pack(fill="x")
        for i, key in enumerate("123456789*0C"):
            letters = ("_" if key == "0" else "" if key in "*C1"
                       else T9Dictionary.letters_of(key))
            tk.Button(pad, text=f"{key}\n{letters}", width=6,
                      command=lambda k=key: controller.press(k)).grid(
                row=i // 3, column=i % 3, sticky="nsew")
        for column in range(3):
            pad.columnconfigure(column, weight=1)
        model.add_listener(self.refresh)

    def refresh(self):
        self._message.config(state="normal")
        self._message.delete("1.0", "end")
        self._message.insert("1.0", self._model.text())
        self._message.config(state="disabled")
        self._status.config(text=" " + self._model.describe_matches())

    def open(self):
        self._root.mainloop()


class TextMessageView:
    """View pengganti untuk uji mandiri: menggambar ke dalam string."""

    def __init__(self, model):
        self.text = "predictive text: enter text with 8 keys."
        self.status = ""
```

Kode 11.8 (Lanjutan) *predictive_keypad.py: papan tombol prediktif dalam Python. Model dan controller-nya sepadan dengan kelas-kelas Java; TkKeypadWindow membangun kotak pesan, baris kecocokan, dan kedua belas tombol dengan Tkinter.*

```
        def refresh():
            self.text = model.text()
            self.status = model.describe_matches()
        model.add_listener(refresh)


def self_test(model, controller):
    """Tekan tombol melalui controller; view teks menampilkan apa yang akan tampak di
jendela."""
    view = TextMessageView(model)
    for key in "4355609675304663**092740143CC":
        controller.press(key)
        quoted = '"' + view.text + '"'
        line = f"key {key}  {quoted:<26} {view.status}"
        print(line.rstrip())
    assert model.text() == "hello world home wash", "selftest failed: " + model.text()
    print(f"selftest passed: {len(model.words())} words")


def main(args):
    dictionary = T9Dictionary(BUILT_IN_WORDS)
    for arg in args:
        if arg != "--selftest":
            dictionary = T9Dictionary.from_file(arg)  # misalnya daftar kata
    model = MessageModel(dictionary)
    controller = KeypadController(model)
    if "--selftest" in args:
        self_test(model, controller)
        return
    TkKeypadWindow(model, controller).open()


if __name__ == "__main__":
    main(sys.argv[1:])
```

`python3 predictive_keypad.py --selftest` menghasilkan keluaran yang identik dengan keluaran Java, karakter demi karakter:

```
key 4  "i"                        4 -> i (1 of 1)
key 3  "he"                       43 -> he (1 of 2)
key 5  "435"                      435 -> no word
key 5  "4355"                     4355 -> no word
key 6  "hello"                    43556 -> hello (1 of 1)
key 0  "hello"
key 9  "hello 9"                  9 -> no word
key 6  "hello 96"                 96 -> no word
key 7  "hello 967"                967 -> no word
key 5  "hello 9675"               9675 -> no word
key 3  "hello world"              96753 -> world (1 of 1)
key 0  "hello world"
key 4  "hello world i"            4 -> i (1 of 1)
key 6  "hello world go"           46 -> go (1 of 2)
key 6  "hello world 466"          466 -> no word
key 3  "hello world gone"         4663 -> gone (1 of 6)
key *  "hello world good"         4663 -> good (2 of 6)
key *  "hello world home"         4663 -> home (3 of 6)
key 0  "hello world home"
key 9  "hello world home 9"       9 -> no word
key 2  "hello world home 92"      92 -> no word
key 7  "hello world home 927"     927 -> no word
key 4  "hello world home wash"    9274 -> wash (1 of 1)
key 0  "hello world home wash"
key 1  "hello world home wash"
key 4  "hello world home wash i"  4 -> i (1 of 1)
key 3  "hello world home wash he" 43 -> he (1 of 2)
key C  "hello world home wash i"  4 -> i (1 of 1)
key C  "hello world home wash"
selftest passed: 4 words
```

Beberapa rincian Python patut mendapat satu kalimat masing-masing. `signature` adalah satu ekspresi generator: untuk setiap karakter `c` dari kata itu, dan untuk setiap tombol dari 2 sampai 9, ia menghasilkan angka tombol bila `c` termasuk huruf tombol tersebut. `setdefault(key, [])` memainkan peran `computeIfAbsent` di Java. `from_file` menyerahkan objek berkas yang terbuka langsung ke konstruktor, karena menelusuri sebuah berkas menghasilkan baris-barisnya. Di jendela Tkinter, pesan ditampilkan oleh *widget* `Text` berukuran 5 baris dan 20 karakter dengan `wrap="word"`; *widget* `Text` hanya dapat diubah selama keadaannya `"normal"`, sehingga `refresh` mengalihkannya ke normal, mengganti isinya, lalu mengembalikannya ke `"disabled"` agar pengguna tidak dapat mengetik di dalamnya. Tombol-tombolnya dibuat dalam sebuah gelung, dan perintah masing-masing adalah `lambda k=key: controller.press(k)`. Bagian `k=key` sangat penting: tanpanya, kedua belas lambda akan merujuk ke variabel `key` yang *sama*, yang setelah gelung selesai berisi `"C"`, sehingga setiap tombol akan menghapus satu angka. Argumen bawaan itu menangkap nilai pada saat setiap lambda dibuat — padanan Python bagi aturan Java bahwa lambda hanya boleh menangkap variabel yang *effectively final*, yang dibahas di Bab 10.

11.8 Menguji Model Tanpa GUI

Uji mandiri di dua subbab terakhir menjalankan sebuah *skenario*: rangkaian klik, dicetak sambil berlangsung, dengan satu pemeriksaan di akhir. Itu berharga, tetapi itu bukan *unit test*. Sebuah **unit test** memeriksa satu perilaku kecil secara terpisah, memiliki nama yang deskriptif, dan melaporkan dengan jelas perilaku mana yang gagal. Bab 6 memperkenalkan *unit test* dengan JUnit 5 dan `unittest` milik Python. Karena model papan tombol kita tidak mengimpor apa pun dari Swing atau Tkinter, kita dapat mengujinya persis seperti kelas lainnya, tanpa jendela, tanpa *thread* peristiwa, dan tanpa layar.

Apa yang sebaiknya diperiksa? Aturan yang baik adalah menguji setiap *janji* yang dibuat model, termasuk janji tentang apa yang *tidak* akan dilakukannya:

- fungsi tanda tangan menghitung Persamaan (11.3) dengan benar;
- kata yang tidak sah ditolak oleh kamus;
- kecocokan dikembalikan dalam urutan abjad;
- urutan tombol yang tak dikenal ditampilkan sebagai angka-angkanya;
- `nextMatch` kembali memutar setelah kecocokan terakhir;
- model menembakkan tepat satu peristiwa per perubahan sungguhan dan tidak satu pun untuk operasi yang tidak mengubah apa-apa;
- model menolak angka yang bukan tombol huruf;
- controller memetakan tombol ke operasi yang benar.

Kode 11.9 menguji janji-janji itu dengan Java biasa: sebuah metode kecil `check` membandingkan nilai yang diharapkan dengan nilai sebenarnya dan mencetak `PASS` atau `FAIL`, dengan gaya program pemeriksa di Bab 6. Dengan JUnit, setiap blok akan menjadi metode beranotasi `@Test` dan setiap `check` menjadi `assertEquals` (API Jupiter dari JUnit 5 tidak berubah di JUnit 6, versi mayor terkini, yaitu 6.1.3 pada Agustus 2026, yang memerlukan Java 17 atau lebih baru [12]); versi biasa ini dapat berjalan di mana saja hanya dengan JDK, dan itulah sebabnya buku ini memakainya di sini.

Kode 11.9 `MessageModelCheck.java`: sepuluh pemeriksaan terhadap model dan controller papan tombol, dijalankan tanpa jendela apa pun.

```
import java.util.List;
import java.util.Objects;

public class MessageModelCheck {
    private static int passed;
    private static int failed;

    public static void main(String[] args) {
        check("signature of hello", "43556", T9Dictionary.signature("hello"));
        check("signature of wash", "9274", T9Dictionary.signature("wash"));
        check("words are rejected unless a-z", false, T9Dictionary.isValid("don't"));

        MessageModel model = newModel();
        type(model, "4663");
        check("matches are alphabetical",
                List.of("gone", "good", "home", "hone", "hood", "hoof"),
model.getMatches());

        model = newModel();
        type(model, "96");
        check("an unknown sequence shows its digits", "96", model.getText());

        model = newModel();
        type(model, "4663");
        for (int i = 0; i < 6; i++) {
            model.nextMatch();
        }
        check("six presses of * wrap round to the first match", "gone",
model.currentWord());

        model = newModel();
        int[] events = {0};
        model.addListener(() -> events[0]++);
        type(model, "43556");
        model.acceptWord();
        model.acceptWord();  // tak ada yang diterima: tak ada perubahan, tak ada peristiwa
        model.deleteDigit(); // tak ada yang dihapus: tak ada perubahan, tak ada peristiwa
        check("one event per real change", 6, events[0]);
        check("accepted text", "hello", model.getText());

        try {
            newModel().addDigit('1');
            check("key 1 is refused", "exception", "no exception");
        } catch (IllegalArgumentException e) {
            check("key 1 is refused", "not a letter key: 1", e.getMessage());
        }
```

Kode 11.9 (Lanjutan) *MessageModelCheck.java: sepuluh pemeriksaan terhadap model dan controller papan tombol, dijalankan tanpa jendela apa pun.*

```
        model = newModel();
        KeypadController controller = new KeypadController(model);
        for (char key : "8430*0".toCharArray()) {
            controller.press(key);
        }
        check("controller: 843, accept, then * and 0 do nothing", "the", model.getText());

        System.out.println(passed + " of " + (passed + failed) + " checks passed");
        if (failed > 0) {
            System.exit(1);
        }
    }

    static MessageModel newModel() {
        return new MessageModel(new T9Dictionary(PredictiveKeypad.BUILT_IN_WORDS));
    }

    static void type(MessageModel model, String digits) {
        for (char d : digits.toCharArray()) {
            model.addDigit(d);
        }
    }

    static void check(String name, Object expected, Object actual) {
        if (Objects.equals(expected, actual)) {
            passed++;
            System.out.println("PASS " + name);
        } else {
            failed++;
            System.out.println("FAIL " + name + ": expected " + expected + ", got " +
actual);
        }
    }
}
```

Kompilasi bersama papan tombolnya (`javac *.java` di folder `java` bab ini mengompilasi semuanya) lalu jalankan `java MessageModelCheck`:

```
PASS signature of hello
PASS signature of wash
PASS words are rejected unless a-z
PASS matches are alphabetical
PASS an unknown sequence shows its digits
PASS six presses of * wrap round to the first match
PASS one event per real change
PASS accepted text
PASS key 1 is refused
PASS controller: 843, accept, then * and 0 do nothing
10 of 10 checks passed
```

Pemeriksaan yang paling menarik adalah *one event per real change*: pendengar pencacah melihat lima angka dan satu kata yang diterima, dan tidak melihat apa pun untuk panggilan `acceptWord` dan `deleteDigit` yang tidak punya pekerjaan — 6 peristiwa, bukan 8. (Pencacahnya berupa larik satu elemen karena lambda Java tidak boleh memberi nilai kepada variabel lokal metode yang melingkupinya; namun ia boleh mengubah isi larik yang dirujuk variabel itu.) Pemeriksaan *key 1 is refused* memperlihatkan cara menguji eksepsi dengan Java biasa: panggil metodenya di dalam `try`, laporkan kegagalan bila metode itu kembali secara normal, dan bandingkan pesan eksepsinya di dalam `catch`. Pemeriksaan terakhir menjalankan controller dan model bersama-sama: `8430*0` mengetik *the*, menerimanya, lalu menekan `*` dan `0` ketika belum ada yang diketik, yang tidak boleh mengubah apa pun.

Program berakhir dengan `System.exit(1)` bila ada pemeriksaan yang gagal; status keluar yang bukan nol memberi tahu sistem operasi, dan proses *build* otomatis mana pun, bahwa program itu gagal.

Kode 11.10 adalah kumpulan pengujian yang sama dengan `unittest` milik Python.

Kode 11.10 `test_keypad_model.py`: *unit test* model dan controller papan tombol dengan `unittest`.

```
"""test_keypad_model.py: unit test untuk model papan tombol, tanpa jendela sama sekali."""
import unittest

from predictive_keypad import (BUILT_IN_WORDS, KeypadController, MessageModel,
                               T9Dictionary)


def new_model():
    return MessageModel(T9Dictionary(BUILT_IN_WORDS))


def type_digits(model, digits):
    for d in digits:
        model.add_digit(d)


class DictionaryTest(unittest.TestCase):
    def test_signatures(self):
        self.assertEqual(T9Dictionary.signature("hello"), "43556")
        self.assertEqual(T9Dictionary.signature("wash"), "9274")

    def test_only_a_to_z_words_are_valid(self):
        self.assertFalse(T9Dictionary.is_valid("don't"))


class MessageModelTest(unittest.TestCase):
    def setUp(self):
        self.model = new_model()  # model baru sebelum setiap pengujian
        self.events = 0

        def count():
            self.events += 1
        self.model.add_listener(count)

    def test_matches_are_alphabetical(self):
        type_digits(self.model, "4663")
        self.assertEqual(self.model.matches,
                         ["gone", "good", "home", "hone", "hood", "hoof"])

    def test_unknown_sequence_shows_its_digits(self):
        type_digits(self.model, "96")
        self.assertEqual(self.model.text(), "96")
```

Kode 11.10 (Lanjutan) *test_keypad_model.py: unit test model dan controller papan tombol dengan unittest.*

```
    def test_star_wraps_round(self):
        type_digits(self.model, "4663")
        for _ in range(6):
            self.model.next_match()
        self.assertEqual(self.model.current_word(), "gone")

    def test_one_event_per_real_change(self):
        type_digits(self.model, "43556")
        self.model.accept_word()
        self.model.accept_word()   # tak ada yang diterima: tak ada perubahan, tak ada
peristiwa
        self.model.delete_digit()  # tak ada yang dihapus: tak ada perubahan, tak ada
peristiwa
        self.assertEqual(self.events, 6)
        self.assertEqual(self.model.text(), "hello")

    def test_key_1_is_refused(self):
        with self.assertRaises(ValueError) as caught:
            self.model.add_digit("1")
        self.assertEqual(str(caught.exception), "not a letter key: 1")


class KeypadControllerTest(unittest.TestCase):
    def test_controller_maps_keys_to_operations(self):
        model = new_model()
        controller = KeypadController(model)
        for key in "8430*0":
            controller.press(key)
        self.assertEqual(model.text(), "the")


if __name__ == "__main__":
    unittest.main()
```

Jalankan pengujian dengan `python3 -m unittest -v test_keypad_model` di folder `python` bab ini (`-v` meminta satu baris untuk setiap pengujian):

```
test_only_a_to_z_words_are_valid
(test_keypad_model.DictionaryTest.test_only_a_to_z_words_are_valid) ... ok
test_signatures (test_keypad_model.DictionaryTest.test_signatures) ... ok
test_controller_maps_keys_to_operations
(test_keypad_model.KeypadControllerTest.test_controller_maps_keys_to_operations) ... ok
test_key_1_is_refused (test_keypad_model.MessageModelTest.test_key_1_is_refused) ... ok
test_matches_are_alphabetical
(test_keypad_model.MessageModelTest.test_matches_are_alphabetical) ... ok
test_one_event_per_real_change
(test_keypad_model.MessageModelTest.test_one_event_per_real_change) ... ok
test_star_wraps_round (test_keypad_model.MessageModelTest.test_star_wraps_round) ... ok
test_unknown_sequence_shows_its_digits
(test_keypad_model.MessageModelTest.test_unknown_sequence_shows_its_digits) ... ok

----------------------------------------------------------------------
Ran 8 tests in 0.001s

OK
```

`unittest` menemukan setiap metode yang namanya diawali `test` di setiap subkelas `TestCase`, menjalankan masing-masing secara terpisah, dan melaporkan `ok` atau alasan kegagalannya. Metode `setUp` berjalan sebelum *setiap* pengujian, sehingga setiap pengujian `MessageModelTest` dimulai dengan model baru dan pencacah peristiwa bernilai nol; pengujian yang berbagi keadaan dapat saling menggagalkan dengan cara yang membingungkan, dan model baru untuk setiap pengujian mencegahnya. `assertRaises` yang dipakai sebagai *context manager* (`with`

`self.assertRaises(ValueError) as caught:`) memeriksa bahwa bloknya melempar eksepsi itu, dan `caught.exception` memungkinkan kita memeriksa pesannya juga. Pengujian dijalankan menurut urutan abjad namanya; pengujian yang ditulis dengan baik tidak bergantung pada urutan. Semuanya selesai dalam beberapa milidetik — cukup cepat untuk dijalankan setelah setiap perubahan, sesuatu yang tidak pernah bisa dicapai pengujian yang memerlukan jendela.

> **WAWASAN KUNCI:** Pertanyaan "bagaimana cara menguji GUI saya?" biasanya pertanyaan yang keliru. Tanyakan sebaliknya: "seberapa sedikit logika yang dapat saya tinggalkan di GUI saya?" Pindahkan setiap keputusan ke model atau controller, tempat sebuah *unit test* dapat menjangkaunya dalam hitungan milidetik, dan jaga view tetap begitu tipis sehingga tak ada lagi yang perlu diuji darinya selain rupanya.

11.9 Meja Depan Profitina Laundry: Aplikasi MVC yang Lengkap

Sekarang kita kembali ke kisah Subbab 11.2 dan membangun program meja depan dengan benar. Persyaratannya adalah hal-hal yang sudah dikenal petugas:

1. Petugas memasukkan nama pelanggan, memilih satu dari tiga layanan, mengetik berat dalam kilogram, dan boleh mencentang *Express*; mengeklik **Add order** mencatat pesanan dengan nomor pesanan berikutnya dan menampilkan harganya.
2. Masukan yang tidak sah ditolak dengan pesan yang jelas dan tidak mengubah apa pun: nama kosong, berat yang bukan angka, berat nol atau kurang, atau lebih dari 50 kg (kapasitas mesin terbesar binatu itu). Karena orang Indonesia menulis bilangan desimal dengan koma, `2,6` harus diterima sebagai 2,6.
3. Pesanan-pesanan didaftar dengan nomor, pelanggan, layanan, berat, tanda kilat, harga, dan statusnya.
4. Petugas memilih sebuah pesanan dan mengeklik **Advance status** untuk memajukannya satu langkah: *Received*, *Washing*, *Ready*, *Picked up*. Pesanan yang sudah diambil tidak dapat maju lagi.
5. Sebuah baris ringkasan menampilkan jumlah pesanan, berapa yang masih *Received*, berapa yang *Ready* untuk diambil, dan total pendapatan.

11.9.1 Aturan-aturan model

Harga mengikuti aturan Bab 7. Setiap layanan s memiliki tarif r_s dalam rupiah per kilogram dan berat minimum w_s: Wash & Fold memiliki $r = 7000$ dan $w = 3$ kg, Wash, Dry & Iron memiliki $r = 10000$ dan $w = 3$ kg, dan Iron Only memiliki $r = 5000$ dan $w = 2$ kg. Pelanggan yang membawa w kilogram dikenai biaya paling sedikit sebesar minimum, dibulatkan *ke atas* ke setengah kilogram berikutnya. Membulatkan bilangan x ke atas ke setengah berikutnya dilakukan dengan menggandakannya, membulatkannya ke atas menjadi bilangan bulat dengan fungsi atap $\lceil \cdot \rceil$, lalu membaginya dua lagi, sehingga **berat yang ditagihkan** adalah

$$\hat{w} = \frac{\lceil 2 \max(w, w_s) \rceil}{2} \tag{11.4}$$

dan **harga dasarnya** adalah

$$p = r_s \cdot \hat{w} \tag{11.5}$$

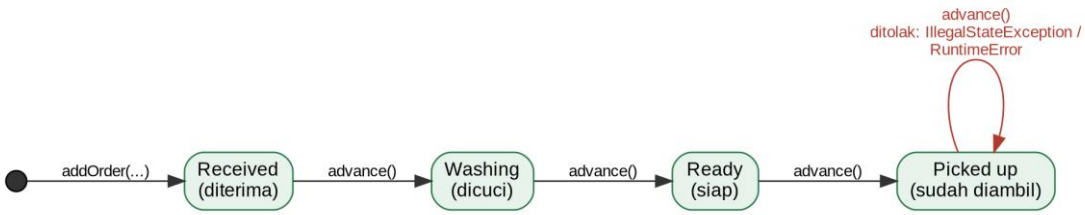
Pesanan kilat lebih mahal separuhnya:

$$p_{\text{ex}} = \lfloor \frac{3p}{2} \rfloor \tag{11.6}$$

dengan $\lfloor \cdot \rfloor$ membulatkan ke bawah; karena setiap harga dasar di sini merupakan kelipatan 500, $3p/2$ selalu bilangan bulat dan pembulatan itu tidak pernah benar-benar terjadi, tetapi program memakai aritmetika bilangan bulat dan rumus itu menyatakan dengan tepat apa yang dihitungnya.

Mari kita hitung ketiga pesanan uji mandiri dengan tangan, satu langkah per baris. Sari membawa 4,2 kg Wash & Fold. Menurut Persamaan (11.4), $\max(4.2,3) = 4.2$; $2 \times 4.2 = 8.4$; $\lceil 8.4 \rceil = 9$; $9/2 = 4.5$ kg. Menurut Persamaan (11.5), $p = 7000 \times 4.5 = 31500$, jadi Rp31,500. Budi membawa 2,6 kg Wash, Dry & Iron, kilat. $\max(2.6,3) = 3$; $\lceil 6 \rceil / 2 = 3$ kg; $p = 10000 \times 3 = 30000$; menurut Persamaan (11.6), $p_{\text{ex}} = \lfloor 90000/2 \rfloor = 45000$, jadi Rp45,000. Citra membawa 1,3 kg untuk disetrika: $\max(1.3,2) = 2$ kg, $p = 5000 \times 2 = 10000$. Totalnya $31500 + 45000 + 10000 = 86500$, Rp86,500 — angka yang akan diperiksa oleh uji mandiri dan *unit test*. (Program dan rumus menulis desimal dengan titik, mengikuti Java dan Python; teks bab ini memakai koma.)

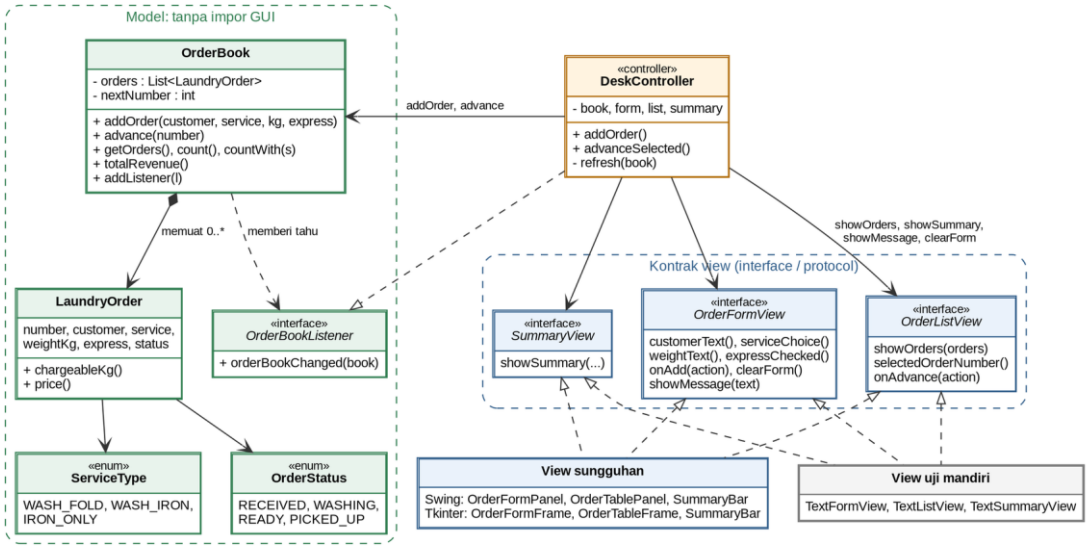
Status-status itu membentuk sebuah **mesin keadaan** (*state machine*) kecil, digambar di **Gambar 11.11**. Sebuah pesanan lahir sebagai *Received* melalui `addOrder`; setiap `advance()` menggesernya satu langkah ke kanan; pada keadaan *Picked up*, `advance()` ditolak dengan eksepsi dan pesanan tetap di tempatnya.



Gambar 11.11 Riwayat hidup sebuah pesanan binatu. Setiap `advance()` menggeser pesanan satu keadaan ke kanan; pesanan yang sudah diambil menolak untuk maju.

11.9.2 Rancangannya

Gambar 11.12 memperlihatkan setiap kelas aplikasi ini. Ia mengikuti susunan mediator **Gambar 11.5**: model memberi tahu controller, dan controller memperbarui view, sehingga view tidak pernah memegang referensi ke model.



Gambar 11.12 Diagram kelas meja depan. Model (kiri) tidak mengimpor apa pun dari GUI. Controller mengimplementasikan `OrderBookListener`, memanggil model, dan memperbarui tiga kontrak view; view Swing atau Tkinter sungguhan dan view teks uji mandiri sama-sama mengimplementasikan kontrak-kontrak itu.

Di sebelah kiri adalah **model**: `OrderBook` menyimpan pesanan dan aturannya serta memberi tahu para `OrderBookListener`; `LaundryOrder` adalah satu pesanan, yang tahu cara menghitung berat tagihan dan harganya; `ServiceType` dan `OrderStatus` adalah enumerasi, himpunan tetap konstanta bernama (`enum` di Java, `Enum` di Python). Di tengah adalah **controller**, `DeskController`, yang mendengarkan model dan view. Di sebelah kanan ada tiga **kontrak view** — *interface* di Java, *protocol* di Python — yang menyatakan apa yang dibutuhkan controller dari formulir, dari daftar pesanan, dan dari baris ringkasan; di bawahnya, view sungguhan dan view teks uji mandiri mengimplementasikan kontrak-kontrak itu. Tidak ada satu pun di dalam model yang menunjuk ke luar model.

Setiap bagian jendela mendapat *interface* kecilnya sendiri, alih-alih satu `DeskView` besar, karena bagian-bagian itu berubah secara terpisah: televisi ruang tunggu dalam kisah kita hanya memerlukan daftar, tidak lebih. Tidak ada kelas yang semestinya dipaksa bergantung pada metode yang tidak dipakainya.

11.9.3 Modelnya

Kode 11.11 adalah model dalam Java. Ia tidak memiliki `main`; uji mandiri dan pemeriksaan-pemeriksaan yang memakainya.

Kode 11.11 `OrderBook.java`: model meja depan, dengan buku pesanan, satu pesanan, dua enumerasi, dan *interface* pendengar. Ia tidak mengimpor apa pun dari Swing.

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import java.util.Locale;

// Model meja depan Profitina Laundry: pesanan dan aturan-aturannya.
// Ia tidak mengimpor apa pun dari Swing dan dapat melayani halaman web atau aplikasi
ponsel.
public class OrderBook {
    public static final double MAX_KG = 50.0;
    private final List<LaundryOrder> orders = new ArrayList<>();
    private final List<OrderBookListener> listeners = new ArrayList<>();
    private int nextNumber = 1;

    public void addListener(OrderBookListener listener) {
        listeners.add(listener);
    }

    public void removeListener(OrderBookListener listener) {
        listeners.remove(listener);
    }

    private void fireChanged() {
        for (OrderBookListener listener : List.copyOf(listeners)) {
            listener.orderBookChanged(this);
        }
    }

    public LaundryOrder addOrder(String customer, ServiceType service,
                                 double weightKg, boolean express) {
        String name = customer == null ? "" : customer.strip();
        if (name.isEmpty()) {
            throw new IllegalArgumentException("customer name is required");
        }
        if (service == null) {
            throw new IllegalArgumentException("choose a service");
        }
        if (!(weightKg > 0 && weightKg <= MAX_KG)) {  // juga menolak NaN
            throw new IllegalArgumentException("weight must be above 0 and at most 50 kg");
        }
        LaundryOrder order = new LaundryOrder(nextNumber++, name, service, weightKg,
express);
        orders.add(order);
        fireChanged();
        return order;
    }
```

Kode 11.11 (Lanjutan) *OrderBook.java: model meja depan, dengan buku pesanan, satu pesanan, dua enumerasi, dan interface pendengar. Ia tidak mengimpor apa pun dari Swing.*

```
    public void advance(int number) {
        LaundryOrder order = find(number);
        if (order.getStatus() == OrderStatus.PICKED_UP) {
            throw new IllegalStateException("order #" + number + " was already picked up");
        }
        order.setStatus(order.getStatus().next());
        fireChanged();
    }

    public LaundryOrder find(int number) {
        for (LaundryOrder order : orders) {
            if (order.getNumber() == number) {
                return order;
            }
        }
        throw new IllegalArgumentException("no order #" + number);
    }

    public List<LaundryOrder> getOrders() {
        return Collections.unmodifiableList(orders);
    }

    public int count() {
        return orders.size();
    }

    public int countWith(OrderStatus status) {
        int n = 0;
        for (LaundryOrder order : orders) {
            if (order.getStatus() == status) {
                n++;
            }
        }
        return n;
    }

    public long totalRevenue() {
        long total = 0;
        for (LaundryOrder order : orders) {
            total += order.price();
        }
        return total;
    }
```

Kode 11.11 (Lanjutan) *OrderBook.java: model meja depan, dengan buku pesanan, satu pesanan, dua enumerasi, dan interface pendengar. Ia tidak mengimpor apa pun dari Swing.*

```
    public static String rupiah(long amount) {
        return String.format(Locale.US, "Rp%,d", amount);
    }
}

enum ServiceType {
    WASH_FOLD("Wash & Fold", 7000, 3.0),
    WASH_IRON("Wash, Dry & Iron", 10000, 3.0),
    IRON_ONLY("Iron Only", 5000, 2.0);

    private final String label;
    private final long ratePerKg;
    private final double minimumKg;

    ServiceType(String label, long ratePerKg, double minimumKg) {
        this.label = label;
        this.ratePerKg = ratePerKg;
        this.minimumKg = minimumKg;
    }

    long ratePerKg() {
        return ratePerKg;
    }

    double minimumKg() {
        return minimumKg;
    }

    static ServiceType fromLabel(String label) {
        for (ServiceType type : values()) {
            if (type.label.equals(label)) {
                return type;
            }
        }
        return null;
    }

    @Override
    public String toString() {
        return label;
    }
}

enum OrderStatus {
    RECEIVED("Received"), WASHING("Washing"), READY("Ready"), PICKED_UP("Picked up");
```

Kode 11.11 (Lanjutan) *OrderBook.java: model meja depan, dengan buku pesanan, satu pesanan, dua enumerasi, dan interface pendengar. Ia tidak mengimpor apa pun dari Swing.*

```
    private final String label;

    OrderStatus(String label) {
        this.label = label;
    }

    OrderStatus next() {
        return values()[ordinal() + 1];  // PICKED_UP tidak punya penerus: pemanggil memeriksa dulu
    }

    @Override
    public String toString() {
        return label;
    }
}

final class LaundryOrder {
    private final int number;
    private final String customer;
    private final ServiceType service;
    private final double weightKg;
    private final boolean express;
    private OrderStatus status = OrderStatus.RECEIVED;

    LaundryOrder(int number, String customer, ServiceType service,
                 double weightKg, boolean express) {
        this.number = number;
        this.customer = customer;
        this.service = service;
        this.weightKg = weightKg;
        this.express = express;
    }

    int getNumber() {
        return number;
    }

    String getCustomer() {
        return customer;
    }

    ServiceType getService() {
        return service;
    }
```

Kode 11.11 (Lanjutan) *OrderBook.java: model meja depan, dengan buku pesanan, satu pesanan, dua enumerasi, dan interface pendengar. Ia tidak mengimpor apa pun dari Swing.*

```
    double getWeightKg() {
        return weightKg;
    }

    boolean isExpress() {
        return express;
    }

    OrderStatus getStatus() {
        return status;
    }

    void setStatus(OrderStatus status) {  // package-private: hanya OrderBook yang
memanggilnya
        this.status = status;
    }

    // Minimal sebesar batas minimum layanan, dibulatkan ke atas ke setengah kilogram
berikutnya.
    double chargeableKg() {
        return Math.ceil(Math.max(weightKg, service.minimumKg()) * 2) / 2;
    }

    long price() {
        long base = Math.round(chargeableKg() * service.ratePerKg());
        return express ? base * 3 / 2 : base;  // kilat lebih mahal separuhnya
    }
}

interface OrderBookListener {
    void orderBookChanged(OrderBook book);
}
```

`addOrder` memperlihatkan urutan kerja yang semestinya diikuti setiap metode model: **validasi, ubah, beri tahu, kembalikan**. Ia menolak nama kosong (setelah spasinya dibuang), layanan yang tidak ada, dan berat di luar (0,50]. Uji beratnya ditulis `!(weightKg > 0 && weightKg <= MAX_KG)` alih-alih `weightKg <= 0 || weightKg > MAX_KG` yang lebih gamblang, dan perbedaannya penting: nilai khusus `NaN` (“*not a number*”, yang dengan senang hati dikembalikan `Double.parseDouble("NaN")`) membuat *setiap* perbandingan bernilai salah, sehingga uji yang gamblang akan meloloskan `NaN`, sedangkan uji kita menolaknya. Baru setelah semua pemeriksaan lolos, metode itu membuat pesanan, menambahkannya ke list, dan menembakkan *satu* peristiwa. Bila ada pemeriksaan yang gagal, eksepsi meninggalkan metode sebelum apa pun berubah dan sebelum siapa pun diberi tahu — sehingga pesanan yang ditolak tidak memakan nomor pesanan dan tidak menggambar ulang view mana pun.

`advance` mengikuti pola yang sama: cari pesanannya (atau gagal dengan `IllegalArgumentException`), tolak memajukan pesanan yang sudah diambil dengan `IllegalStateException`, ubah statusnya, beri tahu. Kedua tipe eksepsi itu menyatakan hal yang berbeda: `IllegalArgumentException` berarti “Anda menanyakan sesuatu yang tidak ada atau cacat bentuknya”, dan `IllegalStateException` berarti “permintaan Anda wajar, tetapi objeknya tidak berada dalam keadaan yang memungkinkan permintaan itu dikabulkan”. Controller menangkap keduanya secara terpisah.

`getOrders` mengembalikan `Collections.unmodifiableList(orders)`, sebuah *view* hanya-baca atas list internal (Bab 9). Siapa pun boleh membaca pesanan; tidak ada yang di luar `OrderBook` dapat menambah atau menghapus pesanan kecuali melalui `addOrder`, yang menegakkan aturan.

Setter status milik `LaundryOrder` bersifat *package-private* dan diberi keterangan "hanya `OrderBook` yang memanggilnya". `fireChanged` menelusuri `List.copyOf(listeners)`, sebuah potret keadaan, dengan alasan yang diberikan Subbab 11.5.2.

Di `LaundryOrder`, `chargeableKg` adalah Persamaan (11.4) dan `price` adalah Persamaan (11.5) disusul Persamaan (11.6). `ServiceType` menyimpan label, tarif, dan minimum setiap layanan di dalam konstanta enum itu sendiri, sehingga menambah layanan keempat cukup dengan satu baris; `fromLabel` mencari konstanta berdasarkan teks yang ditampilkan formulir. `OrderStatus.next()` mengembalikan konstanta berikutnya menurut posisinya (`values()[ordinal() + 1]`); untuk `PICKED_UP` hal itu akan melewati ujung larik, itulah sebabnya `advance` memeriksa lebih dulu.

Kode 11.12 adalah model dalam Python.

Kode 11.12 `order_book.py`: model meja depan dalam Python. Seperti kembarannya di Java, ia tidak mengimpor apa pun dari Tkinter.

```
"""order_book.py: model meja depan Profitina Laundry.

Ia tidak mengimpor apa pun dari Tkinter dan dapat melayani halaman web atau
aplikasi ponsel.
"""
import math
from enum import Enum

MAX_KG = 50.0


def rupiah(amount):
    return f"Rp{amount:,}"


class ServiceType(Enum):
    WASH_FOLD = ("Wash & Fold", 7000, 3.0)
    WASH_IRON = ("Wash, Dry & Iron", 10000, 3.0)
    IRON_ONLY = ("Iron Only", 5000, 2.0)

    def __init__(self, label, rate_per_kg, minimum_kg):
        self.label = label
        self.rate_per_kg = rate_per_kg
        self.minimum_kg = minimum_kg

    @classmethod
    def from_label(cls, label):
        for service in cls:
            if service.label == label:
                return service
        return None

    def __str__(self):
        return self.label


class OrderStatus(Enum):
    RECEIVED = "Received"
    WASHING = "Washing"
    READY = "Ready"
    PICKED_UP = "Picked up"

    def next(self):
        members = list(OrderStatus)
        return members[members.index(self) + 1]  # PICKED_UP tidak punya penerus
```

Kode 11.12 (Lanjutan) *order_book.py: model meja depan dalam Python. Seperti kembarannya di Java, ia tidak mengimpor apa pun dari Tkinter.*

```
    def __str__(self):
        return self.value


class LaundryOrder:
    def __init__(self, number, customer, service, weight_kg, express):
        self.number = number
        self.customer = customer
        self.service = service
        self.weight_kg = weight_kg
        self.express = express
        self.status = OrderStatus.RECEIVED  # hanya diubah oleh OrderBook

    def chargeable_kg(self):
        # Minimal sebesar batas minimum layanan, dibulatkan ke atas ke setengah kilogram
berikutnya.
        return math.ceil(max(self.weight_kg, self.service.minimum_kg) * 2) / 2

    def price(self):
        base = round(self.chargeable_kg() * self.service.rate_per_kg)
        return base * 3 // 2 if self.express else base  # kilat: lebih mahal separuhnya


class OrderBook:
    def __init__(self):
        self._orders = []
        self._listeners = []
        self._next_number = 1

    def add_listener(self, listener):
        self._listeners.append(listener)

    def remove_listener(self, listener):
        self._listeners.remove(listener)

    def _fire_changed(self):
        for listener in list(self._listeners):
            listener(self)
```

Kode 11.12 (Lanjutan) *order_book.py: model meja depan dalam Python. Seperti kembarannya di Java, ia tidak mengimpor apa pun dari Tkinter.*

```
    def add_order(self, customer, service, weight_kg, express):
        name = (customer or "").strip()
        if not name:
            raise ValueError("customer name is required")
        if service is None:
            raise ValueError("choose a service")
        if not (0 < weight_kg <= MAX_KG):  # juga menolak nan
            raise ValueError("weight must be above 0 and at most 50 kg")
        order = LaundryOrder(self._next_number, name, service, weight_kg, express)
        self._next_number += 1
        self._orders.append(order)
        self._fire_changed()
        return order

    def advance(self, number):
        order = self.find(number)
        if order.status is OrderStatus.PICKED_UP:
            raise RuntimeError(f"order #{number} was already picked up")
        order.status = order.status.next()
        self._fire_changed()

    def find(self, number):
        for order in self._orders:
            if order.number == number:
                return order
        raise ValueError(f"no order #{number}")

    def orders(self):
        return tuple(self._orders)  # potret keadaan yang hanya-baca

    def count(self):
        return len(self._orders)

    def count_with(self, status):
        return sum(1 for order in self._orders if order.status is status)

    def total_revenue(self):
        return sum(order.price() for order in self._orders)
```

Enum Python `ServiceType` memperlihatkan fitur yang patut diketahui: bila nilai sebuah anggota `Enum` berupa *tuple* dan kelasnya mendefinisikan `__init__`, Python memanggil `__init__` dengan elemen-elemen *tuple* itu, sehingga setiap anggota mendapat atribut `label`, `rate_per_kg`, dan `minimum_kg`. Uji berat `0 < weight_kg <= MAX_KG` adalah perbandingan berantai Python; `nan` gagal melewatinya dengan alasan yang sama seperti di Java. `orders()` mengembalikan sebuah `tuple`, yang tidak dapat diubah, sebagai *view* hanya-baca. `advance` melempar `RuntimeError` di tempat Java melempar `IllegalStateException`, karena Python tidak memiliki padanan yang persis; tipe eksepsi apa pun boleh, asalkan controller menangkap tipe yang sama. Dan `round(...)` di `price` aman walaupun Python membulatkan setengah tepat ke bilangan genap terdekat: tarif adalah kelipatan 1.000 dan berat tagihan kelipatan 0,5, sehingga hasil kalinya selalu bilangan bulat.

11.9.4 Controller dan uji mandiri

Kode 11.13 memuat ketiga *interface* view, controller, view-view teks, dan metode `main`.

Kode 11.13 `LaundryDesk.java`: kontrak-kontrak view, controller perantara, view-view teks uji mandiri, dan `main`.

```
import java.util.List;
import java.util.Locale;
import javax.swing.SwingUtilities;

public class LaundryDesk {
    public static void main(String[] args) {
        OrderBook book = new OrderBook();
        if (args.length > 0 && args[0].equals("--selftest")) {
            selfTest(book);
            return;
        }
        SwingUtilities.invokeLater(() -> {
            DeskWindow window = new DeskWindow("Profitina Laundry - Front Desk");
            new DeskController(book, window.form(), window.list(), window.summary());
            window.open();
        });
    }

    // Satu giliran kerja penuh di meja depan, diketik dan diklik oleh skrip, bukan orang.
    static void selfTest(OrderBook book) {
        TextFormView form = new TextFormView();
        TextListView list = new TextListView();
        TextSummaryView summary = new TextSummaryView();
        new DeskController(book, form, list, summary);

        form.fill("Sari Wulandari", "Wash & Fold", "4.2", false);
        form.clickAdd();
        form.fill("Budi", "Wash, Dry & Iron", "2,6", true);
        form.clickAdd();
        form.fill("   ", "Iron Only", "1.3", false);
        form.clickAdd();
        form.fill("Citra", "Iron Only", "abc", false);
        form.clickAdd();
        form.fill("Citra", "Iron Only", "75", false);
        form.clickAdd();
        form.fill("Citra", "Iron Only", "1.3", false);
        form.clickAdd();
        list.clickAdvance();
        list.select(2);
        for (int i = 0; i < 4; i++) {
            list.clickAdvance();
        }
```

Kode 11.13 (Lanjutan) *LaundryDesk.java: kontrak-kontrak view, controller perantara, view-view teks uji mandiri, dan main.*

```
        boolean ok = book.count() == 3
                && book.totalRevenue() == 86_500
                && book.find(2).getStatus() == OrderStatus.PICKED_UP
                && book.countWith(OrderStatus.RECEIVED) == 2;
        if (!ok) {
            throw new AssertionError("selftest failed");
        }
        System.out.println("selftest passed: " + book.count() + " orders, "
                + OrderBook.rupiah(book.totalRevenue()));
    }
}

// Apa yang dibutuhkan controller dari formulir pesanan, siapa pun yang menggambarnya.
interface OrderFormView {
    String customerText();

    String serviceChoice();

    String weightText();

    boolean expressChecked();

    void onAdd(Runnable action);

    void clearForm();

    void showMessage(String message);
}

// Apa yang dibutuhkan controller dari daftar pesanan.
interface OrderListView {
    void showOrders(List<LaundryOrder> orders);

    int selectedOrderNumber();  // 0 bila tidak ada yang dipilih

    void onAdvance(Runnable action);
}

// Apa yang dibutuhkan controller dari baris ringkasan.
interface SummaryView {
    void showSummary(int orders, int received, int ready, long revenue);
}
```

Kode 11.13 (Lanjutan) *LaundryDesk.java: kontrak-kontrak view, controller perantara, view-view teks uji mandiri, dan main.*

```
// Controller: sebuah mediator. Ia mengubah gestur menjadi panggilan ke model dan
// perubahan model menjadi pembaruan view; model dan view tak pernah berbicara langsung.
class DeskController {
    private final OrderBook book;
    private final OrderFormView form;
    private final OrderListView list;
    private final SummaryView summary;

    DeskController(OrderBook book, OrderFormView form, OrderListView list, SummaryView
summary) {
        this.book = book;
        this.form = form;
        this.list = list;
        this.summary = summary;
        form.onAdd(this::addOrder);
        list.onAdvance(this::advanceSelected);
        book.addListener(this::refresh);
        refresh(book);
    }

    private void refresh(OrderBook changed) {
        list.showOrders(changed.getOrders());
        summary.showSummary(changed.count(), changed.countWith(OrderStatus.RECEIVED),
                changed.countWith(OrderStatus.READY), changed.totalRevenue());
    }

    void addOrder() {
        // Controller mengubah teks menjadi angka; model menilai angka itu.
        String text = form.weightText().strip().replace(',', '.');
        double kg;
        try {
            kg = Double.parseDouble(text);
        } catch (NumberFormatException e) {
            form.showMessage("Weight must be a number, such as 4.2");
            return;
        }
        ServiceType service = ServiceType.fromLabel(form.serviceChoice());
        try {
            LaundryOrder order = book.addOrder(form.customerText(), service, kg,
                    form.expressChecked());
            form.clearForm();
            form.showMessage("Added order #" + order.getNumber() + " for "
                    + order.getCustomer() + ": " + OrderBook.rupiah(order.price()));
        } catch (IllegalArgumentException e) {
            form.showMessage("Not added: " + e.getMessage());
        }
    }
```

Kode 11.13 (Lanjutan) *LaundryDesk.java: kontrak-kontrak view, controller perantara, view-view teks uji mandiri, dan main.*

```
    void advanceSelected() {
        int number = list.selectedOrderNumber();
        if (number == 0) {
            form.showMessage("Select an order first.");
            return;
        }
        try {
            book.advance(number);
            form.showMessage("Order #" + number + " is now " +
book.find(number).getStatus());
        } catch (IllegalStateException e) {
            form.showMessage("Not changed: " + e.getMessage());
        }
    }
}

// View pengganti untuk uji mandiri: masing-masing mencetak apa yang akan ditampilkan
bagian jendelanya.
class TextFormView implements OrderFormView {
    private String customer = "";
    private String service = "Wash & Fold";
    private String weight = "";
    private boolean express;
    private Runnable addAction = () -> { };

    void fill(String customer, String service, String weight, boolean express) {
        this.customer = customer;
        this.service = service;
        this.weight = weight;
        this.express = express;
        System.out.printf("> type \"%s\", %s, \"%s\" kg%s%n",
                customer, service, weight, express ? ", Express" : "");
    }

    void clickAdd() {
        System.out.println("> click Add order");
        addAction.run();
    }

    @Override
    public String customerText() {
        return customer;
    }
```

Kode 11.13 (Lanjutan) *LaundryDesk.java: kontrak-kontrak view, controller perantara, view-view teks uji mandiri, dan main.*

```
    @Override
    public String serviceChoice() {
        return service;
    }

    @Override
    public String weightText() {
        return weight;
    }

    @Override
    public boolean expressChecked() {
        return express;
    }

    @Override
    public void onAdd(Runnable action) {
        addAction = action;
    }

    @Override
    public void clearForm() {
        customer = "";
        weight = "";
        express = false;
    }

    @Override
    public void showMessage(String message) {
        System.out.println("  [message] " + message);
    }
}

class TextListView implements OrderListView {
    private int selected;
    private Runnable advanceAction = () -> { };

    void select(int number) {
        selected = number;
        System.out.println("> select order #" + number);
    }
```

Kode 11.13 (Lanjutan) *LaundryDesk.java: kontrak-kontrak view, controller perantara, view-view teks uji mandiri, dan main.*

```
    void clickAdvance() {
        System.out.println("> click Advance status");
        advanceAction.run();
    }

    @Override
    public void showOrders(List<LaundryOrder> orders) {
        for (LaundryOrder o : orders) {
            System.out.println(String.format(Locale.US, "  [list] #%d %-15s %-17s %4.1f kg
%-3s %9s %s",
                    o.getNumber(), o.getCustomer(), o.getService(), o.getWeightKg(),
                    o.isExpress() ? "EX" : "", OrderBook.rupiah(o.price()),
o.getStatus()));
        }
    }

    @Override
    public int selectedOrderNumber() {
        return selected;
    }

    @Override
    public void onAdvance(Runnable action) {
        advanceAction = action;
    }
}

class TextSummaryView implements SummaryView {
    @Override
    public void showSummary(int orders, int received, int ready, long revenue) {
        String count = orders == 1 ? "1 order" : orders + " orders";
        System.out.println("  [summary] " + count + ", " + received + " received, "
                + ready + " ready, " + OrderBook.rupiah(revenue));
    }
}
```

Interface view itu kecil dan ditulis dari sudut pandang controller. `OrderFormView` menyediakan empat *pertanyaan* — apa isi kolom pelanggan, layanan mana yang dipilih, apa isi kolom berat, apakah *Express* dicentang — satu *pendaftaran*, `onAdd`, dan dua *perintah*, `clearForm` dan `showMessage`. `OrderListView` menampilkan pesanan, melaporkan pesanan mana yang dipilih (0 bila tidak ada), dan mendaftarkan aksi *Advance*. `SummaryView` menampilkan empat angka. Tidak satu pun metode ini menyebut Swing atau Tkinter.

Konstruktor **controller** mengerjakan seluruh penyambungan: ia mendaftarkan kedua aksinya pada view, mendaftarkan `refresh` pada model, dan memanggil `refresh` sekali untuk menggambar gambar pertama. `refresh` adalah gaya menarik yang sedang bekerja: apa pun yang berubah, ia memberikan kepada view daftar seluruh list saat ini dan kepada view ringkasan keempat angkanya, dihitung ulang dari model. Karena itu view tidak pernah perlu tahu *apa* yang berubah.

`addOrder` memperlihatkan pembagian kerja antara controller dan model dalam bentuknya yang paling murni. Tugas controller adalah mengubah *teks* menjadi *nilai*: ia membuang spasi dari teks berat, mengganti koma dengan titik, dan mengurai sebuah `double`; bila penguraian gagal, ia menampilkan pesan dan berhenti, karena string seperti `abc` sama sekali bukan berat dan model tidak semestinya melihatnya. Ia mengubah label layanan menjadi sebuah `ServiceType`. Lalu ia menyerahkan nilai-nilai itu kepada model dan membiarkan model menilainya: apakah 75 kg dapat diterima adalah aturan *bisnis*, dan aturan itu tinggal di model, tempat *unit test* dapat menjangkaunya. Bila model menolak, controller menampilkan penjelasan model itu sendiri

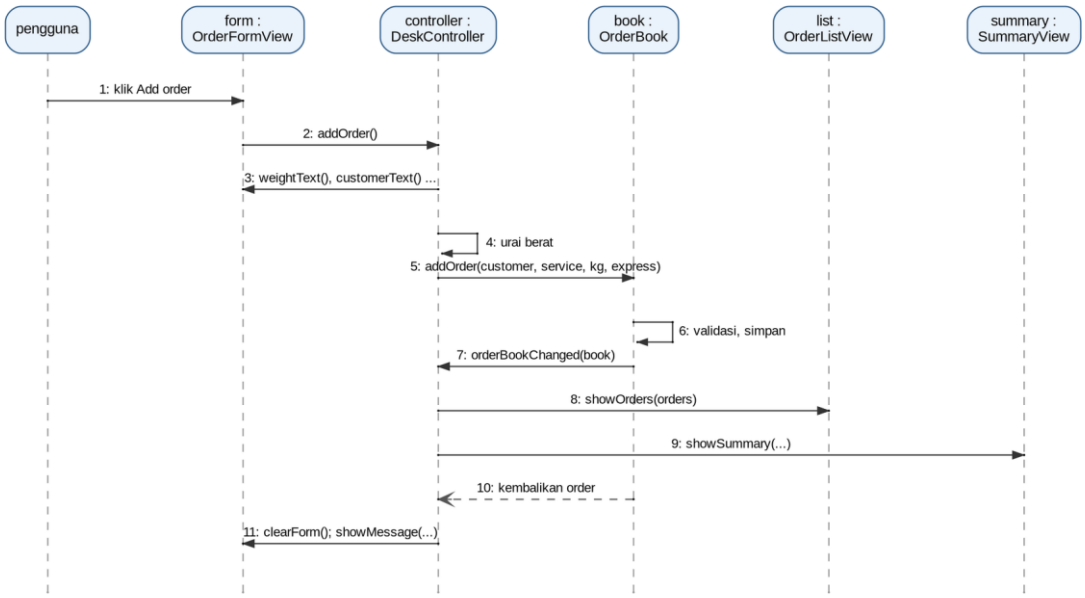
dengan awalan `Not added:`. Bila model menerima, controller mengosongkan formulir dan melaporkan nomor serta harganya. Controller tidak pernah menghitung harga dan tidak pernah memeriksa rentang berat. `advanceSelected` mengikuti pola yang sama untuk tombol kedua.

`main` menjalankan uji mandiri atau membangun jendela sungguhan di *event dispatch thread*. **Uji mandiri** adalah satu giliran kerja penuh di meja depan: ia mengetik enam pesanan, tiga di antaranya keliru dengan cara yang berbeda-beda, mengeklik *Advance status* sekali ketika belum ada yang dipilih, lalu memilih pesanan 2 dan memajukannya empat kali — satu kali lebih banyak dari yang diizinkan mesin keadaan. View-view teks mencetak apa yang akan ditampilkan jendela. `java LaundryDesk --selftest` mencetak:

```
  [summary] 0 orders, 0 received, 0 ready, Rp0
> type "Sari Wulandari", Wash & Fold, "4.2" kg
> click Add order
  [list] #1 Sari Wulandari  Wash & Fold        4.2 kg      Rp31,500 Received
  [summary] 1 order, 1 received, 0 ready, Rp31,500
  [message] Added order #1 for Sari Wulandari: Rp31,500
> type "Budi", Wash, Dry & Iron, "2,6" kg, Express
> click Add order
  [list] #1 Sari Wulandari  Wash & Fold        4.2 kg      Rp31,500 Received
  [list] #2 Budi            Wash, Dry & Iron   2.6 kg EX   Rp45,000 Received
  [summary] 2 orders, 2 received, 0 ready, Rp76,500
  [message] Added order #2 for Budi: Rp45,000
> type "   ", Iron Only, "1.3" kg
> click Add order
  [message] Not added: customer name is required
> type "Citra", Iron Only, "abc" kg
> click Add order
  [message] Weight must be a number, such as 4.2
> type "Citra", Iron Only, "75" kg
> click Add order
  [message] Not added: weight must be above 0 and at most 50 kg
> type "Citra", Iron Only, "1.3" kg
> click Add order
  [list] #1 Sari Wulandari  Wash & Fold        4.2 kg      Rp31,500 Received
  [list] #2 Budi            Wash, Dry & Iron   2.6 kg EX   Rp45,000 Received
  [list] #3 Citra           Iron Only          1.3 kg      Rp10,000 Received
  [summary] 3 orders, 3 received, 0 ready, Rp86,500
  [message] Added order #3 for Citra: Rp10,000
> click Advance status
  [message] Select an order first.
> select order #2
> click Advance status
  [list] #1 Sari Wulandari  Wash & Fold        4.2 kg      Rp31,500 Received
  [list] #2 Budi            Wash, Dry & Iron   2.6 kg EX   Rp45,000 Washing
  [list] #3 Citra           Iron Only          1.3 kg      Rp10,000 Received
  [summary] 3 orders, 2 received, 0 ready, Rp86,500
  [message] Order #2 is now Washing
> click Advance status
  [list] #1 Sari Wulandari  Wash & Fold        4.2 kg      Rp31,500 Received
  [list] #2 Budi            Wash, Dry & Iron   2.6 kg EX   Rp45,000 Ready
  [list] #3 Citra           Iron Only          1.3 kg      Rp10,000 Received
  [summary] 3 orders, 2 received, 1 ready, Rp86,500
  [message] Order #2 is now Ready
> click Advance status
  [list] #1 Sari Wulandari  Wash & Fold        4.2 kg      Rp31,500 Received
  [list] #2 Budi            Wash, Dry & Iron   2.6 kg EX   Rp45,000 Picked up
  [list] #3 Citra           Iron Only          1.3 kg      Rp10,000 Received
  [summary] 3 orders, 2 received, 0 ready, Rp86,500
  [message] Order #2 is now Picked up
> click Advance status
  [message] Not changed: order #2 was already picked up
selftest passed: 3 orders, Rp86,500
```

Setiap `> click Add order` yang sah disusul baris-baris daftar dan ringkasan — dicetak oleh `refresh` selama pemberitahuan model, *di dalam* `book.addOrder` — dan baru kemudian oleh pesan formulir. Ketiga pesanan yang ditolak hanya menghasilkan pesan: model tidak berubah, sehingga tak ada view yang menggambar ulang, dan tak ada nomor pesanan yang terpakai, sehingga pesanan Citra yang sah bernomor #3. Berat Budi diketik `2,6` dan dipahami sebagai 2,6. Keempat klik *Advance status* menggerakkan pesanan #2 melalui keadaan-keadaannya, dan hitungan di ringkasan ikut berubah — 2 *received* dan 1 *ready* selama pakaian Budi menunggu di rak, 0 *ready* setelah ia mengambilnya — sampai klik keempat ditolak dengan pesan dari model. Di akhir, uji mandiri memeriksa jumlah pesanan, pendapatan Rp86,500 yang tadi kita hitung dengan tangan, status pesanan #2, dan jumlah pesanan yang berstatus *received*.

Gambar 11.13 menggambar klik pertama yang berhasil pada **Add order** sebagai diagram sekuens, dan patut ditelusuri sambil membandingkannya dengan keluaran.



Gambar 11.13 Diagram sekuens sebuah klik pada *Add order*. Formulir melaporkan klik kepada controller, yang membaca formulir, mengurai berat, dan memanggil model; model memvalidasi, menyimpan, dan memberi tahu controller, yang menyegarkan daftar dan ringkasan; setelah model kembali, controller mengosongkan formulir dan menampilkan pesan.

Pesan 1–2 adalah gesturnya, 3–4 controller membaca dan mengurai formulir, 5–6 perintah dan validasi oleh model. Pesan 7–9, pemberitahuan dan penggambaran ulang, terjadi *di dalam* pesan 5, sebelum model kembali (10); pesan 11 memperbarui formulir.

11.9.5 View-view Swing

Kode 11.14 memuat view-view sungguhan: tiga komponen Swing, masing-masing mengimplementasikan satu *interface* view, dan kelas `DeskWindow` yang merakitnya.

Kode 11.14 `DeskWindow.java`: view-view Swing meja depan. `OrderFormPanel`, `OrderTablePanel`, dan `SummaryBar` mengimplementasikan ketiga kontrak view; `DeskWindow` menempatkannya dalam sebuah *frame*.

```
import java.awt.BorderLayout;
import java.awt.FlowLayout;
import java.awt.GridLayout;
import java.util.List;
import java.util.Locale;
import javax.swing.BorderFactory;
import javax.swing.JButton;
import javax.swing.JCheckBox;
import javax.swing.JComboBox;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTable;
import javax.swing.JTextField;
import javax.swing.ListSelectionModel;
import javax.swing.table.DefaultTableModel;

// View Swing meja depan, dirakit dalam satu jendela.
public class DeskWindow {
    private final JFrame frame;
    private final OrderFormPanel form = new OrderFormPanel();
    private final OrderTablePanel list = new OrderTablePanel();
    private final SummaryBar summary = new SummaryBar();

    public DeskWindow(String title) {
        frame = new JFrame(title);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.add(form, BorderLayout.NORTH);
        frame.add(list, BorderLayout.CENTER);
        frame.add(summary, BorderLayout.SOUTH);
        frame.setSize(760, 420);
    }

    OrderFormView form() {
        return form;
    }

    OrderListView list() {
        return list;
    }

    SummaryView summary() {
        return summary;
    }
```

Kode 11.14 (Lanjutan) *DeskWindow.java: view-view Swing meja depan. OrderFormPanel, OrderTablePanel, dan SummaryBar mengimplementasikan ketiga kontrak view; DeskWindow menempatkannya dalam sebuah frame.*

```
    void open() {
        frame.setLocationRelativeTo(null);
        frame.setVisible(true);
    }
}

class OrderFormPanel extends JPanel implements OrderFormView {
    private final JTextField customer = new JTextField(14);
    private final JComboBox<String> service = new JComboBox<>();
    private final JTextField weight = new JTextField(5);
    private final JCheckBox express = new JCheckBox("Express");
    private final JButton add = new JButton("Add order");
    private final JLabel message = new JLabel("Ready.");

    OrderFormPanel() {
        super(new BorderLayout());
        for (ServiceType type : ServiceType.values()) {
            service.addItem(type.toString());
        }
        // Dua baris: satu baris FlowLayout pada lebar ini akan menyembunyikan "Add order".
        JPanel rows = new JPanel(new GridLayout(2, 1));
        JPanel first = new JPanel(new FlowLayout(FlowLayout.LEFT));
        first.add(new JLabel("Customer"));
        first.add(customer);
        first.add(new JLabel("Service"));
        first.add(service);
        JPanel second = new JPanel(new FlowLayout(FlowLayout.LEFT));
        second.add(new JLabel("Weight (kg)"));
        second.add(weight);
        second.add(express);
        second.add(add);
        rows.add(first);
        rows.add(second);
        add(rows, BorderLayout.CENTER);
        message.setBorder(BorderFactory.createEmptyBorder(0, 10, 5, 10));
        add(message, BorderLayout.SOUTH);
    }

    @Override
    public String customerText() {
        return customer.getText();
    }
```

Kode 11.14 (Lanjutan) *DeskWindow.java: view-view Swing meja depan. OrderFormPanel, OrderTablePanel, dan SummaryBar mengimplementasikan ketiga kontrak view; DeskWindow menempatkannya dalam sebuah frame.*

```
    @Override
    public String serviceChoice() {
        return (String) service.getSelectedItem();
    }

    @Override
    public String weightText() {
        return weight.getText();
    }

    @Override
    public boolean expressChecked() {
        return express.isSelected();
    }

    @Override
    public void onAdd(Runnable action) {
        add.addActionListener(e -> action.run());
    }

    @Override
    public void clearForm() {
        customer.setText("");
        weight.setText("");
        express.setSelected(false);
        customer.requestFocusInWindow();
    }

    @Override
    public void showMessage(String text) {
        message.setText(text);
    }
}

class OrderTablePanel extends JPanel implements OrderListView {
    private static final String[] COLUMNS =
            {"No.", "Customer", "Service", "kg", "Express", "Price", "Status"};
    private final DefaultTableModel rows = new DefaultTableModel(COLUMNS, 0) {
        @Override
        public boolean isCellEditable(int row, int column) {
            return false;  // tabel hanya menampilkan; perubahan lewat controller
        }
    };
    private final JTable table = new JTable(rows);
    private final JButton advance = new JButton("Advance status");
```

Kode 11.14 (Lanjutan) *DeskWindow.java: view-view Swing meja depan. OrderFormPanel, OrderTablePanel, dan SummaryBar mengimplementasikan ketiga kontrak view; DeskWindow menempatkannya dalam sebuah frame.*

```
    OrderTablePanel() {
        super(new BorderLayout());
        table.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
        add(new JScrollPane(table), BorderLayout.CENTER);
        JPanel buttons = new JPanel(new FlowLayout(FlowLayout.RIGHT));
        buttons.add(advance);
        add(buttons, BorderLayout.SOUTH);
    }

    @Override
    public void showOrders(List<LaundryOrder> orders) {
        int keep = selectedOrderNumber();
        rows.setRowCount(0);
        for (LaundryOrder o : orders) {
            rows.addRow(new Object[] {o.getNumber(), o.getCustomer(), o.getService(),
                String.format(Locale.US, "%.1f", o.getWeightKg()),
                o.isExpress() ? "yes" : "", OrderBook.rupiah(o.price()), o.getStatus()});
            if (o.getNumber() == keep) {
                table.setRowSelectionInterval(rows.getRowCount() - 1, rows.getRowCount() -
1);
            }
        }
    }

    @Override
    public int selectedOrderNumber() {
        int row = table.getSelectedRow();
        return row < 0 ? 0 : (Integer) rows.getValueAt(row, 0);
    }

    @Override
    public void onAdvance(Runnable action) {
        advance.addActionListener(e -> action.run());
    }
}

class SummaryBar extends JLabel implements SummaryView {
    SummaryBar() {
        setBorder(BorderFactory.createEmptyBorder(5, 10, 5, 10));
    }
```

Kode 11.14 (Lanjutan) *DeskWindow.java: view-view Swing meja depan. OrderFormPanel, OrderTablePanel, dan SummaryBar mengimplementasikan ketiga kontrak view; DeskWindow menempatkannya dalam sebuah frame.*

```
    @Override
    public void showSummary(int orders, int received, int ready, long revenue) {
        String count = orders == 1 ? "1 order" : orders + " orders";
        setText(count + "  |  " + received + " received  |  " + ready
                + " ready for pickup  |  total " + OrderBook.rupiah(revenue));
    }
}
```

Setiap metode kelas-kelas ini entah membangun *widget* atau memindahkan data antara sebuah *widget* dan sebuah parameter; tidak ada satu `if` pun tentang bisnis di seluruh berkas. `OrderFormPanel` membangun dua baris ber-`FlowLayout` di dalam sebuah `GridLayout` dua baris: kolom pelanggan dan kotak layanan di baris pertama, kolom berat, kotak centang *Express*, dan tombol **Add order** di baris kedua, serta label pesan di bawahnya. Komentar di atas baris-baris itu mencatat pelajaran yang dipetik ketika membangun program ini: versi pertama menaruh

kedelapan komponen dalam *satu* baris `FlowLayout`. `FlowLayout` memindahkan komponen yang tidak muat ke baris baru, tetapi wilayah `NORTH` dari sebuah `BorderLayout` hanya memberi panel setinggi *satu* baris — sehingga pada lebar 760 piksel, tombol **Add order** berpindah ke baris kedua yang tidak pernah digambar. Program terkompilasi, uji mandiri lolos, dan tombolnya tak terlihat. Hanya melihat jendelanya yang mengungkapkan hal itu. Itulah peragaan paling jelas tentang apa yang dapat dan tidak dapat dilihat uji mandiri.

`OrderTablePanel` menampilkan pesanan dalam `JTable` yang `DefaultTableModel`-nya menolak penyuntingan: perubahan melalui controller. `showOrders` memulihkan pilihan sebelumnya setelah membangun ulang baris-barisnya, sehingga petugas dapat mengeklik *Advance status* berulang kali tanpa memilih ulang. `SummaryBar` adalah `JLabel` dengan margin.

Jalankan `java LaundryDesk` di komputer yang memiliki layar, dan jendelanya tampak seperti ini setelah ketiga pesanan sah dari uji mandiri ditambahkan, pesanan 2 dimajukan dua kali, dan petugas mengetik berat yang tidak sah untuk pelanggan keempat:

```
+-------------------------------------------------------------------------------------+
| Profitina Laundry - Front Desk                                              _ o x |
+-------------------------------------------------------------------------------------+
| Customer [Dewi          ]  Service [Iron Only       v]                              |
| Weight (kg) [abc  ]  [ ] Express  [ Add order ]                                     |
|  Weight must be a number, such as 4.2                                               |
| +-----+----------------+------------------+-----+---------+----------+-----------+ |
| | No. | Customer       | Service          | kg  | Express | Price    | Status    | |
| +-----+----------------+------------------+-----+---------+----------+-----------+ |
| | 1   | Sari Wulandari | Wash & Fold      | 4.2 |         | Rp31,500 | Received  | |
| |#2###|#Budi###########|#Wash, Dry & Iron#|#2.6#|#yes#####|#Rp45,000#|#Ready#####| |
| | 3   | Citra          | Iron Only        | 1.3 |         | Rp10,000 | Received  | |
| |     |                |                  |     |         |          |           | |
| +-----+----------------+------------------+-----+---------+----------+-----------+ |
|                                                                  [ Advance status ] |
| 3 orders  |  2 received  |  1 ready for pickup  |  total Rp86,500                  |
+-------------------------------------------------------------------------------------+
```

Jendela itu berukuran 760 kali 420 piksel dan terbuka di tengah layar. Di bagian atas ada formulir, dalam dua baris, dengan baris pesan di bawahnya; baris itu berbunyi *Ready.* ketika program dimulai. Tabel mengisi bagian tengah dan ikut membesar ketika jendela diperbesar; baris yang dipilih (digambar dengan `#` pada sketsa) disorot dengan warna pilihan sistem. Tombol **Advance status** berada di kanan bawah tabel, dan baris ringkasan membentang di sepanjang tepi bawah jendela.

11.9.6 Meja depan yang sama dalam Python

Di Python, controller dan view-view teks tinggal di `laundry_desk.py` (**Kode 11.15**) dan view-view Tkinter di modul terpisah, `tk_desk_views.py` (**Kode 11.16**). Pemisahan menjadi dua berkas itu disengaja: `laundry_desk.py` mengimpor modul Tkinter hanya di cabang `main` yang membuka jendela, sehingga uji mandiri tidak pernah memuat Tkinter sama sekali.

Kode 11.15 `laundry_desk.py`: *protocol* view, controller, view-view teks uji mandiri, dan `main`.

```
"""laundry_desk.py: meja depan Profitina Laundry sebagai model, view, dan controller."""
import sys
from typing import Callable, Protocol, Sequence

from order_book import LaundryOrder, OrderBook, OrderStatus, ServiceType, rupiah


class OrderFormView(Protocol):
    """Apa yang dibutuhkan controller dari formulir pesanan, siapa pun yang
menggambarnya."""
    def customer_text(self) -> str: ...
    def service_choice(self) -> str: ...
    def weight_text(self) -> str: ...
    def express_checked(self) -> bool: ...
    def on_add(self, action: Callable[[], None]) -> None: ...
    def clear_form(self) -> None: ...
    def show_message(self, message: str) -> None: ...


class OrderListView(Protocol):
    """Apa yang dibutuhkan controller dari daftar pesanan."""
    def show_orders(self, orders: Sequence[LaundryOrder]) -> None: ...
    def selected_order_number(self) -> int: ...  # 0 bila tidak ada yang dipilih
    def on_advance(self, action: Callable[[], None]) -> None: ...


class SummaryView(Protocol):
    """Apa yang dibutuhkan controller dari baris ringkasan."""
    def show_summary(self, orders: int, received: int, ready: int,
                     revenue: int) -> None: ...


class DeskController:
    """Controller: sebuah mediator. Ia mengubah gestur menjadi panggilan ke model
    dan perubahan model menjadi pembaruan view; model dan view tak pernah
    berbicara langsung."""
```

Kode 11.15 (Lanjutan) *laundry_desk.py: protocol view, controller, view-view teks uji mandiri, dan main.*

```
    def __init__(self, book: OrderBook, form: OrderFormView,
                 order_list: OrderListView, summary: SummaryView):
        self._book = book
        self._form = form
        self._list = order_list
        self._summary = summary
        form.on_add(self.add_order)
        order_list.on_advance(self.advance_selected)
        book.add_listener(self._refresh)
        self._refresh(book)

    def _refresh(self, changed):
        self._list.show_orders(changed.orders())
        self._summary.show_summary(changed.count(),
                                   changed.count_with(OrderStatus.RECEIVED),
                                   changed.count_with(OrderStatus.READY),
                                   changed.total_revenue())

    def add_order(self):
        # Controller mengubah teks menjadi angka; model menilai angka itu.
        text = self._form.weight_text().strip().replace(",", ".")
        try:
            kg = float(text)
        except ValueError:
            self._form.show_message("Weight must be a number, such as 4.2")
            return
        service = ServiceType.from_label(self._form.service_choice())
        try:
            order = self._book.add_order(self._form.customer_text(), service, kg,
                                         self._form.express_checked())
        except ValueError as e:
            self._form.show_message(f"Not added: {e}")
            return
        self._form.clear_form()
        self._form.show_message(f"Added order #{order.number} for "
                                f"{order.customer}: {rupiah(order.price())}")
```

Kode 11.15 (Lanjutan) *laundry_desk.py: protocol view, controller, view-view teks uji mandiri, dan main.*

```
    def advance_selected(self):
        number = self._list.selected_order_number()
        if number == 0:
            self._form.show_message("Select an order first.")
            return
        try:
            self._book.advance(number)
        except RuntimeError as e:
            self._form.show_message(f"Not changed: {e}")
            return
        self._form.show_message(f"Order #{number} is now {self._book.find(number).status}")


class TextFormView:
    """Formulir pengganti untuk uji mandiri: mencetak apa yang akan ditampilkan jendela."""

    def __init__(self):
        self._customer, self._service, self._weight = "", "Wash & Fold", ""
        self._express = False
        self._add = lambda: None

    def fill(self, customer, service, weight, express):
        self._customer, self._service = customer, service
        self._weight, self._express = weight, express
        print(f'> type "{customer}", {service}, "{weight}" kg'
              + (", Express" if express else ""))

    def click_add(self):
        print("> click Add order")
        self._add()

    def customer_text(self):
        return self._customer

    def service_choice(self):
        return self._service

    def weight_text(self):
        return self._weight

    def express_checked(self):
        return self._express

    def on_add(self, action):
        self._add = action
```

Kode 11.15 (Lanjutan) *laundry_desk.py: protocol view, controller, view-view teks uji mandiri, dan main.*

```
    def clear_form(self):
        self._customer, self._weight, self._express = "", "", False

    def show_message(self, message):
        print("  [message] " + message)


class TextListView:
    def __init__(self):
        self._selected = 0
        self._advance = lambda: None

    def select(self, number):
        self._selected = number
        print(f"> select order #{number}")

    def click_advance(self):
        print("> click Advance status")
        self._advance()

    def show_orders(self, orders):
        for o in orders:
            print(f"  [list] #{o.number} {o.customer:<15} {str(o.service):<17} "
                  f"{o.weight_kg:4.1f} kg {'EX' if o.express else '':<3} "
                  f"{rupiah(o.price()):>9} {o.status}")

    def selected_order_number(self):
        return self._selected

    def on_advance(self, action):
        self._advance = action


class TextSummaryView:
    def show_summary(self, orders, received, ready, revenue):
        count = "1 order" if orders == 1 else f"{orders} orders"
        print(f"  [summary] {count}, {received} received, {ready} ready, "
              f"{rupiah(revenue)}")


def self_test(book):
    """Satu giliran kerja penuh di meja depan, diketik dan diklik oleh skrip, bukan
orang."""
    form, order_list, summary = TextFormView(), TextListView(), TextSummaryView()
    DeskController(book, form, order_list, summary)
```

Kode 11.15 (Lanjutan) *laundry_desk.py: protocol view, controller, view-view teks uji mandiri, dan main.*

```
    form.fill("Sari Wulandari", "Wash & Fold", "4.2", False)
    form.click_add()
    form.fill("Budi", "Wash, Dry & Iron", "2,6", True)
    form.click_add()
    form.fill("   ", "Iron Only", "1.3", False)
    form.click_add()
    form.fill("Citra", "Iron Only", "abc", False)
    form.click_add()
    form.fill("Citra", "Iron Only", "75", False)
    form.click_add()
    form.fill("Citra", "Iron Only", "1.3", False)
    form.click_add()
    order_list.click_advance()
    order_list.select(2)
    for _ in range(4):
        order_list.click_advance()

    assert (book.count() == 3 and book.total_revenue() == 86_500
            and book.find(2).status is OrderStatus.PICKED_UP
            and book.count_with(OrderStatus.RECEIVED) == 2), "selftest failed"
    print(f"selftest passed: {book.count()} orders, {rupiah(book.total_revenue())}")


def main(args):
    book = OrderBook()
    if args[:1] == ["--selftest"]:
        self_test(book)
        return
    from tk_desk_views import TkDeskWindow  # GUI dimuat hanya bila diperlukan
    window = TkDeskWindow("Profitina Laundry - Front Desk")
    DeskController(book, window.form, window.list, window.summary)
    window.open()


if __name__ == "__main__":
    main(sys.argv[1:])
```

`python3 laundry_desk.py --selftest` mencetak transkrip yang sama dengan Java, baris demi baris:

```
  [summary] 0 orders, 0 received, 0 ready, Rp0
> type "Sari Wulandari", Wash & Fold, "4.2" kg
> click Add order
  [list] #1 Sari Wulandari  Wash & Fold        4.2 kg      Rp31,500 Received
  [summary] 1 order, 1 received, 0 ready, Rp31,500
  [message] Added order #1 for Sari Wulandari: Rp31,500
> type "Budi", Wash, Dry & Iron, "2,6" kg, Express
> click Add order
  [list] #1 Sari Wulandari  Wash & Fold        4.2 kg      Rp31,500 Received
  [list] #2 Budi            Wash, Dry & Iron   2.6 kg EX   Rp45,000 Received
  [summary] 2 orders, 2 received, 0 ready, Rp76,500
  [message] Added order #2 for Budi: Rp45,000
> type "   ", Iron Only, "1.3" kg
> click Add order
  [message] Not added: customer name is required
> type "Citra", Iron Only, "abc" kg
> click Add order
  [message] Weight must be a number, such as 4.2
> type "Citra", Iron Only, "75" kg
> click Add order
  [message] Not added: weight must be above 0 and at most 50 kg
> type "Citra", Iron Only, "1.3" kg
> click Add order
  [list] #1 Sari Wulandari  Wash & Fold        4.2 kg      Rp31,500 Received
  [list] #2 Budi            Wash, Dry & Iron   2.6 kg EX   Rp45,000 Received
  [list] #3 Citra           Iron Only          1.3 kg      Rp10,000 Received
  [summary] 3 orders, 3 received, 0 ready, Rp86,500
  [message] Added order #3 for Citra: Rp10,000
> click Advance status
  [message] Select an order first.
> select order #2
> click Advance status
  [list] #1 Sari Wulandari  Wash & Fold        4.2 kg      Rp31,500 Received
  [list] #2 Budi            Wash, Dry & Iron   2.6 kg EX   Rp45,000 Washing
  [list] #3 Citra           Iron Only          1.3 kg      Rp10,000 Received
  [summary] 3 orders, 2 received, 0 ready, Rp86,500
  [message] Order #2 is now Washing
> click Advance status
  [list] #1 Sari Wulandari  Wash & Fold        4.2 kg      Rp31,500 Received
  [list] #2 Budi            Wash, Dry & Iron   2.6 kg EX   Rp45,000 Ready
  [list] #3 Citra           Iron Only          1.3 kg      Rp10,000 Received
  [summary] 3 orders, 2 received, 1 ready, Rp86,500
  [message] Order #2 is now Ready
> click Advance status
  [list] #1 Sari Wulandari  Wash & Fold        4.2 kg      Rp31,500 Received
  [list] #2 Budi            Wash, Dry & Iron   2.6 kg EX   Rp45,000 Picked up
  [list] #3 Citra           Iron Only          1.3 kg      Rp10,000 Received
  [summary] 3 orders, 2 received, 0 ready, Rp86,500
  [message] Order #2 is now Picked up
> click Advance status
  [message] Not changed: order #2 was already picked up
selftest passed: 3 orders, Rp86,500
```

Kontrak-kontrak view ditulis sebagai ***protocol***, `class OrderFormView(Protocol)`, dari modul `typing` [10]. Sebuah *protocol* mendaftar tanda tangan metode; kelas apa pun yang memiliki metode-metode itu *adalah* sebuah `OrderFormView` tanpa perlu menyatakannya — ia menuliskan *duck typing* Python. Python tidak memeriksa *protocol* saat program berjalan; pemeriksa statis seperti *mypy* yang melakukannya, dan pembaca dapat melihat sekilas apa yang diharapkan controller. Selebihnya, controller adalah terjemahan controller Java: `float(text)` melempar `ValueError` untuk teks yang bukan angka, `except ValueError as e` menangkap hal itu maupun penolakan dari model, dan `except RuntimeError` menangkap penolakan untuk memajukan pesanan yang sudah diambil.

Kode 11.16 `tk_desk_views.py`: view-view Tkinter meja depan.

```
"""tk_desk_views.py: view Tkinter meja depan, dirakit dalam satu jendela."""
import tkinter as tk
from tkinter import ttk

from order_book import ServiceType, rupiah


class OrderFormFrame(ttk.Frame):
    def __init__(self, parent):
        super().__init__(parent, padding=5)
        self._customer = tk.StringVar()
        self._service = tk.StringVar(value=ServiceType.WASH_FOLD.label)
        self._weight = tk.StringVar()
        self._express = tk.BooleanVar()
        self._message = tk.StringVar(value="Ready.")
        ttk.Label(self, text="Customer").grid(row=0, column=0, sticky="w")
        self._customer_entry = ttk.Entry(self, textvariable=self._customer, width=18)
        self._customer_entry.grid(row=0, column=1, sticky="w")
        ttk.Label(self, text="Service").grid(row=0, column=2, sticky="w", padx=(10, 0))
        ttk.Combobox(self, textvariable=self._service, state="readonly", width=16,
                     values=[s.label for s in ServiceType]).grid(row=0, column=3)
        ttk.Label(self, text="Weight (kg)").grid(row=1, column=0, sticky="w")
        ttk.Entry(self, textvariable=self._weight, width=6).grid(row=1, column=1,
sticky="w")
        ttk.Checkbutton(self, text="Express", variable=self._express).grid(
            row=1, column=2, sticky="w", padx=(10, 0))
        self._add = ttk.Button(self, text="Add order")
        self._add.grid(row=1, column=3, sticky="e")
        ttk.Label(self, textvariable=self._message).grid(
            row=2, column=0, columnspan=4, sticky="w", pady=(5, 0))

    def customer_text(self):
        return self._customer.get()

    def service_choice(self):
        return self._service.get()

    def weight_text(self):
        return self._weight.get()

    def express_checked(self):
        return self._express.get()

    def on_add(self, action):
        self._add.config(command=action)
```

Kode 11.16 (Lanjutan) *tk_desk_views.py: view-view Tkinter meja depan.*

```
    def clear_form(self):
        self._customer.set("")
        self._weight.set("")
        self._express.set(False)
        self._customer_entry.focus_set()

    def show_message(self, message):
        self._message.set(message)


class OrderTableFrame(ttk.Frame):
    WIDTHS = {"No.": 50, "Customer": 140, "Service": 140, "kg": 60,
              "Express": 70, "Price": 100, "Status": 100}

    def __init__(self, parent):
        super().__init__(parent, padding=5)
        self._table = ttk.Treeview(self, columns=tuple(self.WIDTHS), show="headings",
                                   selectmode="browse", height=10)
        for name, width in self.WIDTHS.items():
            self._table.heading(name, text=name)
            self._table.column(name, width=width)
        self._table.pack(fill="both", expand=True)
        self._advance = ttk.Button(self, text="Advance status")
        self._advance.pack(anchor="e", pady=(5, 0))

    def show_orders(self, orders):
        keep = self.selected_order_number()
        self._table.delete(*self._table.get_children())
        for o in orders:
            # Nomor pesanan menjadi id baris, sehingga pilihan bertahan setelah penyegaran.
            self._table.insert("", "end", iid=str(o.number), values=(
                o.number, o.customer, str(o.service), f"{o.weight_kg:.1f}",
                "yes" if o.express else "", rupiah(o.price()), str(o.status)))
        if keep:
            self._table.selection_set(str(keep))

    def selected_order_number(self):
        selected = self._table.selection()
        return int(selected[0]) if selected else 0

    def on_advance(self, action):
        self._advance.config(command=action)
```

Kode 11.16 (Lanjutan) *tk_desk_views.py: view-view Tkinter meja depan.*

```
class SummaryBar(ttk.Label):
    def __init__(self, parent):
        super().__init__(parent, padding=(10, 5))

    def show_summary(self, orders, received, ready, revenue):
        count = "1 order" if orders == 1 else f"{orders} orders"
        self.config(text=f"{count}  |  {received} received  |  {ready} ready "
                         f"for pickup  |  total {rupiah(revenue)}")


class TkDeskWindow:
    def __init__(self, title):
        self._root = tk.Tk()
        self._root.title(title)
        self.form = OrderFormFrame(self._root)
        self.form.pack(fill="x")
        self.list = OrderTableFrame(self._root)
        self.list.pack(fill="both", expand=True)
        self.summary = SummaryBar(self._root)
        self.summary.pack(fill="x")

    def open(self):
        self._root.mainloop()
```

View-view Tkinter memakai *widget* bertema dari `tkinter.ttk`, yang tampak asli di setiap sistem operasi. Formulirnya berupa kisi tiga baris. Setiap masukan terhubung ke variabel Tkinter — `StringVar` untuk teks, `BooleanVar` untuk kotak centang — sehingga pertanyaan-pertanyaan kontrak view cukup memanggil `get()`. Layanan dipilih dari `Combobox` hanya-baca yang berisi label-label layanan. Daftar pesanan adalah `ttk.Treeview` dengan `show="headings"`, yang mengubah *widget* pohon itu menjadi tabel; *item id* setiap baris adalah nomor pesanan sebagai string, sehingga `showOrders` dapat memulihkan pilihan dengan `selection_set(str(keep))` dan `selected_order_number` dapat membaca kembali nomornya dengan `int(...)`. Jendelanya bertata letak sama dengan jendela Swing: formulir di atas bersama baris pesan, tabel di tengah dengan **Advance status** di kanan bawahnya, dan baris ringkasan di dasar.

11.10 Unit Test untuk Model Meja Depan

Uji mandiri meja depan adalah sebuah skenario; modelnya juga layak mendapat *unit test* yang sungguhan, yang memeriksa setiap aturan secara terpisah. **Kode 11.17** memeriksa dua puluh fakta dengan Java biasa.

Kode 11.17 `OrderBookCheck.java`: dua puluh pemeriksaan terhadap model meja depan, termasuk harga, penolakan, mesin keadaan, list hanya-baca, dan penghentian langganan.

```
import java.util.Objects;

public class OrderBookCheck {
    private static int passed;
    private static int failed;

    public static void main(String[] args) {
        OrderBook book = new OrderBook();
        int[] events = {0};
        OrderBookListener counter = b -> events[0]++;
        book.addListener(counter);

        LaundryOrder sari = book.addOrder("  Sari Wulandari ", ServiceType.WASH_FOLD, 4.2,
false);
        LaundryOrder budi = book.addOrder("Budi", ServiceType.WASH_IRON, 2.6, true);
        LaundryOrder citra = book.addOrder("Citra", ServiceType.IRON_ONLY, 1.3, false);
        check("name is trimmed", "Sari Wulandari", sari.getCustomer());
        check("4.2 kg is charged as 4.5 kg", 4.5, sari.chargeableKg());
        check("wash and fold 4.2 kg", 31_500L, sari.price());
        check("minimum 3 kg, then express x 3/2", 45_000L, budi.price());
        check("iron only has a 2 kg minimum", 10_000L, citra.price());
        check("numbers count up", 3, citra.getNumber());
        check("total revenue", 86_500L, book.totalRevenue());
        check("one event per order", 3, events[0]);

        expectRefused("blank name", () -> book.addOrder("   ", ServiceType.WASH_FOLD, 3,
false));
        expectRefused("zero weight", () -> book.addOrder("Dewi", ServiceType.WASH_FOLD, 0,
false));
        expectRefused("too heavy", () -> book.addOrder("Dewi", ServiceType.WASH_FOLD, 50.5,
false));
        expectRefused("not a number", () -> book.addOrder("Dewi", ServiceType.WASH_FOLD,
Double.NaN, false));
        expectRefused("no service", () -> book.addOrder("Dewi", null, 3, false));
        check("refused orders change nothing", 3, book.count());
        check("refused orders fire no event", 3, events[0]);

        for (int i = 0; i < 3; i++) {
            book.advance(2);
        }
        check("three steps reach Picked up", OrderStatus.PICKED_UP, budi.getStatus());
        try {
            book.advance(2);
            check("a fourth step is refused", "exception", "no exception");
        } catch (IllegalStateException e) {
            check("a fourth step is refused", "order #2 was already picked up",
e.getMessage());
        }
        check("counting by status", 2, book.countWith(OrderStatus.RECEIVED));
```

Kode 11.17 (Lanjutan) *OrderBookCheck.java: dua puluh pemeriksaan terhadap model meja depan, termasuk harga, penolakan, mesin keadaan, list hanya-baca, dan penghentian langganan.*

```
        try {
            book.getOrders().clear();
            check("the list is read-only", "exception", "no exception");
        } catch (UnsupportedOperationException e) {
            check("the list is read-only", 3, book.count());
        }

        book.removeListener(counter);
        book.advance(1);
        check("a removed listener hears nothing", 6, events[0]);

        System.out.println(passed + " of " + (passed + failed) + " checks passed");
        if (failed > 0) {
            System.exit(1);
        }
    }

    static void expectRefused(String name, Runnable action) {
        try {
            action.run();
            check(name + " is refused", "exception", "no exception");
        } catch (IllegalArgumentException e) {
            check(name + " is refused", true, true);
        }
    }

    static void check(String name, Object expected, Object actual) {
        if (Objects.equals(expected, actual)) {
            passed++;
            System.out.println("PASS " + name);
        } else {
            failed++;
            System.out.println("FAIL " + name + ": expected " + expected + ", got " +
actual);
        }
    }
}
```

`java OrderBookCheck` mencetak:

```
PASS name is trimmed
PASS 4.2 kg is charged as 4.5 kg
PASS wash and fold 4.2 kg
PASS minimum 3 kg, then express x 3/2
PASS iron only has a 2 kg minimum
PASS numbers count up
PASS total revenue
PASS one event per order
PASS blank name is refused
PASS zero weight is refused
PASS too heavy is refused
PASS not a number is refused
PASS no service is refused
PASS refused orders change nothing
PASS refused orders fire no event
PASS three steps reach Picked up
PASS a fourth step is refused
PASS counting by status
PASS the list is read-only
PASS a removed listener hears nothing
20 of 20 checks passed
```

Pemeriksaan-pemeriksaan itu mengikuti aturan Subbab 11.9.1 satu per satu. Harga-harganya adalah ketiga harga yang kita hitung dengan tangan. Kelima penolakan mencakup `Double.NaN`, yang akan lolos dari uji rentang yang gamblang. Dua pemeriksaan memastikan bahwa pesanan yang ditolak tidak mengubah apa pun dan tidak menembakkan apa pun. Mesin keadaan **Gambar 11.11** ditelusuri sampai ujungnya dan satu langkah lebih jauh. List hanya-baca diuji dengan mencoba `clear()`-nya, yang harus melempar `UnsupportedOperationException` dan menyisakan tiga pesanan. Dan pemeriksaan terakhir mencabut pendengar pencacah dan memastikan bahwa ia tidak mendengar apa-apa lagi: hitungan peristiwa tetap 6, dari tiga pesanan dan tiga kali memajukan. `expectRefused` menerima aksi yang diuji sebagai lambda `Runnable`, sehingga satu metode pembantu melayani kelima penolakan.

Kode 11.18 adalah versi Python dengan `unittest`.

Kode 11.18 `test_order_book.py`: *unit test* model meja depan dengan `unittest`, memakai `subTest` untuk kelima penolakan.

```
"""test_order_book.py: unit test untuk model meja depan, tanpa jendela sama sekali."""
import unittest

from order_book import OrderBook, OrderStatus, ServiceType


class OrderBookTest(unittest.TestCase):
    def setUp(self):
        self.book = OrderBook()
        self.events = 0
        self.book.add_listener(self.count_event)
        self.sari = self.book.add_order("  Sari Wulandari ", ServiceType.WASH_FOLD, 4.2,
False)
        self.budi = self.book.add_order("Budi", ServiceType.WASH_IRON, 2.6, True)
        self.citra = self.book.add_order("Citra", ServiceType.IRON_ONLY, 1.3, False)

    def count_event(self, book):
        self.events += 1

    def test_prices_follow_the_rules(self):
        self.assertEqual(self.sari.customer, "Sari Wulandari")
        self.assertEqual(self.sari.chargeable_kg(), 4.5)
        self.assertEqual(self.sari.price(), 31_500)
        self.assertEqual(self.budi.price(), 45_000)   # minimum 3 kg, lalu x 3/2
        self.assertEqual(self.citra.price(), 10_000)  # minimum 2 kg
        self.assertEqual(self.book.total_revenue(), 86_500)
        self.assertEqual(self.events, 3)

    def test_bad_orders_are_refused_and_change_nothing(self):
        bad = [("   ", ServiceType.WASH_FOLD, 3), ("Dewi", ServiceType.WASH_FOLD, 0),
               ("Dewi", ServiceType.WASH_FOLD, 50.5),
               ("Dewi", ServiceType.WASH_FOLD, float("nan")), ("Dewi", None, 3)]
        for customer, service, kg in bad:
            with self.subTest(customer=customer, service=service, kg=kg):
                with self.assertRaises(ValueError):
                    self.book.add_order(customer, service, kg, False)
        self.assertEqual(self.book.count(), 3)
        self.assertEqual(self.events, 3)
```

Kode 11.18 (Lanjutan) *test_order_book.py: unit test model meja depan dengan unittest, memakai subTest untuk kelima penolakan.*

```
    def test_status_moves_forward_then_stops(self):
        for _ in range(3):
            self.book.advance(2)
        self.assertIs(self.budi.status, OrderStatus.PICKED_UP)
        with self.assertRaises(RuntimeError) as caught:
            self.book.advance(2)
        self.assertEqual(str(caught.exception), "order #2 was already picked up")
        self.assertEqual(self.book.count_with(OrderStatus.RECEIVED), 2)

    def test_orders_are_a_read_only_snapshot(self):
        with self.assertRaises(AttributeError):
            self.book.orders().append(None)  # tuple tidak punya append

    def test_removed_listener_hears_nothing(self):
        self.book.remove_listener(self.count_event)
        self.book.advance(1)
        self.assertEqual(self.events, 3)


if __name__ == "__main__":
    unittest.main()
```

`python3 -m unittest -v test_order_book` melaporkan:

```
test_bad_orders_are_refused_and_change_nothing
(test_order_book.OrderBookTest.test_bad_orders_are_refused_and_change_nothing) ... ok
test_orders_are_a_read_only_snapshot
(test_order_book.OrderBookTest.test_orders_are_a_read_only_snapshot) ... ok
test_prices_follow_the_rules (test_order_book.OrderBookTest.test_prices_follow_the_rules)
... ok
test_removed_listener_hears_nothing
(test_order_book.OrderBookTest.test_removed_listener_hears_nothing) ... ok
test_status_moves_forward_then_stops
(test_order_book.OrderBookTest.test_status_moves_forward_then_stops) ... ok

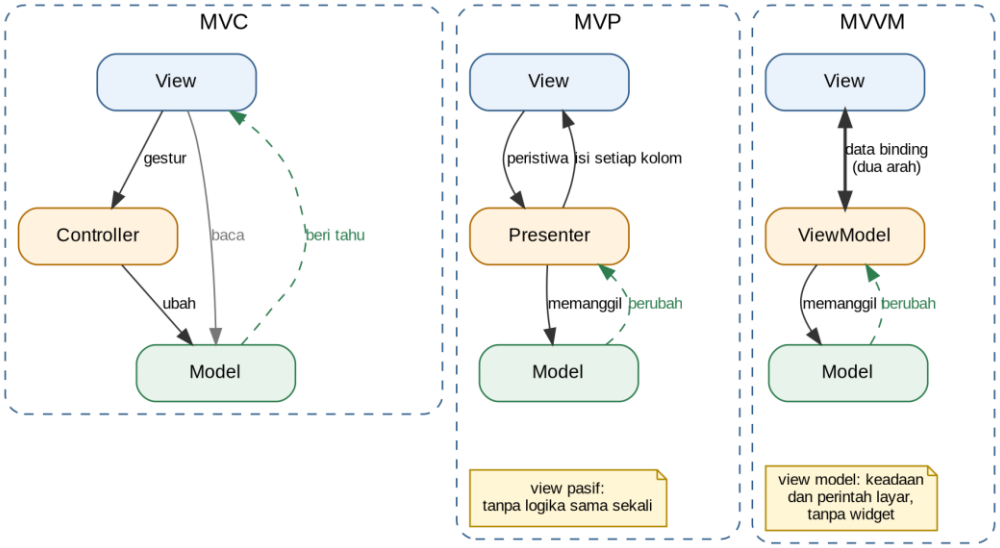
----------------------------------------------------------------------
Ran 5 tests in 0.003s

OK
```

Dua fitur `unittest` muncul di sini untuk pertama kalinya. `self.subTest(...)` di dalam sebuah gelung mengubah satu metode pengujian menjadi beberapa kasus yang dilaporkan: bila pesanan buruk yang ketiga keliru diterima, laporan kegagalan akan menyebut tepat `customer`, `service`, dan `kg` itu, dan gelung akan berlanjut dengan yang keempat. Dan `setUp` membangun ketiga pesanan yang sama sebelum setiap pengujian, sehingga setiap pengujian terbaca seperti cerita pendek yang dimulai dari toko yang sudah dikenal. Pendengarnya adalah *metode terikat*, `self.count_event`, bukan lambda, dan itu bukan kebetulan: untuk menghentikan langganan seorang pendengar Anda harus menyerahkan objek yang *sama* kepada `remove_listener`, dan `self.count_event` menghasilkan metode terikat yang setara setiap kali Anda menuliskannya, sedangkan dua lambda dengan badan yang sama adalah dua objek yang berbeda. Subbab 11.13 kembali ke jebakan ini.

11.11 Variasi: MVP dan MVVM

MVC memiliki banyak keturunan, dan dua di antaranya begitu tersebar luas sehingga Anda patut mengenalinya dari namanya. **Gambar 11.14** menggambar ketiganya berdampingan.



Gambar 11.14 MVC dan dua keturunannya. Dalam MVC, view mengamati model dan mengirim gestur ke controller. Dalam MVP, view pasif melaporkan peristiwa kepada *presenter*, yang mengisi setiap kolom view. Dalam MVVM, view diikat, dua arah, pada sebuah *view model* yang memegang keadaan dan perintah layar.

Model–View–Presenter (MVP) dideskripsikan oleh Taligent pada tahun 1996 [13] dan dipopulerkan untuk keperluan pengujian oleh artikel Michael Feathers, "The Humble Dialog Box" [9]. Dalam MVP view bersifat *pasif*: ia tidak memegang referensi ke model sama sekali, dan tidak memuat logika apa pun, bahkan logika tentang apa yang ditampilkan. ***Presenter*** menerima peristiwa dari view, berbicara dengan model, lalu *mengisi setiap kolom view* melalui sebuah *interface*. Bila ini terdengar akrab, memang seharusnya begitu: meja depan binatu kita, secara ketat, adalah MVP. `DeskController` adalah sebuah *presenter*; *interface* view adalah *interface* view pasif; dan view-view teks uji mandiri persis merupakan "view palsu" yang menjadi alasan diciptakannya MVP. Nama-nama di bab ini mengikuti kata yang lebih umum, *controller*, karena perbedaannya hanya soal kadar: seberapa banyak yang diketahui view. Pada papan tombol, view mengamati model secara langsung — MVC klasik; pada meja depan, view tidak tahu apa-apa — MVP.

Model–View–ViewModel (MVVM) diperkenalkan Microsoft untuk *toolkit* WPF-nya pada tahun 2005 [14] dan kini menjadi struktur baku banyak *framework* ponsel dan web. Ia menambahkan sebuah ***view model***: objek yang memegang *keadaan layar* — teks di setiap kolom, apakah sebuah tombol aktif, pesan yang harus ditampilkan — sebagai properti yang dapat diamati, dan menyediakan perintah-perintah layar sebagai metode. View dihubungkan dengan *view model* melalui ***data binding*** (pengikatan data): pernyataan, dibuat sekali, bahwa sebuah kolom teks dan sebuah properti selalu memegang nilai yang sama, dua arah, dijaga tetap selaras oleh pengamat di bawahnya. *View model* tidak pernah menyentuh *widget*, sehingga dapat diuji seperti model. Tkinter menyuguhkan sedikit rasanya: sebuah `Entry` yang dibuat dengan `textvariable=var` terikat dua arah pada sebuah `StringVar`, dan `var.trace_add("write", callback)` memungkinkan kode mengamatinya. Soal Latihan 11.10 membangun MVVM mini dengan properti *observable* buatan sendiri, dan memperlihatkan formulir yang menghitung pratinjau harga serta mengaktifkan tombol **Add order** sambil pengguna mengetik.

Untuk program-program di buku ini, pilihannya kurang penting dibandingkan kesamaan ketiganya: aturan tinggal di model yang tidak tahu apa-apa tentang layar, dan logikanya dapat

diuji tanpa jendela. Esai Martin Fowler *GUI Architectures* [15] adalah sejarah singkat terbaik tentang bagaimana varian-varian ini lahir dan apa yang masing-masing coba perbaiki.

11.12 Profitina dalam Praktik

Binatu adalah bengkel latihan kita; produk Profitina yang sesungguhnya memperlihatkan gagasan yang sama dalam skala lebih besar. Profitina adalah platform visibilitas AI: ia memindai situs web sebuah merek dan mengukur seberapa siap situs itu untuk dikutip oleh mesin penjawab AI, disiplin yang disebut *generative engine optimization* (GEO), lalu membantu memperbaiki apa yang ditemukannya. Aplikasi webnya ditulis dalam Python dengan *framework* FastAPI, dan antarmuka penggunanya berjalan di peramban.

Contoh MVC yang paling jelas dalam produk itu adalah **dasbor**. Ketika pengguna masuk, dasbor menampilkan, dalam satu layar, pemberitahuan tentang balasan, suka, *repost*, dan pesan langsung, status umpan balik yang telah dikirim pengguna, pembelian PROFIT, kanal-kanal yang diikuti pengguna beserta penghitung unggahan baru, serta peringkat dan aktivitas pengguna. Setiap panel itu adalah sebuah *view*, dan masing-masing merupakan view atas *model* yang berbeda: unggahan komunitas, alur umpan balik, buku besar PROFIT, langganan kanal. Dasbor tidak memiliki satu pun data itu. Ia dirakit oleh satu panggilan, `GET /api/me/dashboard`, yang mengumpulkan dari model-model apa yang diperlukan view lalu mengembalikannya; setiap butir di layar dapat diklik dan membawa ke bagian yang memilikinya. Itulah Persamaan (11.1) yang sedang bekerja: dasbor adalah fungsi dari keadaan model-model, dan dasbor yang menyimpan salinan hitungannya sendiri, cepat atau lambat, akan menampilkan hitungan yang berselisih dengan kanal yang diringkasnya.

Pola Observer juga tampak jelas. Umpan komunitas waktu-nyata memakai ***Server-Sent Events***: peramban berlangganan sekali pada sebuah aliran, dan *server* mendorong pesan setiap kali ada unggahan baru — sebuah subjek yang memberi tahu pendengarnya melintasi jaringan. Pengunggahan bersifat *optimistis*: unggahan baru langsung muncul di view, dan bila pengiriman gagal teksnya dipulihkan — pengecualian yang disengaja dan dibatalkan dengan cermat terhadap prinsip "model lebih dulu".

Dan strategi pengujian Subbab 11.8 dan 11.10 adalah aturan rilis Profitina. Rangkaian pengujian otomatisnya yang lengkap — jauh lebih dari seribu pengujian, sebagian besar memeriksa model dan penangan permintaan tanpa peramban apa pun, ditambah penjaga untuk antarmuka JavaScript — harus lolos sebelum sebuah rilis diizinkan. Hal itu hanya praktis karena logikanya tinggal di tempat yang dapat dijangkau pengujian yang cepat. Seperti di mana pun dalam Profitina, skor dan saran yang ditampilkan view semacam itu adalah nasihat: platform menjelaskan apa yang diukurnya, dan pengguna yang memutuskan apa yang akan dilakukan.

11.13 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya

1. Aturan bisnis di dalam *listener* tombol. Harga, berat minimum, atau pemeriksaan keabsahan dihitung di dalam `actionPerformed` atau `command` Tkinter. *Hindari:* badan *listener* semestinya satu baris yang memanggil controller; aturannya milik model, tempat *unit test* dapat menjangkaunya (Subbab 11.2).

2. Model yang mengimpor *toolkit* GUI. Satu `import javax.swing.JOptionPane` di dalam model "sekadar untuk menampilkan galat" mengikat model itu pada layar untuk selamanya. *Hindari:*

model melempar eksepsi atau mengembalikan hasil; controller yang menentukan cara menampilkannya. Periksalah dengan mencari `swing`, `awt`, atau `tkinter` di berkas model.

3. Lupa menggambar gambar pertama. Pengamat diberi tahu tentang *perubahan*; saat mulai, belum ada yang berubah, sehingga view tetap kosong sampai klik pertama. *Hindari:* controller (atau view) memanggil metode penyegarnya sekali setelah penyambungan, seperti yang dilakukan setiap controller di bab ini.

4. Memberi tahu sebelum memvalidasi, atau memberi tahu sekali per *field*. Model yang menembakkan peristiwa lalu melempar eksepsi meninggalkan view yang menampilkan nilai yang tidak pernah ada; model yang menembakkan tiga peristiwa untuk satu operasi melipattigakan penggambaran ulang (Persamaan (11.2)). *Hindari:* validasi, ubah, beri tahu sekali, kembalikan.

5. Mengubah daftar pendengar ketika sedang memberi tahu. Pendengar yang berhenti berlangganan saat diberi tahu membuat gelung `for` biasa melompati pendengar berikutnya di kedua bahasa, diam-diam (Soal Latihan 11.3). *Hindari:* telusuri salinan list-nya (`List.copyOf`, `list(...)`), atau pakai `PropertyChangeSupport`, yang melakukannya untuk Anda.

6. Mendaftarkan lambda yang tidak dapat dicabut langganannya. `model.add_listener(lambda b: ...)` yang kemudian disusul `model.remove_listener(lambda b: ...)` gagal, karena kedua lambda itu objek yang berbeda: Python melempar `ValueError`, dan `remove` milik Java diam-diam tidak menemukan apa pun untuk dihapus. View yang sudah ditutup tetapi tidak pernah dicabut langganannya tetap hidup, dan terus menggambar ulang, selama modelnya hidup — masalah *lapsed listener* (pendengar yang terlupakan). *Hindari:* simpan referensi ke pendengar yang Anda daftarkan, atau daftarkan metode terikat, dan cabut langganannya ketika view ditutup.

7. Membuat komponen Swing di luar *event dispatch thread*, atau di mesin tanpa layar. Swing berulir tunggal: bangun dan ubah *widget* hanya di dalam `SwingUtilities.invokeLater` [8]. Di komputer tanpa layar, `JFrame` pertama akan gagal; menjalankan `java CounterMvc` di mesin yang membangun buku ini menghasilkan:

```
Exception in thread "AWT-EventQueue-0" java.awt.HeadlessException:
No X11 DISPLAY variable was set,
or no headful library support was found,
but this program performed an operation which requires it.
```

Hindari: jauhkan logika dari jendela dan ujilah dengan `--selftest` serta *unit test*, yang tidak pernah membuat jendela. Padanannya di Python pada mesin tanpa Tkinter adalah `ModuleNotFoundError: No module named 'tkinter'`, yang dihindari dengan cara yang sama — dan dengan mengimpor `tkinter` hanya di dalam view.

8. Lambda di dalam gelung yang menangkap variabel gelung (Python). `command=lambda: controller.press(key)` di dalam `for key in "123..."` membuat setiap tombol menekan tombol *terakhir*. *Hindari:* tangkap nilai saat itu dengan argumen bawaan, `lambda k=key: controller.press(k)` (Subbab 11.7.5).

9. Percaya bahwa uji mandiri membuktikan jendela sudah benar. Uji mandiri menjalankan model dan controller; ia tidak dapat melihat bahwa sebuah tombol menghilang dari tepi panel (Subbab 11.9.5) atau bahwa sebuah label bertuliskan *vwxy*, bukan *wxyz*. *Hindari:* jaga view bebas logika, hasilkan labelnya dari tabel milik model sendiri, dan lihatlah jendelanya setelah setiap perubahan padanya.

10. Menamai sebuah kelas `View` lalu membiarkannya mengerjakan tugas controller. Kelas bernama `View` yang mengimplementasikan `ActionListener` dan memanggil model di dalam `actionPerformed` adalah view *sekaligus* controller, apa pun namanya. *Hindari:* tanyakan kepada setiap metode "apakah ia menggambar, memutuskan, atau menerjemahkan?", lalu pindahkan ke kelas yang tugasnya memang itu.

11.14 Ringkasan Bab

- Program yang menaruh *widget*, aturan bisnis, dan data dalam satu kelas tidak dapat menampilkan datanya dua kali, mengubah aturannya dengan aman, diuji tanpa layar, atau pindah ke antarmuka baru. **MVC** memecahnya menjadi **model** (data dan aturan, tanpa impor GUI), **view** (penyajian, tanpa aturan), dan **controller** (penerjemahan gestur menjadi operasi model).
- Idealnya view adalah fungsi dari keadaan model, $V = R(S)$ (Persamaan (11.1)); dengan begitu beberapa view dari satu model tidak mungkin berselisih (**Gambar 11.4**), dan model dapat diuji sendirian.
- Pola **Observer** memungkinkan model memberi tahu view tanpa mengenalnya: subjek menyimpan daftar pengamat dan memanggil masing-masing pada setiap perubahan. `Observable` milik Java sudah usang; pakailah `PropertyChangeSupport`, yang menembakkan peristiwa bernama dengan nilai lama dan baru serta melewatkan yang bukan perubahan. Di Python, list berisi *callable* sudah cukup. Pemberitahuan bersifat sinkron dan berbiaya $m \cdot k$ panggilan (Persamaan (11.2)), jadi beri tahu **sekali per perubahan logis**; telusuri salinan daftar pendengar.
- Pengamat dapat **didorong** nilai barunya atau **menarik**-nya dari model; menarik cocok untuk model besar dengan banyak view yang berbeda.
- Controller dapat membiarkan view mengamati model secara langsung (papan tombol) atau bertindak sebagai **mediator** di antara keduanya (meja binatu, **Gambar 11.5**), yang juga disebut MVP.
- Berikan setiap view sebuah ***interface*** (Java) atau ***protocol*** (Python); dengan begitu view teks dapat menggantikan jendela dalam mode `--selftest` yang menjalankan model dan controller tanpa layar, dan ***unit test*** dapat memeriksa setiap aturan model dalam hitungan milidetik.
- Metode model mengikuti urutan **validasi, ubah, beri tahu, kembalikan**; controller mengubah teks menjadi nilai, dan model menilai nilai-nilai itu.
- **MVP** membuat view pasif dan *presenter* bertanggung jawab atas setiap kolom; **MVVM** mengikat view pada *view model* berisi properti yang dapat diamati. Ketiganya menjauhkan aturan dari jendela.

11.15 Soal Latihan

Soal Latihan 11.1 (peran yang mana?). Untuk setiap potongan kode berikut dari sebuah program binatu, tentukan apakah ia milik model, view, atau controller, dan berikan alasan dalam satu kalimat. (a) `price = Math.round(chargeableKg() * ratePerKg)`. (b) `table.setRowSelectionInterval(row, row)`. (c) `double kg = Double.parseDouble(weightField.getText().replace(',', '.'))`. (d) `if (status == OrderStatus.PICKED_UP) throw new IllegalStateException(...)`. (e) `summary.setText(count + " orders")`. (f) `if (key == '0') model.acceptWord()`. (g) `listeners.add(listener)`.

Soal Latihan 11.2 (view kedua). Tambahkan pada program mahasiswa di Subbab 11.4 sebuah view kedua yang mencetak setiap mahasiswa sebagai satu baris nilai yang dipisahkan koma, nomor urut lebih dulu (`10,Robert`). Biarkan controller memegang sebanyak apa pun view dan memperbarui semuanya. Jangan mengubah kelas `Student`.

Soal Latihan 11.3 (pendengar yang pergi). Tiga pendengar didaftarkan dengan urutan ini: `log`, yang mencetak `log`; `once`, yang mencetak `once` lalu menghapus dirinya dari list; dan `display`, yang mencetak `display`. Subjek memberi tahu mereka dengan gelung `for` biasa atas list itu sendiri (bukan salinannya), dua kali. Ramalkan keluarannya di Java (dengan `ArrayList` dan `for` yang disempurnakan) dan di Python, lalu jalankan keduanya dan jelaskan apa yang Anda lihat.

Soal Latihan 11.4 (tak berubah, tak ada peristiwa). Di **Kode 11.3**, `setLoads(2)` tidak menembakkan peristiwa. (a) Objek mana yang membuat keputusan itu, dan di bagian mana keluaran program Anda dapat melihatnya? (b) Apa yang akan terjadi bila dua model saling mengamati dan menyalin setiap nilai baru ke dirinya sendiri, dan tak satu pun menahan yang bukan perubahan? (c) Mengapa penting bahwa `setLoads` memvalidasi *sebelum* memanggil `firePropertyChange`?

Soal Latihan 11.5 (aturan untuk pencacah). Binatu ingin pencacah di Subbab 11.6 menghitung muatan di mesin cuci, yang paling banyak menampung 12. Ubahlah program itu agar hitungannya tidak pernah di bawah 0 atau di atas 12. Di kelas mana aturan itu diletakkan? Apa yang semestinya ditampilkan view ketika *Up* diklik pada angka 12? Perluas uji mandiri untuk membuktikan aturan itu.

Soal Latihan 11.6 (*bug* label). Dalam program papan tombol asli, tombol 9 diberi label *vwxy* sementara pencariannya memakai *wxyz*. (a) Mengapa tidak ada *unit test* model yang dapat menemukannya? (b) Bagaimana **Kode 11.7** membuat *bug* itu mustahil? (c) Huruf apa yang tidak dapat dilihat pengguna program asli di tombol mana pun?

Soal Latihan 11.7 (membatalkan sebuah kata). Perluas papan tombol agar menekan `C` ketika belum ada angka yang diketik menghapus kata terakhir yang telah diterima. Ubahlah model, bukan controller; tambahkan *unit test* ke kedua program pengujian; dan pastikan model tetap menembakkan tepat satu peristiwa per perubahan sungguhan.

Soal Latihan 11.8 (layar ruang tunggu). Tulislah view baru atas `OrderBook` binatu untuk televisi di ruang tunggu: ia hanya menampilkan pesanan yang *Ready*, sebagai `#2 Budi`, dan menggambar ulang hanya bila daftar pesanan yang siap itu benar-benar berubah. Jangan mengubah `OrderBook.java` atau `order_book.py`.

Soal Latihan 11.9 (membatalkan pesanan). Binatu menginginkan tombol **Cancel order** yang menghapus sebuah pesanan, tetapi hanya selama statusnya masih *Received*. Uraikan setiap perubahan yang akan Anda buat — di model, controller, *interface* view, view Swing dan Tkinter, view teks, dan pengujian — dan tulislah metode model yang baru dalam kedua bahasa.

Soal Latihan 11.10 (MVVM mini). Tulislah kelas `Property` generik yang dapat diamati dengan `get`, `set` (yang hanya memberi tahu pada perubahan sungguhan), `observe`, dan `bind` (yang juga langsung memanggil pengamat satu kali). Pakailah kelas itu untuk menulis *view model* formulir pesanan, terbatas pada Wash & Fold, dengan properti masukan `customer`, `weightText`, dan `express` serta properti keluaran `preview` (misalnya `4.5 kg charged, Rp31,500`) dan `canAdd`. Tampilkan keluarannya ketika pengguna mengetik nama, berat, mencentang *Express*, lalu mengetik berat yang keliru.

Soal Latihan 11.11 (diagram sekuens). Gambarlah diagram sekuens untuk sebuah klik pada **Advance status** ketika pesanan 2 dipilih dan berstatus *Washing*, dengan gaya **Gambar 11.13**. Lalu gambarlah lagi untuk kasus pesanan 2 sudah diambil. Berapa pesan yang dikirim model dalam masing-masing kasus?

Soal Latihan 11.12 (versi web). Profitina Laundry membuka cabang kedua dan menginginkan meja depan sebagai halaman web. Periksalah setiap kelas di Subbab 11.9 dan tentukan apakah kelas itu dapat dipertahankan tanpa perubahan, harus diubah, atau harus diganti. Keputusan rancangan tunggal apa yang membuat sebagian besar jawabannya "tanpa perubahan"?

11.16 Penyelesaian Soal Latihan

Penyelesaian 11.1. (a) *Model*: menghitung harga adalah aturan bisnis. (b) *View*: baris mana yang disorot adalah soal penyajian. (c) *Controller*: mengubah teks yang diketik pengguna menjadi angka adalah penerjemahan gestur menjadi nilai; model semestinya menerima sebuah `double`, tidak pernah sebuah kolom teks. (d) *Model*: siklus hidup pesanan adalah aturan bisnis, ditegakkan di tempat keadaannya tinggal. (e) *View*: memformat hitungan menjadi kata-kata di sebuah label adalah penyajian. (f) *Controller*: bahwa tombol 0 berarti "terima katanya" adalah fakta tentang papan tombol ini, bukan tentang pesan. (g) *Model* (sisi subjek dari Observer): model menyimpan pendengarnya, walaupun ia tidak tahu apa-apa tentang siapa mereka.

Penyelesaian 11.2. **Kode 11.19** dan **Kode 11.20** memberi setiap view satu metode saja, sehingga "sebuah view" cukup berarti "sesuatu yang dapat menampilkan nama dan nomor urut". Controller menyimpan list berisi view-view itu.

Kode 11.19 `StudentTwoViews.java`: controller dengan sebanyak apa pun view; `StudentView` yang asli dan sebuah view baru yang dipisahkan koma. Ia memakai `Student` dan `StudentView` dari `StudentMvcDemo.java`.

```
import java.util.ArrayList;
import java.util.List;

public class StudentTwoViews {
    public static void main(String[] args) {
        Student model = new Student();
        model.setName("Robert");
        model.setRollNo("10");
        MultiViewController controller = new MultiViewController(model);
        controller.addView(new StudentView()::printStudentDetails);
        controller.addView(new CsvStudentView());
        controller.updateViews();
        controller.setStudentName("John");
        controller.updateViews();
    }
}

// View apa pun untuk mahasiswa: satu metode, jadi lambda atau referensi metode pun cukup.
interface StudentDisplay {
    void show(String name, String rollNo);
}

// View kedua dari model yang sama: satu baris, siap untuk lembar kerja.
class CsvStudentView implements StudentDisplay {
    @Override
    public void show(String name, String rollNo) {
        System.out.println(rollNo + "," + name);
    }
}

class MultiViewController {
    private final Student model;
    private final List<StudentDisplay> views = new ArrayList<>();

    MultiViewController(Student model) {
        this.model = model;
    }

    void addView(StudentDisplay view) {
        views.add(view);
    }

    void setStudentName(String name) {
        model.setName(name);
    }
```

Kode 11.19 (Lanjutan) *StudentTwoViews.java: controller dengan sebanyak apa pun view; StudentView yang asli dan sebuah view baru yang dipisahkan koma. Ia memakai Student dan StudentView dari StudentMvcDemo.java.*

```
    void updateViews() {
        for (StudentDisplay view : views) {
            view.show(model.getName(), model.getRollNo());
        }
    }
}
```

```
Student:
Name: Robert
Roll No: 10
10,Robert
Student:
Name: John
Roll No: 10
10,John
```

Di Java, *interface* `StudentDisplay` hanya memiliki satu metode abstrak, sehingga ia *interface* fungsional, dan view asli dapat disesuaikan dengannya melalui referensi metode `new StudentView()::printStudentDetails` — kelas lama dipakai ulang tanpa diubah. `Student` tidak disentuh, sesuai permintaan.

Kode 11.20 `student_two_views.py`: kedua view yang sama dalam Python, mengimpor model dan view pertama dari `student_mvc_demo.py`.

```python
"""student_two_views.py: satu model mahasiswa, dua view, satu controller."""
from student_mvc_demo import Student, StudentView


class CsvStudentView:
    """View kedua dari model yang sama: satu baris, siap untuk lembar kerja."""

    def show(self, name, roll_no):
        print(f"{roll_no},{name}")


class MultiViewController:
    def __init__(self, model):
        self._model = model
        self._views = []  # setiap callable yang menerima (name, roll_no) adalah view

    def add_view(self, view):
        self._views.append(view)

    def set_student_name(self, name):
        self._model.name = name

    def update_views(self):
        for view in self._views:
            view(self._model.name, self._model.roll_no)


def main():
    model = Student(name="Robert", roll_no="10")
    controller = MultiViewController(model)
    controller.add_view(StudentView().print_student_details)
    controller.add_view(CsvStudentView().show)
    controller.update_views()
    controller.set_student_name("John")
    controller.update_views()


if __name__ == "__main__":
    main()
```

```
Student:
Name: Robert
Roll No: 10
10,Robert
Student:
Name: John
Roll No: 10
10,John
```

Di Python sebuah view adalah *callable* apa pun yang menerima `(name, roll_no)`, sehingga controller menyimpan metode terikat: `StudentView().print_student_details` dan `CsvStudentView().show`. Tak satu pun versi mengubah model: model seorang mahasiswa tidak peduli ada berapa cara untuk mencetaknya.

Penyelesaian 11.3. Kedua bahasa mencetak hal yang sama, dan tak satu pun mengeluh:

```
round 1
log
once
round 2
log
display
```

Pada putaran 1, `display` terlewati. `for` yang disempurnakan di Java atas sebuah `ArrayList` memakai iterator yang menyimpan *kursor*, yaitu indeks elemen berikutnya. Setelah `log` (indeks 0) dan `once` (indeks 1), kursornya bernilai 2; `once` menghapus dirinya, sehingga list kini berukuran 2, dan `hasNext()` milik iterator menguji `cursor != size`, mendapati `2 != 2` bernilai salah, lalu mengakhiri gelung — tanpa `ConcurrentModificationException` yang akan muncul bila penghapusan terjadi di bagian lain list. Iterator list Python juga berjalan berdasarkan indeks: setelah indeks 1 ia mencoba indeks 2, yang tidak lagi ada di list dua elemen itu, lalu berhenti. Pada putaran 2 `once` sudah tiada dan kedua pendengar yang tersisa berjalan. *Bug* ini sunyi, dan bergantung pada posisi pendengar yang pergi — jenis yang paling buruk. Obatnya adalah memberi tahu dari sebuah salinan: `for (Runnable l : List.copyOf(listeners))` atau `for listener in list(listeners):`, seperti yang dilakukan setiap subjek di bab ini.

Penyelesaian 11.4. (a) `PropertyChangeSupport.firePropertyChange` membandingkan nilai lama dan nilai baru peristiwa dan tidak menembakkan apa pun bila keduanya sama; di keluaran, judul `setLoads(2)` langsung disusul judul berikutnya, tanpa baris pengamat di antaranya. (b) Pingpong tanpa akhir: A memberi tahu B, B menyalin nilainya dan memberi tahu A, A menyalinnya kembali, dan seterusnya sampai tumpukan panggilan meluap (`StackOverflowError`, di Python `RecursionError`). Menahan yang bukan perubahan menghentikannya setelah satu putaran. (c) Bila model memberi tahu lebih dulu lalu melempar eksepsi, para pengamat sudah telanjur menampilkan -1, nilai yang tidak pernah benar-benar diterima model; memvalidasi lebih dulu menjamin bahwa pengamat hanya pernah melihat keadaan yang sah.

Penyelesaian 11.5. Aturan itu milik **model**, karena ia fakta tentang mesin cuci, berlaku bagaimanapun hitungannya ditampilkan:

```
    static final int MAX_LOADS = 12;

    void setValue(int newValue) {
        if (newValue < 0 || newValue > MAX_LOADS) {
            return;                    // atau lempar eksepsi, dan biarkan controller
    menjelaskan
        }
        if (newValue == value) {
            return;
        }
        value = newValue;
        for (IntConsumer listener : List.copyOf(listeners)) {
            listener.accept(value);
        }
    }
```

Klik pada *Up* di angka 12 kini tidak mengubah apa pun dan tidak memberi tahu siapa pun, sehingga view tetap menampilkan 12. Bila binatu menginginkan pesan (“mesinnya penuh”), `setValue` sebaiknya melempar eksepsi, dan controller menangkapnya lalu meminta view menampilkan pesan itu. Uji mandiri mendapat sebuah gelung yang mengeklik *Up* tiga belas kali dari 0 dan mengharapkan 12, serta satu gelung yang mengeklik *Down* di angka 0 dan mengharapkan 0. Python memerlukan pemeriksaan yang sama di `set_value`.

Penyelesaian 11.6. (a) Label-label itu ditulis dengan tangan di dalam view, sebagai literal string, dan model tidak pernah melihatnya: tabel pencarian model sudah benar, sehingga setiap pengujian model lolos. *Bug* yang hanya tinggal di view tidak terlihat oleh pengujian model — salah satu alasan untuk menjaga view sekecil mungkin. (b) `KeypadPanel` menghitung setiap label dengan `T9Dictionary.lettersOf(key)`, dari tabel `LETTERS` yang sama yang menghitung tanda tangan, sehingga label dan pencarian tidak mungkin berselisih; fakta itu hanya punya satu salinan. (c) Huruf *z*: label tombol 9 menampilkan *v*, *w*, *x*, dan *y*, dan *v* sudah ada di tombol 8, sehingga *z* tidak muncul di tombol mana pun, padahal mengetik 9 untuk *z* tetap berfungsi.

Penyelesaian 11.7. Perubahannya ada di `deleteDigit`, yang kini berarti “hapus angka terakhir, atau bila tidak ada, kata terakhir yang telah diterima”:

```
    void deleteDigit() {
        if (!signature.isEmpty()) {
            signature = signature.substring(0, signature.length() - 1);
            lookUp();
            fireChanged();
        } else if (!accepted.isEmpty()) {
            accepted.remove(accepted.size() - 1);
            fireChanged();
        }
    }
```

Controller tidak berubah, karena `C` tetap berarti `deleteDigit`; hanya makna operasinya yang meluas (Anda mungkin ingin menamainya ulang `deleteBack`). Di Python cabang barunya adalah `elif self._accepted: self._accepted.pop(); self._fire_changed()`. Diperlukan dua pemeriksaan baru: satu mengetik `8430` (*the*), menekan `C`, dan mengharapkan teks kosong; satu lagi menghitung peristiwa dan mengharapkan tepat satu untuk `C` itu dan tidak ada untuk `C` berikutnya pada pesan yang kosong.

Satu pengujian yang sudah ada harus berubah, dan alasannya patut disimak. *One event per real change* mengetik *hello*, menerimanya, lalu memanggil `deleteDigit` dengan harapan “tidak ada yang dihapus”. Dengan aturan baru, panggilan itu menghapus *hello*: hitungannya menjadi 7 dan teksnya kosong. Pengujian yang gagal itu sedang menjalankan tugasnya — ia melaporkan bahwa perilaku telah berubah — dan memutuskan bahwa perilaku baru memang dikehendaki, lalu memperbarui pengujian agar menyatakannya, adalah keputusan pemrogram.

Penyelesaian 11.8. **Kode 11.21** dan **Kode 11.22** menambahkan satu kelas, pendengar buku pesanan, dan satu baris yang mendaftarkannya.

Kode 11.21 `PickupBoard.java`: view baru atas buku pesanan untuk layar ruang tunggu, ditambahkan tanpa mengubah model. Kompilasi bersama `OrderBook.java`.

```
import java.util.ArrayList;
import java.util.List;

public class PickupBoard {
    public static void main(String[] args) {
        OrderBook book = new OrderBook();
        book.addListener(new PickupBoardView());  // satu-satunya baris baru: sebuah view
baru
        System.out.println("> three orders arrive");
        book.addOrder("Sari Wulandari", ServiceType.WASH_FOLD, 4.2, false);
        book.addOrder("Budi", ServiceType.WASH_IRON, 2.6, true);
        book.addOrder("Citra", ServiceType.IRON_ONLY, 1.3, false);
        System.out.println("> Budi's order is washed, then ready");
        book.advance(2);
        book.advance(2);
        System.out.println("> Citra's order is washed, then ready");
        book.advance(3);
        book.advance(3);
        System.out.println("> Budi picks up");
        book.advance(2);
    }
}

// Layar ruang tunggu: hanya menampilkan pesanan yang siap, dan menggambar ulang
// hanya bila daftar itu benar-benar berubah.
class PickupBoardView implements OrderBookListener {
    private String lastShown = "";

    @Override
    public void orderBookChanged(OrderBook book) {
        List<String> ready = new ArrayList<>();
        for (LaundryOrder order : book.getOrders()) {
            if (order.getStatus() == OrderStatus.READY) {
                ready.add("#" + order.getNumber() + " " + order.getCustomer());
            }
        }
        String board = ready.isEmpty() ? "(nothing yet)" : String.join("   ", ready);
        if (!board.equals(lastShown)) {
            System.out.println("  READY FOR PICKUP: " + board);
            lastShown = board;
        }
    }
}
```

```
> three orders arrive
  READY FOR PICKUP: (nothing yet)
> Budi's order is washed, then ready
  READY FOR PICKUP: #2 Budi
> Citra's order is washed, then ready
  READY FOR PICKUP: #2 Budi   #3 Citra
> Budi picks up
  READY FOR PICKUP: #3 Citra
```

View itu menghitung gambarnya dari keadaan model saat ini dan hanya menggambar ulang bila gambarnya berbeda dari yang terakhir: tiga pesanan baru menghasilkan tiga pemberitahuan, tetapi hanya satu penggambaran. Inilah televisi dari Subbab 11.2, dan modelnya tidak berubah.

Kode 11.22 `pickup_board.py`: view ruang tunggu dalam Python; objek yang memiliki `__call__` adalah *callable*, sehingga dapat didaftarkan sebagai pendengar.

```python
"""pickup_board.py: view baru dari buku pesanan, ditambahkan tanpa menyentuh model."""
from order_book import OrderBook, OrderStatus, ServiceType


class PickupBoardView:
    """Layar ruang tunggu: hanya menampilkan pesanan yang siap, dan menggambar
    ulang hanya bila daftar itu benar-benar berubah."""

    def __init__(self):
        self._last_shown = ""

    def __call__(self, book):
        ready = [f"#{o.number} {o.customer}" for o in book.orders()
                 if o.status is OrderStatus.READY]
        board = "   ".join(ready) if ready else "(nothing yet)"
        if board != self._last_shown:
            print("  READY FOR PICKUP: " + board)
            self._last_shown = board


def main():
    book = OrderBook()
    book.add_listener(PickupBoardView())  # satu-satunya baris baru: sebuah view baru
    print("> three orders arrive")
    book.add_order("Sari Wulandari", ServiceType.WASH_FOLD, 4.2, False)
    book.add_order("Budi", ServiceType.WASH_IRON, 2.6, True)
    book.add_order("Citra", ServiceType.IRON_ONLY, 1.3, False)
    print("> Budi's order is washed, then ready")
    book.advance(2)
    book.advance(2)
    print("> Citra's order is washed, then ready")
    book.advance(3)
    book.advance(3)
    print("> Budi picks up")
    book.advance(2)


if __name__ == "__main__":
    main()
```

```
> three orders arrive
  READY FOR PICKUP: (nothing yet)
> Budi's order is washed, then ready
  READY FOR PICKUP: #2 Budi
> Citra's order is washed, then ready
  READY FOR PICKUP: #2 Budi   #3 Citra
> Budi picks up
  READY FOR PICKUP: #3 Citra
```

Penyelesaian 11.9. *Model:* status baru atau, lebih sederhana, penghapusan dari list. Di Java:

```java
public void cancel(int number) {
    LaundryOrder order = find(number);
    if (order.getStatus() != OrderStatus.RECEIVED) {
        throw new IllegalStateException("order #" + number + " is already being washed");
    }
    orders.remove(order);
    fireChanged();
}
```

dan di Python:

```python
def cancel(self, number):
    order = self.find(number)
    if order.status is not OrderStatus.RECEIVED:
        raise RuntimeError(f"order #{number} is already being washed")
    self._orders.remove(order)
    self._fire_changed()
```

Interface view: `OrderListView` mendapat `onCancel(Runnable)` / `on_cancel(action)`. *Controller:* sebuah metode `cancelSelected()`, didaftarkan di konstruktor, dengan bentuk persis seperti `advanceSelected()`. *View Swing dan Tkinter:* satu tombol lagi di sebelah **Advance status**, yang *listener* atau *command*-nya menjalankan aksi yang didaftarkan. *View teks:* sebuah metode `clickCancel()`. *Pengujian:* batalkan pesanan yang berstatus *received* (jumlah dan pendapatan turun, satu peristiwa), tolak pembatalan pesanan yang sedang dicuci (tidak ada yang berubah, tidak ada peristiwa), dan perluas uji mandiri. View ringkasan sama sekali tidak perlu diubah: ia menghitung ulang angka-angkanya dari model.

Penyelesaian 11.10. **Kode 11.23** dan **Kode 11.24** menulis properti dan *view model*-nya. "View"-nya adalah dua lambda yang diikat pada properti keluaran; dalam program sungguhan, keduanya akan mengatur teks sebuah label dan status aktif sebuah tombol.

Kode 11.23 `MiniBinding.java`: `Property` generik yang dapat diamati dan *view model* formulir pesanan. Kompilasi bersama `OrderBook.java`.

```
import java.util.ArrayList;
import java.util.List;
import java.util.Locale;
import java.util.Objects;
import java.util.function.Consumer;

public class MiniBinding {
    public static void main(String[] args) {
        OrderFormViewModel vm = new OrderFormViewModel();
        // "View"-nya: dua widget yang diikat ke properti view model.
        vm.preview.bind(text -> System.out.println("  preview label: " + text));
        vm.canAdd.bind(ok -> System.out.println("  Add order enabled: " + ok));
        System.out.println("> type customer \"Sari\"");
        vm.customer.set("Sari");
        System.out.println("> type weight \"4.2\"");
        vm.weightText.set("4.2");
        System.out.println("> tick Express");
        vm.express.set(true);
        System.out.println("> type weight \"4,2x\"");
        vm.weightText.set("4,2x");
    }
}

// Nilai yang memberi tahu pengamatnya setiap kali ia benar-benar berubah.
class Property<T> {
    private final List<Consumer<T>> observers = new ArrayList<>();
    private T value;

    Property(T initial) {
        value = initial;
    }

    T get() {
        return value;
    }

    void set(T newValue) {
        if (!Objects.equals(value, newValue)) {
            value = newValue;
            for (Consumer<T> observer : List.copyOf(observers)) {
                observer.accept(value);
            }
        }
    }
```

Kode 11.23 (Lanjutan) *MiniBinding.java: Property generik yang dapat diamati dan view model formulir pesanan. Kompilasi bersama OrderBook.java.*

```
    void observe(Consumer<T> observer) {
        observers.add(observer);
    }

    // Seperti observe, tetapi juga langsung menampilkan nilai saat ini.
    void bind(Consumer<T> observer) {
        observe(observer);
        observer.accept(value);
    }
}

// View model formulir pesanan (hanya Wash & Fold, agar ringkas):
// properti masukan masuk, properti tampilan keluar, tanpa widget di mana pun.
class OrderFormViewModel {
    final Property<String> customer = new Property<>("");
    final Property<String> weightText = new Property<>("");
    final Property<Boolean> express = new Property<>(false);
    final Property<String> preview = new Property<>("enter a weight");
    final Property<Boolean> canAdd = new Property<>(false);

    OrderFormViewModel() {
        customer.observe(v -> recompute());
        weightText.observe(v -> recompute());
        express.observe(v -> recompute());
    }

    private void recompute() {
        double kg;
        try {
            kg = Double.parseDouble(weightText.get().strip().replace(',', '.'));
        } catch (NumberFormatException e) {
            kg = Double.NaN;
        }
        if (!(kg > 0 && kg <= OrderBook.MAX_KG)) {
            preview.set("enter a weight");
            canAdd.set(false);
            return;
        }
        LaundryOrder draft = new LaundryOrder(0, customer.get(), ServiceType.WASH_FOLD,
                kg, express.get());
        preview.set(String.format(Locale.US, "%.1f kg charged, %s",
                draft.chargeableKg(), OrderBook.rupiah(draft.price())));
        canAdd.set(!customer.get().isBlank());
    }
}
```

```
  preview label: enter a weight
  Add order enabled: false
> type customer "Sari"
> type weight "4.2"
  preview label: 4.5 kg charged, Rp31,500
  Add order enabled: true
> tick Express
  preview label: 4.5 kg charged, Rp47,250
> type weight "4,2x"
  preview label: enter a weight
  Add order enabled: false
```

`bind` langsung menampilkan nilai-nilai awal. Mengetik nama tidak mengubah properti keluaran apa pun, sehingga tidak ada yang digambar ulang. Berat 4.2 menghasilkan pratinjau dan mengaktifkan tombol; mencentang *Express* hanya mengubah pratinjau (Rp47,250 adalah $\lfloor 3 \times 31500/2 \rfloor$ menurut Persamaan (11.6)), sehingga hanya label pratinjau yang digambar ulang;

dan berat `4,2x` menonaktifkan tombol lagi. *View model* memakai ulang aturan harga milik model melalui `LaundryOrder` alih-alih mengulanginya — *view model* yang menghitung harga sendiri akan menjadi kekusutan Subbab 11.2 di tempat yang baru.

Kode 11.24 `mini_binding.py`: properti dan *view model* yang sama dalam Python.

```
"""mini_binding.py: MVVM dalam bentuk mini, dengan properti yang dapat diamati."""
from order_book import MAX_KG, LaundryOrder, ServiceType, rupiah


class Property:
    """Nilai yang memberi tahu pengamatnya setiap kali ia benar-benar berubah."""

    def __init__(self, initial):
        self._value = initial
        self._observers = []

    def get(self):
        return self._value

    def set(self, new_value):
        if new_value != self._value:
            self._value = new_value
            for observer in list(self._observers):
                observer(new_value)

    def observe(self, observer):
        self._observers.append(observer)

    def bind(self, observer):
        """Seperti observe, tetapi juga langsung menampilkan nilai saat ini."""
        self.observe(observer)
        observer(self._value)


class OrderFormViewModel:
    """View model formulir pesanan (hanya Wash & Fold, agar ringkas): properti
    masukan masuk, properti tampilan keluar, tanpa widget di mana pun."""

    def __init__(self):
        self.customer = Property("")
        self.weight_text = Property("")
        self.express = Property(False)
        self.preview = Property("enter a weight")
        self.can_add = Property(False)
        for source in (self.customer, self.weight_text, self.express):
            source.observe(lambda _value: self._recompute())
```

Kode 11.24 (Lanjutan) *mini_binding.py: properti dan view model yang sama dalam Python.*

```
    def _recompute(self):
        try:
            kg = float(self.weight_text.get().strip().replace(",", "."))
        except ValueError:
            kg = float("nan")
        if not (0 < kg <= MAX_KG):
            self.preview.set("enter a weight")
            self.can_add.set(False)
            return
        draft = LaundryOrder(0, self.customer.get(), ServiceType.WASH_FOLD,
                             kg, self.express.get())
        self.preview.set(f"{draft.chargeable_kg():.1f} kg charged,
{rupiah(draft.price())}")
        self.can_add.set(self.customer.get().strip() != "")


def main():
    vm = OrderFormViewModel()
    # "View"-nya: dua widget yang diikat ke properti view model.
    vm.preview.bind(lambda text: print("  preview label: " + text))
    vm.can_add.bind(lambda ok: print("  Add order enabled: " + str(ok).lower()))
    print('> type customer "Sari"')
    vm.customer.set("Sari")
    print('> type weight "4.2"')
    vm.weight_text.set("4.2")
    print("> tick Express")
    vm.express.set(True)
    print('> type weight "4,2x"')
    vm.weight_text.set("4,2x")


if __name__ == "__main__":
    main()
```

```
  preview label: enter a weight
  Add order enabled: false
> type customer "Sari"
> type weight "4.2"
  preview label: 4.5 kg charged, Rp31,500
  Add order enabled: true
> tick Express
  preview label: 4.5 kg charged, Rp47,250
> type weight "4,2x"
  preview label: enter a weight
  Add order enabled: false
```

Penyelesaian 11.11. Dengan pesanan 2 berstatus *Washing*: 1 pengguna → view tabel: klik *Advance status*; 2 view → controller: `advanceSelected()`; 3 controller → view daftar: `selectedOrderNumber()` mengembalikan 2; 4 controller → model: `advance(2)`; 5 model → model: cari, periksa, atur status *Ready*; 6 model → controller: `orderBookChanged(book)`; 7 controller → view daftar: `showOrders(orders)`; 8 controller → ringkasan: `showSummary(...)`; 9 model → controller: kembali; 10 controller → model: `find(2)` untuk membaca status baru; 11 controller → formulir: `showMessage("Order #2 is now Ready")`. Model mengirim satu pesan kepada objek lain (6), ditambah nilai kembalinya. Bila pesanan 2 sudah diambil, pesan 5 berakhir dengan melempar `IllegalStateException`; pesan 6 sampai 10 tidak terjadi; eksepsi itu merambat kembali ke controller, yang menangkapnya dan mengirim `showMessage("Not changed: order #2 was already picked up")`. Dalam kasus ini model *tidak* mengirim pesan apa pun — ia hanya melempar eksepsi — dan itulah persisnya makna "operasi yang ditolak tidak mengubah apa pun dan tidak memberi tahu siapa pun".

Penyelesaian 11.12. *Tanpa perubahan:* `OrderBook`, `LaundryOrder`, `ServiceType`, `OrderStatus`, dan `OrderBookListener` — seluruh model — beserta *unit test*-nya. *Diganti:* view-view (halaman HTML atau antarmuka JavaScript sebagai ganti Swing atau Tkinter). *Diubah:* controller, yang menerima permintaan HTTP alih-alih klik tetapi tetap mengubah teks menjadi nilai dan memanggil metode model yang sama; bila lapisan web mengimplementasikan kontrak-kontrak view, `DeskController` bahkan dapat tetap seperti semula. Keputusan yang membuat model dapat dipakai ulang adalah aturan **Tabel 11.1**: model tidak mengimpor apa pun dari *toolkit* antarmuka pengguna mana pun.

Daftar Pustaka

[1] R. Eckstein, "Java SE application design with MVC," Oracle Technical Article, 2007. [Online]. Available: https://www.oracle.com/technical-resources/articles/javase/application-design-with-mvc.html

[2] T. Reenskaug, "Models–Views–Controllers," Technical note, Xerox PARC, Palo Alto, CA, USA, Dec. 1979. [Online]. Available: https://folk.universitetetioslo.no/trygver/themes/mvc/mvc-index.html

[3] G. E. Krasner and S. T. Pope, "A cookbook for using the model-view controller user interface paradigm in Smalltalk-80," *Journal of Object-Oriented Programming*, vol. 1, no. 3, pp. 26–49, 1988.

[4] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Reading, MA, USA: Addison-Wesley, 1994.

[5] A. Fowler, "A Swing architecture overview," Sun Microsystems / Oracle Technical Article. [Online]. Available: https://www.oracle.com/java/technologies/a-swing-architecture.html

[6] TutorialsPoint, "Design patterns – MVC pattern." [Online]. Available: https://www.tutorialspoint.com/design_pattern/mvc_pattern.htm

[7] Oracle, *Java Platform, Standard Edition 21 API Specification*: `java.util.Observable` (deprecated since 9), `java.beans.PropertyChangeSupport`, `java.beans.PropertyChangeListener`. [Online]. Available: https://docs.oracle.com/en/java/javase/21/docs/api/

[8] Oracle, "Concurrency in Swing," *The Java Tutorials*. [Online]. Available: https://docs.oracle.com/javase/tutorial/uiswing/concurrency/

[9] M. Feathers, "The humble dialog box," Object Mentor, 2002. [Online]. Available: https://martinfowler.com/articles/humble-dialog-box.pdf

[10] Python Software Foundation, *The Python Standard Library*, version 3.14: `tkinter`, `tkinter.ttk`, `typing.Protocol`, `unittest`. [Online]. Available: https://docs.python.org/3/library/

[11] D. L. Grover, M. T. King, and C. A. Kushler, "Reduced keyboard disambiguating computer," U.S. Patent 5 818 437, Oct. 6, 1998.

[12] The JUnit Team, *JUnit User Guide*, version 6.1.3 (Aug. 2026), and "Release notes: 6.0.0" (Java 17 baseline). [Online]. Available: https://docs.junit.org/current/

[13] M. Potel, "MVP: Model-View-Presenter, the Taligent programming model for C++ and Java," Taligent, Inc., Cupertino, CA, USA, 1996.

[14] J. Gossman, "Introduction to Model/View/ViewModel pattern for building WPF apps," Microsoft developer blog, Oct. 2005.

[15] M. Fowler, "GUI architectures," 2006. [Online]. Available: https://martinfowler.com/eaaDev/uiArchs.html

Bab 12

Evaluasi dan Pembahasan III: Teks Prediktif dengan List, Map, Pohon, dan Papan Tombol

Bab ini tidak menambahkan fitur bahasa yang baru. Bab ini mengerjakan enam belas soal, masing-masing diselesaikan dari halaman kosong dalam Java dan dalam Python. Sepuluh soal pertama membentuk satu proyek: *teks prediktif* pada telepon genggam lama, tempat Anda menekan setiap tombol satu kali dan telepon menebak katanya. Kamusnya kita bangun empat kali: membaca berkas pada setiap pertanyaan, menyimpan *list* terurut, menyimpan *map*, dan menyimpan pohon yang menemukan kata ketika kata itu masih sedang diketik. Keempatnya kita ukur waktunya, lalu kita pasang papan tombol di layar dengan pola Model–View–Controller dari Bab 11. Enam soal lainnya baru, satu untuk setiap gagasan Bab 9 sampai Bab 11 yang layak ditengok sekali lagi: memilih koleksi dengan mengukurnya, kelas generik berbatas, rantai *comparator*, penghitung yang modelnya diuji tanpa jendela, dua *bug* observer, dan tombol-tombol yang dibuat dalam gelung. Setiap soal menempuh jalan yang sama seperti di Bab 4 dan Bab 8: nyatakan ulang, rancang, selesaikan dalam kedua bahasa, jalankan ujinya, jelaskan kodenya baris demi baris, pelajari penyelesaian keliru yang menggoda, lalu uji diri dengan sebuah varian. Di akhir bab, Anda akan memilih struktur data menurut operasi yang harus didukungnya, membuktikan pilihan itu dengan pengukuran, dan menjaga antarmuka pengguna cukup tipis sehingga hampir semuanya dapat diuji tanpa antarmuka itu.

12.1 Titik Tolak Kita

Bab 9, 10, dan 11 memberi kita tiga perkakas, dan bab ini memakai ketiganya bersama-sama.

Dari **Bab 9** kita memiliki *koleksi* dan *generik*. Koleksi adalah objek yang menampung objek lain: *list* (barisan berurutan yang boleh berisi duplikat: `ArrayList` di Java, `list` di Python), *set* atau himpunan (tanpa duplikat: `HashSet` dan `TreeSet`, `set` di Python), dan *map* (dari kunci ke nilai: `HashMap` dan `TreeMap`, `dict` di Python). Generik memungkinkan satu kelas melayani banyak tipe elemen sementara kompilator tetap memeriksanya: `ArrayList<String>` hanya menampung string. Kita juga sudah mengenal `Comparable` dan `Comparator`, dua cara Java mengurutkan objek, serta argumen `key=` yang melakukan hal yang sama di Python.

Dari **Bab 10** kita memiliki *antarmuka pengguna grafis* dan *peristiwa* (*event*). Program Swing atau Tkinter membangun jendela dari komponen-komponen lalu menunggu; ketika pengguna mengeklik, *toolkit* memanggil *listener* (Java) atau *callback* (Python) yang dipasang pada tombol, yang sering ditulis sebagai lambda atau kelas dalam.

Dari **Bab 11** kita memiliki *Model–View–Controller*. *Model* memegang data dan aturan serta tidak tahu apa pun tentang jendela, *view* menampilkan model, dan *controller* mengubah tindakan pengguna menjadi pemanggilan pada model. Ketika model berubah, ia memberi tahu setiap view yang terdaftar: pola *observer* [1].

Bab ini tersusun dalam empat lapis: sebuah metode (Subbab 12.2); proyek teks prediktif (Subbab 12.3 sampai 12.7, Soal 12.1 sampai 12.10); enam soal baru berlatar toko Profitina Laundry yang menyertai buku ini (Subbab 12.8 dan 12.9, Soal 12.11 sampai 12.16); dan subbab penutup seperti biasa. Setiap soal memakai label tebal yang sama agar Anda mudah menemukan jalan kembali: **Soal**, **Dengan bahasa kita sendiri**, **Rancangan**, **Penyelesaian Java**, **Penyelesaian Python**, **Baris demi baris**, **Penyelesaian keliru yang menggoda**, dan **Uji diri**. Seperti di Bab 8, setiap program Java menguji dirinya sendiri dengan metode bantu kecil `check` yang mencetak `PASS` beserta sebuah nama, dan setiap program Python memakai `unittest` [2]. Program yang mengukur waktu mencetak angka yang berbeda di setiap komputer; angka di sini adalah angka sungguhan dari mesin tempat buku ini dibangun, dan yang penting adalah *bentuknya*: kolom mana yang tumbuh, dan seberapa cepat.

> **CARA MEMAKAI BAB INI.** Bacalah sebuah soal, tutup sisa halamannya, lalu cobalah sendiri selama sepuluh menit. Untuk soal-soal kamus, tuliskan lebih dulu *operasi apa saja* yang harus didukung strukturnya dan *seberapa sering*; untuk soal-soal GUI, *apa yang harus diingat model*. Sesudah itu, lanjutkan membaca dan bandingkan.

12.2 Cara Menghadapi Soal Koleksi dan Antarmuka

Soal tentang koleksi tampak seperti soal tentang kode, padahal sebenarnya soal tentang *operasi*. "Simpanlah kamus" bukan spesifikasi; "diberikan sebuah signature, temukan kata-katanya seribu kali per detik, sementara kata baru sesekali datang" barulah spesifikasi. Begitu operasi dan frekuensinya tertulis di kertas, strukturnya hampir memilih dirinya sendiri, dan antarmuka pengguna menjadi lapisan tipis yang memanggil operasi-operasi itu. **Tabel 12.1** meringkas metodenya dalam enam langkah.

> **DEFINISI.** *Biaya* sebuah operasi adalah banyaknya langkah dasar yang diperlukannya, sebagai fungsi dari ukuran data n. $O(f(n))$ ("orde $f(n)$") menyatakan biaya yang tumbuh paling cepat seperti $f(n)$. $O(1)$ tidak bergantung pada n; $O(\log n)$ bertambah satu langkah ketika n berlipat dua; $O(n)$ berlipat dua ketika n berlipat dua.

Langkah 1 — Daftarkan operasinya dan seberapa sering masing-masing terjadi. *Menambah kata*: sesekali. *Mencari kata-kata sebuah signature*: pada setiap tekanan tombol. *Mencetak semuanya secara urut*: sekali sehari. Operasi yang sering menentukan rancangan; operasi yang jarang cukup dimungkinkan saja.

Langkah 2 — Cocokkan setiap operasi yang sering dengan sebuah struktur. "Apakah ini ada?" dan "apa yang menjadi milik kunci ini?" adalah tugas *hash set* atau *hash map*, rata-rata $O(1)$. "Semuanya secara urut" atau "semua di antara dua nilai" adalah tugas struktur terurut (*list* terurut dengan pencarian biner, `TreeSet`, `TreeMap`). "Semua yang diawali prefiks ini" adalah tugas pohon yang lintasannya mengeja kuncinya. *List* biasa hanya tepat bila Anda kebanyakan menambah di ujung dan menelusuri semuanya.

Langkah 3 — Sembunyikan pilihan itu di balik antarmuka. Kode yang *memakai* struktur ditulis sekali terhadap antarmukanya, dan mengganti struktur hanya mengubah satu baris.

Langkah 4 — Taruh keadaan di dalam model yang tidak tahu apa pun tentang jendela. Tentukan apa yang harus *diingat* program di antara dua klik, lalu taruh tepat itu di dalam kelas dengan metode-metode biasa.

Langkah 5 — Uji model tanpa jendela. Metode biasa mendapat uji biasa; sakelar `--selftest` memungkinkan program GUI mengeklik tombolnya sendiri di mesin yang tidak memiliki layar.

Langkah 6 — Ukur, dengan data yang sama untuk setiap kandidat. Rumus biaya meramalkan *bentuk* waktu eksekusi; stopwatch memastikannya.

Tabel 12.1 Enam langkah untuk soal yang memadukan koleksi, antarmuka, dan antarmuka pengguna. Langkah 1 sampai 3 memilih struktur data, Langkah 4 dan 5 membentuk GUI, dan Langkah 6 memeriksa pilihannya.

Langkah	Tulis di kertas	Pertanyaan yang dijawabnya
1	Operasi dan frekuensinya	Apa yang harus dikerjakan data, dan seberapa sering?
2	Sebuah struktur untuk setiap operasi yang sering	Koleksi mana yang mengerjakannya dengan murah?
3	Antarmuka berisi operasi-operasi itu	Bagaimana pilihan itu tetap dapat diganti?
4	*Field* dan metode model	Apa yang harus diingat program di antara peristiwa?
5	Uji untuk model; `--selftest` untuk GUI	Bagaimana menguji tanpa mengeklik?
6	Tabel waktu dengan data yang sama untuk semua	Apakah ramalannya terbukti?

Dua peringatan menyertai Langkah 6. Mesin virtual Java mengompilasi kode yang sering dipakai menjadi kode mesin selagi program berjalan (*just-in-time compilation*), sehingga eksekusi awal lebih lambat daripada eksekusi berikutnya; program pengukur waktu kita mengulangi setiap pengukuran dan menyimpan yang terbaik. Selain itu, pengukuran yang sangat kecil sebagian besar hanyalah derau: buatlah datanya besar dan pertanyaannya banyak.

12.3 Masalah Teks Prediktif

12.3.1 Tombol, huruf, dan signature

Sebelum telepon memiliki papan ketik lengkap, telepon memiliki dua belas tombol, delapan di antaranya memuat huruf, dengan susunan seperti pada **Gambar 12.1**: tombol 2 memuat a, b, dan c; tombol 7 memuat p, q, r, dan s; tombol 9 memuat w, x, y, dan z; tombol 0 adalah spasi; tombol 1, `*`, dan `#` tidak memuat huruf.



Gambar 12.1 Papan tombol telepon. Delapan tombol memuat huruf; tombol 7 dan 9 (diarsir) masing-masing memuat empat huruf, enam tombol lainnya tiga huruf; tombol 0 adalah spasi.

Pada sistem lama, sebuah huruf diketik dengan menekan tombolnya satu sampai empat kali: “hello” memerlukan tiga belas tekanan, 4, 4, 3, 3, 5, 5, 5, jeda, 5, 5, 5, 6, 6, 6. *Teks prediktif*, yang

dipasarkan dengan nama T9 [3], hanya meminta setiap tombol ditekan sekali: Anda mengetik 4, 3, 5, 5, 6, lalu telepon mencari kata-kata yang mungkin dimaksud oleh kelima tekanan itu. Deretan angka itu disebut *signature* (tanda tombol) kata tersebut. Jika $\kappa(c)$ adalah tombol yang memuat huruf c, signature kata $w = c_1 c_2 \ldots c_m$ adalah

$$\text{sig}(c_1 c_2 \ldots c_m) = \kappa(c_1)\,\kappa(c_2)\ \ldots\ \kappa(c_m) \tag{12.1}$$

yaitu untai m angka, bukan hasil kali. Menurut Persamaan (12.1), sig(hello) = 43556 dan sig(home) = 4663. Fungsi ini tidak dapat dibalik secara unik, karena beberapa kata dapat berbagi signature: sig(home) = sig(good) = sig(gone) = 4663. Itulah sebabnya telepon memerlukan kamus dan menawarkan kandidat. Dalam program kita, `*` berpindah ke kandidat berikutnya, `0` menerima kata dan mengetik spasi, dan `#` menghapus huruf terakhir.

12.3.2 Berkas kamus

Kamus di sini hanyalah berkas teks berisi satu kata per baris. Untuk contoh-contoh kita memakai `words.txt`, 99 baris, di folder yang sama dengan programnya. Awal dan akhirnya tampak seperti ini:

```
a
act
bad
bat
...
worm
don't
e-mail
U.S.
3rd

café
  soap
London
```

Berkas ini sengaja sedikit kotor, seperti daftar kata yang sesungguhnya: ada apostrof, tanda hubung, singkatan, angka, baris kosong, huruf beraksen, spasi di sekitar kata, duplikat berhuruf kapital (`Hello`, `Home`), dan nama diri. Setiap program mengikuti satu aturan: pangkas baris itu, ubah menjadi huruf kecil, dan pakailah hanya bila *sah*, yaitu tidak kosong dan hanya terdiri atas huruf a sampai z. Jadi `Home` menjadi `home` (dan tidak boleh muncul dua kali dalam jawaban mana pun), `soap` menjadi `soap`, `London` menjadi `london`, dan enam baris aneh lainnya dilewati. Tersisa 93 baris.

Untuk percobaan pengukuran waktu kita memerlukan kamus yang jauh lebih besar, jadi kita membangkitkannya: `MakeDictionary` dan `make_dictionary.py` menulis 200.000 kata rekaan yang dapat dilafalkan (`ululofasu`, `gaweju`, ...) ke `big_words.txt`, sama persis dalam kedua bahasa (Subbab 12.5.3).

12.3.3 Paket dan cara menjalankan program

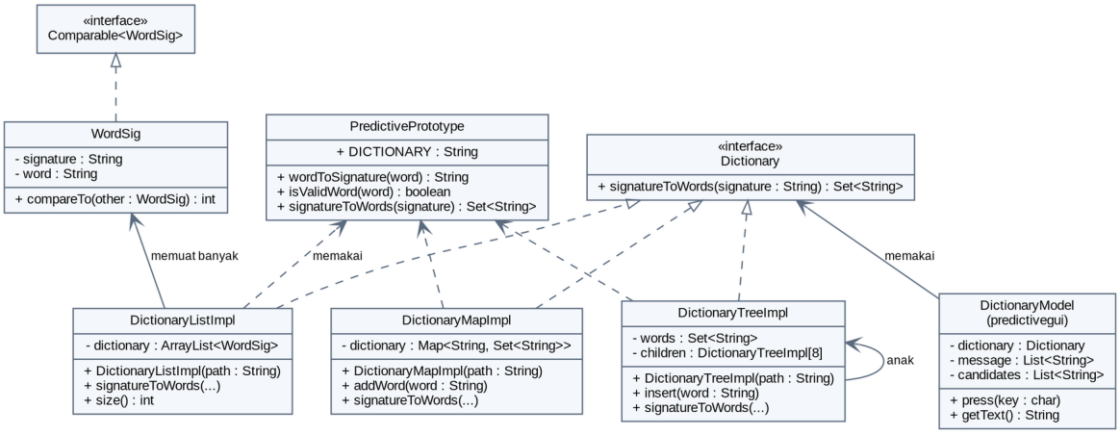
Kelas-kelas teks prediktif tinggal dalam *package* (paket) Java bernama `predictive`, kelas-kelas GUI dalam `predictivegui`. Paket adalah kelompok kelas yang bernama; namanya mengawali setiap berkas (`package predictive;`) dan harus sama dengan folder berkas itu. Dari folder `java` bab ini, yang berisi `words.txt` dan kedua folder paket:

```
javac predictive/*.java predictivegui/*.java *.java
java predictive.Sigs2WordsList 4663 43556
java predictivegui.PredictiveGUI
```

Kompilator menerima *path* berkas dengan garis miring; `java` menerima *nama* kelas dengan titik, diketik dari folder *di atas* `predictive`. Folder `predictive` dan `predictivegui` di Python juga merupakan paket (masing-masing berisi `__init__.py`), dan modul di dalamnya dijalankan dengan `python -m predictive.sigs2words_list 4663 43556` dari folder di atasnya. Setiap program membaca kamus yang dinamai oleh variabel lingkungan `PREDICTIVE_DICT`, atau `words.txt` bila variabel itu tidak diisi: di Linux dan macOS, `PREDICTIVE_DICT=big_words.txt java ...` mengisinya untuk satu perintah; di Windows PowerShell, `$env:PREDICTIVE_DICT="big_words.txt"`. Berkas `README.md` di folder itu memuat setiap perintah untuk Windows, macOS, dan Linux, di terminal maupun di VS Code.

12.3.4 Seluruh rancangan sekilas

Gambar 12.2 memperlihatkan kelas-kelas yang akan kita tulis. Antarmuka `Dictionary` memiliki satu metode, `signatureToWords`. Tiga kelas mengimplementasikannya dengan koleksi yang berbeda di dalamnya: `DictionaryListImpl` (`ArrayList` terurut berisi pasangan `WordSig`), `DictionaryMapImpl` (`HashMap` dari signature ke himpunan kata), dan `DictionaryTreeImpl` (pohon dengan paling banyak delapan anak per simpul). Semuanya memakai metode bantu statis `PredictivePrototype`, dan `DictionaryModel` milik GUI memakai `Dictionary` *apa pun*.



Gambar 12.2 Diagram kelas UML proyek teks prediktif. Garis putus-putus dengan segitiga kosong berarti "mengimplementasikan"; panah padat berarti "memiliki" atau "memakai"; `DictionaryTreeImpl` merujuk objek dari kelasnya sendiri.

Tabel 12.2 memetakan sepuluh soal proyek ini ke empat bagian.

Tabel 12.2 Empat bagian proyek teks prediktif dan bab-bab yang gagasannya dipakai setiap bagian.

Bagian	Soal	Yang dibangun	Bab yang dilatih
Satu: prototipe	12.1–12.3	signature, pencarian yang memindai berkas, dua program baris perintah	9 (himpunan), 5 (berkas)
Dua: *list* terurut	12.4–12.6	`WordSig`, pencarian biner, antarmuka `Dictionary`, pengukuran pertama	9 (*list*, `Comparable`), 6 (antarmuka)
Tiga: *map* dan pohon	12.7–12.9	*map* bernilai jamak, pohon prefiks, pengukuran keempatnya	9 (*map*), 8 (rekursi)
Empat: papan tombol	12.10	model, view, dan controller dalam Swing dan Tkinter	10, 11

12.4 Bagian Satu: Prototipe

Prototipe adalah versi pertama yang boleh lambat, asalkan sederhana dan jelas benar. Prototipe menetapkan jawaban yang benar, dan kelak menjadi tolok ukur untuk memeriksa dan mengukur versi-versi yang lebih cepat.

12.4.1 Soal 12.1 — Dari kata ke signature-nya

> **SOAL.** In a class `PredictivePrototype` of the package `predictive`, write the method `public static String wordToSignature(String word)`. It takes a word and returns its numeric signature; for example, `"home"` gives `"4663"`. If the word contains a character that is not a letter, that character becomes a space in the signature. Build the result character by character in a `StringBuffer` rather than a `String`, and explain in a comment why this is more efficient.
>
> *Dengan kata lain:* dalam kelas `PredictivePrototype` pada paket `predictive`, tulislah metode `public static String wordToSignature(String word)` yang mengembalikan signature angka sebuah kata, misalnya `"4663"` untuk `"home"`. Karakter yang bukan huruf menjadi spasi. Bangunlah hasilnya karakter demi karakter dalam `StringBuffer`, bukan `String`, dan jelaskan dalam komentar mengapa cara itu lebih efisien.

Dengan bahasa kita sendiri. Telusuri kata dari kiri ke kanan; untuk setiap karakter, tanyakan tombol mana yang memuatnya, lalu tambahkan angka tombol itu di ujung jawaban. Karakter yang tidak dimuat tombol mana pun (apostrof, tanda hubung, angka, huruf beraksen) menyumbang sebuah spasi, sehingga signature tetap memiliki satu posisi per karakter. Huruf kapital tetap huruf, jadi `"Home"` juga menghasilkan `"4663"`.

Rancangan. Inti metode ini adalah pemetaan κ pada Persamaan (12.1) dari huruf ke tombolnya. *Switch expression* Java menuliskannya sebagai rantai kasus; *dictionary* Python menuliskannya sebagai tabel yang dibangun sekali. **Tabel 12.3** menelusuri gelungnya dengan tangan.

Tabel 12.3 Gelung `wordToSignature("Home")`, satu karakter per langkah. *Buffer* bertambah satu angka pada setiap langkah dan diubah menjadi `String` sekali saja, di akhir.

Langkah i	Karakter	Huruf kecil	Tombol κ	Isi *buffer* setelah langkah
0	`H`	`h`	`4`	`4`
1	`o`	`o`	`6`	`46`
2	`m`	`m`	`6`	`466`
3	`e`	`e`	`3`	`4663`

Mengapa `StringBuffer`? `String` di Java bersifat *imutabel*: `s = s + '6'` tidak memperpanjang `s`, tetapi membuat string baru, menyalin semua karakter lama ke dalamnya, lalu menambahkan satu. Untuk kata sepanjang n huruf, gelung itu menyalin

$$1 + 2 + \cdots + n = \sum_{k=1}^{n} k = \frac{n(n+1)}{2} \tag{12.2}$$

karakter, kira-kira $n^2/2$: pekerjaannya tumbuh sebanding dengan *kuadrat* panjangnya. `StringBuffer` menyimpan larik dengan ruang cadangan dan menambahkan di tempat, menggandakan lariknya pada saat-saat jarang ketika penuh, sehingga total pekerjaannya tetap sebanding dengan n. (`StringBuilder` adalah kelas yang sama tanpa penguncian yang membuat `StringBuffer` aman bagi beberapa *thread*, pokok bahasan Bab 14; untuk variabel lokal, keduanya sama baiknya.)

Penyelesaian Java. **Kode 12.1** adalah kelas lengkapnya: metode soal ini, metode bantu `isValidWord`, dan pencarian `signatureToWords` dari Soal 12.2.

Kode 12.1 `PredictivePrototype.java`: signature sebuah kata, uji keabsahan, dan pencarian yang memindai berkas, semuanya sebagai metode statis dari satu kelas utilitas.

```
// PredictivePrototype.java - Soal 12.1 dan 12.2: signature papan tombol sebuah kata,
// dan pencarian yang sengaja sederhana, yang membaca berkas kamus setiap kali.
package predictive;

import java.io.File;
import java.io.IOException;
import java.nio.charset.StandardCharsets;
import java.util.HashSet;
import java.util.Scanner;
import java.util.Set;

public class PredictivePrototype {

    /** Berkas kamus: variabel lingkungan PREDICTIVE_DICT, atau words.txt. */
    public static final String DICTIONARY =
            System.getenv().getOrDefault("PREDICTIVE_DICT", "words.txt");

    private PredictivePrototype() {             // kelas utilitas: tanpa objek
    }

    /**
     * Mengembalikan signature papan tombol sebuah kata, misalnya "home" menghasilkan
"4663".
     * Karakter yang bukan huruf a sampai z menjadi spasi.
     * Hasilnya dikumpulkan dalam StringBuffer, yang tumbuh di tempat. Menulis
     * result = result + digit pada String akan menyalin setiap karakter yang sudah
     * terkumpul pada setiap langkah: sekitar n*n/2 salinan untuk kata n huruf, bukan n.
     */
    public static String wordToSignature(String word) {
        StringBuffer result = new StringBuffer(word.length());
        for (int i = 0; i < word.length(); i++) {
            result.append(keyOf(Character.toLowerCase(word.charAt(i))));
        }
        return result.toString();
    }

    /** Tombol yang memuat sebuah huruf: a, b, c di '2', ..., w, x, y, z di '9'. */
    static char keyOf(char letter) {
        return switch (letter) {
            case 'a', 'b', 'c' -> '2';
            case 'd', 'e', 'f' -> '3';
            case 'g', 'h', 'i' -> '4';
            case 'j', 'k', 'l' -> '5';
            case 'm', 'n', 'o' -> '6';
            case 'p', 'q', 'r', 's' -> '7';
            case 't', 'u', 'v' -> '8';
            case 'w', 'x', 'y', 'z' -> '9';
```

Kode 12.1 (Lanjutan) *PredictivePrototype.java: signature sebuah kata, uji keabsahan, dan pencarian yang memindai berkas, semuanya sebagai metode statis dari satu kelas utilitas.*

```
            default -> ' ';
        };
    }

    /** True bila kata tidak kosong dan hanya berisi huruf a sampai z. */
    public static boolean isValidWord(String word) {
        if (word.isEmpty()) {
            return false;
        }
        for (int i = 0; i < word.length(); i++) {
            char c = word.charAt(i);
            if (c < 'a' || c > 'z') {
                return false;
            }
        }
        return true;
    }

    /**
     * Mengembalikan setiap kata kamus yang signature-nya sama dengan yang diberikan, huruf
kecil
     * dan tanpa duplikat (Set tidak dapat memuat kata yang sama dua kali).
     * Sengaja tidak efisien: setiap panggilan membuka berkas dan membaca SELURUHNYA, dan
     * menghitung ulang signature setiap kata, sehingga satu pencarian memakan waktu
     * sebanding dengan ukuran seluruh kamus, dan seribu pencarian
     * membaca berkas yang sama seribu kali.
     */
    public static Set<String> signatureToWords(String signature) {
        Set<String> result = new HashSet<>();
        try (Scanner in = new Scanner(new File(DICTIONARY), StandardCharsets.UTF_8)) {
            while (in.hasNextLine()) {
                String word = in.nextLine().trim().toLowerCase();
                if (isValidWord(word) && wordToSignature(word).equals(signature)) {
                    result.add(word);
                }
            }
        } catch (IOException e) {
            throw new IllegalStateException("cannot open the dictionary " + DICTIONARY, e);
        }
        return result;
    }
}
```

Kelas yang hanya berisi metode statis adalah *kelas utilitas*, seperti `Math`, dan konstruktor `private`-nya mencegah pembuatan objek yang tak berguna. `DICTIONARY` membaca variabel lingkungan `PREDICTIVE_DICT` dari *map* `System.getenv()`, dengan `"words.txt"` sebagai nilai bawaan. **Kode 12.2** menguji kelas itu, menjalankan dua penyelesaian keliru yang menggoda, dan mengukur waktu membangun signature *n* angka dengan `String` dan dengan `StringBuffer`.

Kode 12.2 `PrototypeCheck.java`: pemeriksaan Soal 12.1 dan 12.2, dua varian keliru, dan pengukuran waktu `String` lawan `StringBuffer`.

```java
// PrototypeCheck.java - pemeriksaan Soal 12.1 dan 12.2, dengan dua penyelesaian
// keliru yang menggoda dan pengukuran waktu String lawan StringBuffer.
package predictive;

import java.util.ArrayList;
import java.util.List;
import java.util.Set;
import java.util.TreeSet;

public class PrototypeCheck {

    /** Menggoda tetapi salah: aritmetika "tiga huruf per tombol". */
    static String wordToSignatureWrong(String word) {
        StringBuffer result = new StringBuffer();
        for (char c : word.toCharArray()) {
            result.append((char) ('2' + (c - 'a') / 3));
        }
        return result.toString();
    }

    /** Menggoda tetapi salah: sebuah List, tanpa huruf kecil, tanpa pemeriksaan keabsahan.
*/
    static List<String> signatureToWordsWrong(String signature) throws Exception {
        List<String> result = new ArrayList<>();
        for (String line :
java.nio.file.Files.readAllLines(java.nio.file.Path.of("words.txt"))) {
            if (PredictivePrototype.wordToSignature(line).equals(signature)) {
                result.add(line);
            }
        }
        return result;
    }

    /** Membangun signature n angka dengan penyambungan String: n String baru. */
    static String concatWithString(int n) {
        String s = "";
        for (int i = 0; i < n; i++) {
            s = s + '2';
        }
        return s;
    }
```

Kode 12.2 (Lanjutan) *PrototypeCheck.java: pemeriksaan Soal 12.1 dan 12.2, dua varian keliru, dan pengukuran waktu String lawan StringBuffer.*

```
    static String concatWithBuffer(int n) {
        StringBuffer sb = new StringBuffer();
        for (int i = 0; i < n; i++) {
            sb.append('2');
        }
        return sb.toString();
    }

    private static int passed = 0;

    private static void check(String name, Object expected, Object actual) {
        if (!java.util.Objects.equals(expected, actual)) {
            throw new AssertionError(name + ": expected " + expected + " but got " +
actual);
        }
        passed++;
        System.out.println("PASS " + name);
    }

    public static void main(String[] args) throws Exception {
        check("home", "4663", PredictivePrototype.wordToSignature("home"));
        check("hello", "43556", PredictivePrototype.wordToSignature("hello"));
        check("four-letter keys", "7777 9999", PredictivePrototype.wordToSignature("pqrs
wxyz"));
        check("upper case", "4663", PredictivePrototype.wordToSignature("Home"));
        check("apostrophe", "366 8", PredictivePrototype.wordToSignature("don't"));
        check("empty word", "", PredictivePrototype.wordToSignature(""));
        check("valid", true, PredictivePrototype.isValidWord("laundry"));
        check("invalid e-mail", false, PredictivePrototype.isValidWord("e-mail"));
        check("invalid empty", false, PredictivePrototype.isValidWord(""));

        Set<String> w = new TreeSet<>(PredictivePrototype.signatureToWords("4663"));
        check("4663", "[gone, good, goof, home, hone, hood, hoof]", w.toString());
        check("5477", "[kiss, lips]", new
TreeSet<>(PredictivePrototype.signatureToWords("5477")).toString());
        check("no match", "[]", PredictivePrototype.signatureToWords("22222").toString());

        System.out.println("wrong 'you'      : " + wordToSignatureWrong("you")
                + "   right: " + PredictivePrototype.wordToSignature("you"));
        System.out.println("wrong 4663 list  : " + signatureToWordsWrong("4663"));
```

Kode 12.2 (Lanjutan) *PrototypeCheck.java: pemeriksaan Soal 12.1 dan 12.2, dua varian keliru, dan pengukuran waktu String lawan StringBuffer.*

```
        for (int n : new int[] {20_000, 40_000, 80_000}) {
            long t0 = System.nanoTime();
            concatWithString(n);
            long t1 = System.nanoTime();
            concatWithBuffer(n);
            long t2 = System.nanoTime();
            System.out.printf("n = %6d   String: %7.2f ms   StringBuffer: %5.2f ms%n",
                    n, (t1 - t0) / 1e6, (t2 - t1) / 1e6);
        }
        System.out.println("All " + passed + " checks passed.");
    }
}
```

Jalankan dari folder `java` dengan `java predictive.PrototypeCheck`:

```
PASS home
PASS hello
PASS four-letter keys
PASS upper case
PASS apostrophe
PASS empty word
PASS valid
PASS invalid e-mail
PASS invalid empty
PASS 4663
PASS 5477
PASS no match
wrong 'you'      : :68   right: 968
wrong 4663 list  : [gone, good, goof, home, Home, hone, hood, hoof]
n =  20000   String:  120.26 ms   StringBuffer:  2.37 ms
n =  40000   String:  184.31 ms   StringBuffer:  1.71 ms
n =  80000   String:  837.10 ms   StringBuffer: 13.24 ms
All 12 checks passed.
```

Pada ketiga ukuran itu, n tumbuh empat kali lipat dan waktu `String` kira-kira tujuh kali lipat, sesuai dengan hukum kuadrat Persamaan (12.2) yang dikaburkan oleh mesin virtual yang sedang "memanas"; kolom `StringBuffer` tetap di kisaran beberapa milidetik (satu nilainya yang lebih besar adalah derau dari mesin yang sibuk). Angka di komputer Anda akan berbeda; bentuknya tidak.

Penyelesaian Python. `str` di Python juga imutabel dan tidak ada `StringBuffer`; idiomnya adalah mengumpulkan potongan-potongan dalam sebuah *list*, yang tumbuh di tempat, lalu menggabungkannya sekali dengan `"".join(...)`. **Kode 12.3** melakukan hal itu, dengan *dictionary* yang dibangun sekali sebagai tabel κ, dan **Kode 12.4** mengujinya.

Kode 12.3 `prototype.py`: ketiga fungsi yang sama dalam Python. Tabel `_KEY_OF` dibangun sekali, saat modul pertama kali diimpor.

```python
# prototype.py - Soal 12.1 dan 12.2: signature papan tombol sebuah kata,
# dan pencarian yang sengaja sederhana, yang membaca berkas kamus setiap kali.
import os

DICTIONARY = os.environ.get("PREDICTIVE_DICT", "words.txt")

_KEY_OF = {}                                    # huruf -> tombol, dibangun sekali
for _digit, _letters in (("2", "abc"), ("3", "def"), ("4", "ghi"), ("5", "jkl"),
                         ("6", "mno"), ("7", "pqrs"), ("8", "tuv"), ("9", "wxyz")):
    for _letter in _letters:
        _KEY_OF[_letter] = _digit


def word_to_signature(word):
    """Kembalikan signature papan tombol sebuah kata: "home" menghasilkan "4663".

    Karakter yang bukan huruf a sampai z menjadi spasi. Angka-angkanya
    dikumpulkan dalam list dan digabung sekali di akhir; inilah StringBuffer
    versi Python, karena str tidak pernah dapat tumbuh di tempat.
    """
    digits = []
    for letter in word.lower():
        digits.append(_KEY_OF.get(letter, " "))
    return "".join(digits)


def is_valid_word(word):
    """True bila kata tidak kosong dan hanya berisi huruf a sampai z."""
    return len(word) > 0 and all("a" <= c <= "z" for c in word)


def signature_to_words(signature):
    """Kembalikan himpunan kata kamus (huruf kecil) yang memiliki signature ini.

    Sengaja tidak efisien: setiap panggilan membuka berkas, membaca SELURUHNYA, dan
    menghitung ulang signature setiap kata.
    """
    result = set()
    with open(DICTIONARY, encoding="utf-8") as f:
        for line in f:
            word = line.strip().lower()
            if is_valid_word(word) and word_to_signature(word) == signature:
                result.add(word)
    return result
```

Kode 12.4 `prototype_check.py`: uji Soal 12.1 dan 12.2 dalam Python, dengan varian keliru yang sama dan pengukuran waktu tiga cara membangun string.

```
# prototype_check.py - uji Soal 12.1 dan 12.2, dengan dua penyelesaian
# keliru yang menggoda dan pengukuran waktu += lawan list yang digabung sekali.
import sys
import time
import unittest

from predictive.prototype import is_valid_word, signature_to_words, word_to_signature


def word_to_signature_wrong(word):
    """Menggoda tetapi salah: aritmetika "tiga huruf per tombol"."""
    return "".join(chr(ord("2") + (ord(c) - ord("a")) // 3) for c in word)


def signature_to_words_wrong(signature):
    """Menggoda tetapi salah: sebuah list, tanpa huruf kecil, tanpa pemeriksaan
keabsahan."""
    with open("words.txt", encoding="utf-8") as f:
        return [line.rstrip("\n") for line in f
                if word_to_signature(line.rstrip("\n")) == signature]


def concat_with_str(n):
    s = ""
    for _ in range(n):
        s = s + "2"
    return s


def concat_with_str_shared(n):
    s = ""
    for _ in range(n):
        keep = s                    # referensi kedua menggagalkan jalan pintas CPython
        s = s + "2"
    return s + keep[:0]


def concat_with_join(n):
    parts = []
    for _ in range(n):
        parts.append("2")
    return "".join(parts)
```

Kode 12.4 (Lanjutan) *prototype_check.py: uji Soal 12.1 dan 12.2 dalam Python, dengan varian keliru yang sama dan pengukuran waktu tiga cara membangun string.*

```
class TestPrototype(unittest.TestCase):
    def test_signatures(self):
        self.assertEqual("4663", word_to_signature("home"))
        self.assertEqual("43556", word_to_signature("hello"))
        self.assertEqual("7777 9999", word_to_signature("pqrs wxyz"))
        self.assertEqual("4663", word_to_signature("Home"))
        self.assertEqual("366 8", word_to_signature("don't"))
        self.assertEqual("", word_to_signature(""))

    def test_valid_words(self):
        self.assertTrue(is_valid_word("laundry"))
        self.assertFalse(is_valid_word("e-mail"))
        self.assertFalse(is_valid_word(""))

    def test_lookup(self):
        self.assertEqual(["gone", "good", "goof", "home", "hone", "hood", "hoof"],
                         sorted(signature_to_words("4663")))
        self.assertEqual(["kiss", "lips"], sorted(signature_to_words("5477")))
        self.assertEqual(set(), signature_to_words("22222"))


if __name__ == "__main__":
    print("wrong 'you'      :", word_to_signature_wrong("you"),
          "  right:", word_to_signature("you"))
    print("wrong 4663 list  :", signature_to_words_wrong("4663"))
    for n in (20_000, 40_000, 80_000):
        t0 = time.perf_counter()
        concat_with_str(n)
        t1 = time.perf_counter()
        concat_with_str_shared(n)
        t2 = time.perf_counter()
        concat_with_join(n)
        t3 = time.perf_counter()
        print(f"n = {n:6d}   str +: {(t1 - t0) * 1e3:6.2f} ms   "
              f"str + (shared): {(t2 - t1) * 1e3:7.2f} ms   join: {(t3 - t2) * 1e3:5.2f}
ms")
    sys.stdout.flush()
    unittest.main(verbosity=2)
```

Jalankan dari folder `python` dengan `python -m predictive.prototype_check`:

```
wrong 'you'      : :68   right: 968
wrong 4663 list  : ['gone', 'good', 'goof', 'home', 'Home', 'hone', 'hood', 'hoof']
n =  20000   str +:   0.70 ms   str + (shared):    2.60 ms   join:  0.48 ms
n =  40000   str +:   1.23 ms   str + (shared):   15.39 ms   join:  0.96 ms
n =  80000   str +:   2.66 ms   str + (shared):   75.86 ms   join:  2.10 ms
test_lookup (__main__.TestPrototype.test_lookup) ... ok
test_signatures (__main__.TestPrototype.test_signatures) ... ok
test_valid_words (__main__.TestPrototype.test_valid_words) ... ok

----------------------------------------------------------------------
Ran 3 tests in 0.000s

OK
```

Pengukuran Python menyimpan kejutan: gelung `s = s + "2"` yang polos *tidak* kuadratik. CPython, interpreter Python yang baku, diam-diam memperpanjang string di tempat bila tidak ada hal lain yang merujuknya. Kolom tengah memperlihatkan betapa rapuhnya jalan pintas itu: satu referensi tambahan (`keep = s`) mematikannya, dan waktunya tumbuh lima sampai enam kali lipat setiap kali n berlipat dua, hukum kuadrat lagi. Itulah sebabnya dokumentasi Python menganjurkan `join` [4].

Baris demi baris. Pada **Kode 12.1**, `new StringBuffer(word.length())` memesan ruang yang pas. Gelungnya membaca setiap karakter dengan `word.charAt(i)`, mengecilkannya dengan `Character.toLowerCase`, lalu menambahkan `keyOf`-nya. `keyOf` adalah *switch expression*: setiap `case` mendaftar huruf-huruf satu tombol, panah memberikan nilainya, dan `default` menghasilkan spasi untuk hal lainnya. `result.toString()` membuat satu `String` di akhir. Pada **Kode 12.3**, dua gelung bersarang di tingkat modul mengisi `_KEY_OF` dengan 26 entri (garis bawah menandainya sebagai urusan internal modul), `_KEY_OF.get(letter, " ")` mengembalikan tombol atau spasi, dan `"".join(digits)` merekatkan *list* itu.

Penyelesaian keliru yang menggoda. Tiga huruf per tombol mengundang aritmetika: beri nomor huruf-huruf dari 0 (`a`) sampai 25 (`z`) lalu hitung

$$\kappa_{\text{wrong}}(c) = 2 + \left\lfloor \frac{\text{index}(c)}{3} \right\rfloor$$

dengan $\lfloor x \rfloor$ berarti "bulatkan ke bawah". `wordToSignatureWrong` melakukan hal ini, dan untuk `"you"` ia mencetak `:68`, bukan `968`. Tombol 7 memuat *empat* huruf, sehingga mulai dari s rumus itu selalu satu tombol terlalu jauh: s jatuh ke 8, v ke 9, serta y dan z ke "10", yang sebagai karakter adalah karakter sesudah `9`, yaitu titik dua. Rumus itu memberi tombol yang salah untuk tepat empat huruf, yaitu s, v, y, dan z, lolos dari setiap uji yang dibangun dari "home" dan "hello", dan hanya tertangkap oleh uji yang memakai tombol empat huruf, seperti pemeriksaan kita `four-letter keys`. Tabel tidak mungkin berbuat kesalahan ini.

> **UJI DIRI.** Apa hasil `wordToSignature` untuk `"Laundry"`, dan untuk `"e-mail"`?
>
> *Jawaban.* `"5286379"` (huruf kapital L dikecilkan lebih dulu), dan `"3 6245"`: tanda hubung menjadi spasi, sehingga signature tetap memiliki enam posisi.

12.4.2 Soal 12.2 — Dari signature ke kata-katanya, cara yang lambat

> **SOAL.** In the same class, write `public static Set<String> signatureToWords(String signature)`. It takes a numeric signature and returns the set of dictionary words that match it, in lower case and without duplicates. The method must use the dictionary file to find the matching words, and it must not store the dictionary in the program. Explain in a comment why this implementation is inefficient. A helper `isValidWord(String word)` that decides whether a line of the dictionary is a proper word is recommended.
>
> *Dengan kata lain:* dalam kelas yang sama, tulislah `signatureToWords` yang menerima signature dan mengembalikan himpunan kata kamus yang cocok, berhuruf kecil dan tanpa duplikat. Metode ini harus membaca berkas kamus dan tidak boleh menyimpan kamus di dalam program. Jelaskan dalam komentar mengapa implementasi ini tidak efisien. Metode bantu `isValidWord` yang memutuskan apakah sebuah baris kamus merupakan kata yang layak dianjurkan.

Dengan bahasa kita sendiri. Untuk setiap pertanyaan, buka berkasnya, baca dari atas sampai bawah, dan simpan setiap baris yang merupakan kata layak dengan signature yang diminta. Hasilnya berupa *himpunan*, sehingga `home` dan `Home`, yang menjadi kata yang sama setelah dikecilkan, hanya muncul sekali.

Rancangan. Tipe kembaliannya `Set` karena `add` pada himpunan yang sudah memuat kata itu tidak berbuat apa-apa: "tanpa duplikat" dijamin oleh tipenya. Berkas dibuka dalam pernyataan *try-with-resources* (Bab 5), sehingga selalu ditutup apa pun yang terjadi. Biayanya adalah seluruh berkas: dengan n kata yang rata-rata panjangnya $\bar{m}$, satu pencarian membaca dan mengonversi semuanya,

$$T_{\text{proto}}(n) = O(n\,\bar{m}) \quad \text{per pencarian,} \tag{12.3}$$

dan q pencarian memakan q kali lipat, karena tidak ada yang diingat di antara pemanggilan. Itulah ketidakefisienan yang dijelaskan komentarnya dan yang dihilangkan oleh Bagian Dua dan Tiga.

Penyelesaiannya adalah `signatureToWords` pada **Kode 12.1** dan `signature_to_words` pada **Kode 12.3**. Uji-ujinya (`4663`, `5477`, `no match`, `test_lookup`) sudah lolos di atas: 4663 menghasilkan gone, good, goof, home, hone, hood, dan hoof, dan 5477 menghasilkan kiss dan lips, gangguan terkenal dari teks prediktif.

Baris demi baris. `Scanner` dibuat dengan *encoding* `UTF_8` yang eksplisit, sehingga `café` terbaca sama di setiap sistem. `hasNextLine` dan `nextLine` membaca baris demi baris; `.trim().toLowerCase()` membersihkan setiap baris. Dalam `isValidWord(word) && wordToSignature(word).equals(signature)`, `&&` berhenti pada `false` yang pertama, sehingga baris yang tidak sah tidak pernah dikonversi, dan string dibandingkan dengan `equals`, tidak pernah dengan `==` (Bab 3). `catch` mengubah `IOException` yang *checked* menjadi `IllegalStateException` yang *unchecked*, karena kamus yang hilang tidak dapat diperbaiki oleh pemanggil. `isValidWord` menolak string kosong, lalu membandingkan setiap karakter dengan rentang `'a'` sampai `'z'`; versi Python-nya adalah `all("a" <= c <= "z" for c in word)`, dan `with open(...)` berperan sebagai *try-with-resources*.

Penyelesaian keliru yang menggoda. Versi yang ditulis terburu-buru, `signatureToWordsWrong`, membaca baris-barisnya, menyimpan baris yang signature-nya cocok, dan mengembalikan sebuah `List`. Jawabannya untuk 4663 memuat `Home` di samping `home`: baris-barisnya tidak pernah dikecilkan, dan *list* menyimpan keduanya. Versi itu juga tidak pernah memangkas, sehingga `soap` mendapat signature dengan spasi di sekelilingnya dan hilang tanpa jejak. Tidak ada uji dengan kata-kata bersih berhuruf kecil yang menemukan kesalahan-kesalahan itu; kamus yang realistis menemukan keduanya. Jebakan klasik kedua terletak pada penutupan berkas secara manual: dengan `Scanner fileScan = null; try { fileScan = new Scanner(...); ... } finally { fileScan.close(); }`, berkas yang tidak ada membuat `fileScan` tetap `null`, dan blok `finally` melempar `NullPointerException` yang menyembunyikan galat yang sebenarnya. *Try-with-resources* tidak pernah menutup apa yang tidak pernah dibuka.

> **UJI DIRI.** Tambahkan satu uji murah yang membuat `signatureToWords` melewatkan sebagian besar kata. Apakah itu mengubah Persamaan (12.3)?
>
> *Jawaban.* Uji `word.length() == signature.length()` lebih dulu (signature memiliki satu angka per huruf). Uji ini melewatkan sebagian besar konversi, tetapi seluruh berkas tetap dibaca, sehingga biaya per pencarian tetap sebanding dengan n.

12.4.3 Soal 12.3 — Dua program baris perintah

> **SOAL.** Write two command-line programs (classes with a `main` method): `Words2SigProto`, which calls `wordToSignature` for every word given on the command line, and `Sigs2WordsProto`, which calls `signatureToWords` for every signature given on the command

> line. For example, `Words2SigProto home hello world my name is` should print `input: [home, hello, world, my, name, is]` and `output: 4663 43556 96753 69 6263 47`, and `Sigs2WordsProto` should print one line per signature, the signature followed by its words.
>
> *Dengan kata lain:* tulislah dua program baris perintah: `Words2SigProto` mencetak signature setiap kata yang diberikan di baris perintah, dan `Sigs2WordsProto` mencetak kata-kata setiap signature, satu baris per signature, dengan format seperti contoh.

Dengan bahasa kita sendiri. Kata-kata sesudah nama program tiba sebagai larik `args` (Java) atau *list* `sys.argv` (Python). Program pertama mengubah masing-masing menjadi signature; program kedua mencari masing-masing dan mencetak satu baris per signature.

Rancangan. Keduanya adalah gelung atas argumen-argumen itu. Satu hal memerlukan kecermatan: keluaran yang *dapat diulang*. `HashSet` mencetak elemennya dalam urutan yang bergantung pada kode *hash*-nya, dan Python bahkan mengubah kode *hash* string dari satu eksekusi ke eksekusi berikutnya (*hash randomization*), jadi kita mengurutkan sebelum mencetak: `TreeSet` di Java, `sorted()` di Python.

Penyelesaian Java. Kode 12.5 dan **Kode 12.6**.

Kode 12.5 `Words2SigProto.java`: signature setiap kata di baris perintah.

```java
// Words2SigProto.java - Soal 12.3: mencetak signature setiap kata
// yang diberikan di baris perintah.
package predictive;

import java.util.Arrays;

public class Words2SigProto {

    public static void main(String[] args) {
        if (args.length == 0) {
            System.out.println("USAGE: java predictive.Words2SigProto word1 word2 ...
wordN");
            return;
        }
        System.out.println("input: " + Arrays.toString(args));
        StringBuilder output = new StringBuilder("output:");
        for (String word : args) {
            output.append(' ').append(PredictivePrototype.wordToSignature(word));
        }
        System.out.println(output);
    }
}
```

Kode 12.6 `Sigs2WordsProto.java`: kata-kata setiap signature di baris perintah, satu baris per signature, menurut abjad.

```java
// Sigs2WordsProto.java - Soal 12.3: mencetak kata-kata yang cocok untuk setiap
// signature yang diberikan di baris perintah, satu baris per signature.
package predictive;

import java.util.TreeSet;

public class Sigs2WordsProto {

    public static void main(String[] args) {
        if (args.length == 0) {
            System.out.println("USAGE: java predictive.Sigs2WordsProto sig1 sig2 ...
sigN");
            return;
        }
        for (String sig : args) {
            // TreeSet mencetak kata menurut abjad, sama pada setiap eksekusi
            System.out.println(sig + ": " + new
TreeSet<>(PredictivePrototype.signatureToWords(sig)));
        }
    }
}
```

`java predictive.Words2SigProto home hello world my name is` mencetak:

```
input: [home, hello, world, my, name, is]
output: 4663 43556 96753 69 6263 47
```

dan `java predictive.Sigs2WordsProto 4663 43556 96753 69 6263 47` mencetak:

```
4663: [gone, good, goof, home, hone, hood, hoof]
43556: [hello]
96753: [world]
69: [my, ow, ox]
6263: [mane, name, oboe]
47: [is]
```

Penyelesaian Python. **Kode 12.7** dan **Kode 12.8**, yang dijalankan dengan `python -m predictive.words2sig_proto ...` dan `python -m predictive.sigs2words_proto ...`, mencetak baris-baris yang persis sama.

Kode 12.7 `words2sig_proto.py`: program yang sama dalam Python; kata-katanya adalah `sys.argv[1:]`.

```python
# words2sig_proto.py - Soal 12.3: mencetak signature setiap kata
# yang diberikan di baris perintah.
import sys

from predictive.prototype import word_to_signature


def main(args):
    if not args:
        print("USAGE: python -m predictive.words2sig_proto word1 word2 ... wordN")
        return
    print("input: [" + ", ".join(args) + "]")
    print("output: " + " ".join(word_to_signature(word) for word in args))


if __name__ == "__main__":
    main(sys.argv[1:])                # sys.argv[0] adalah program itu sendiri
```

Kode 12.8 `sigs2words_proto.py`: kata-kata setiap signature di baris perintah, terurut.

```python
# sigs2words_proto.py - Soal 12.3: mencetak kata-kata yang cocok untuk setiap
# signature yang diberikan di baris perintah, satu baris per signature.
import sys

from predictive.prototype import signature_to_words


def main(args):
    if not args:
        print("USAGE: python -m predictive.sigs2words_proto sig1 sig2 ... sigN")
        return
    for sig in args:
        # sorted() memberi urutan abjad, sama pada setiap eksekusi
        print(sig + ": [" + ", ".join(sorted(signature_to_words(sig))) + "]")


if __name__ == "__main__":
    main(sys.argv[1:])
```

```
input: [home, hello, world, my, name, is]
output: 4663 43556 96753 69 6263 47
4663: [gone, good, goof, home, hone, hood, hoof]
43556: [hello]
96753: [world]
69: [my, ow, ox]
6263: [mane, name, oboe]
47: [is]
```

Dengan kamus seperempat juta kata, seperti yang disertakan di banyak sistem Linux, perintah yang sama juga menemukan entri-entri aneh, misalnya `yorke` untuk 96753; programnya tidak berubah, hanya berkasnya.

Baris demi baris. `if` mencetak pesan cara pakai bila tidak ada argumen. `Arrays.toString(args)` memformat larik menjadi `[home, hello, ...]`, dan baris keluaran tumbuh dalam sebuah `StringBuilder`. `new TreeSet<>(...)` menyalin jawabannya ke himpunan terurut yang `toString()`-nya mendaftarnya secara urut. Di Python, `sys.argv[1:]` membuang nama program itu sendiri, dan `", ".join(sorted(...))` membangun daftar berkurung siku.

Penyelesaian keliru yang menggoda. Mencetak argumen secara langsung keliru dalam kedua bahasa, dengan cara yang berbeda. Di Java, `System.out.println("input: " + args)` mencetak

```
input: [Ljava.lang.String;@659e0bfd
```

karena larik tidak meng-*override* `toString()`: `[L` berarti "larik berisi", diikuti nama kelas dan kode *hash* yang berubah dari eksekusi ke eksekusi. Di Python, `print("input:", sys.argv)` mencetak

```
input: ['words2sig_wrong.py', 'home', 'hello']
```

dengan nama berkas program di depan. `sys.argv[0]` di Python adalah programnya; `args[0]` di Java sudah merupakan argumen pertama, selisih satu yang patut diingat ketika memindahkan program baris perintah dari satu bahasa ke bahasa lain.

> **UJI DIRI.** Apa yang dicetak `Words2SigProto don't panic`, dan berapa argumen yang diterima `Sigs2WordsProto` bila baris keluarannya Anda tempelkan sesudah nama program itu?
>
> *Jawaban.* `input: [don't, panic]` dan `output: 366 8 72642`. Spasi di dalam `366 8` memisahkan dua argumen, sehingga `Sigs2WordsProto` menerima tiga: tanda baca memotong signature menjadi dua, itulah sebabnya soal membolehkan kita mengabaikannya.

12.5 Bagian Dua: *List* Terurut dan Antarmuka

Telepon yang tersendat setelah setiap tekanan tombol terasa rusak, padahal prototipe membaca seluruh kamus untuk setiap pertanyaan. Obatnya: baca kamus *sekali*, simpan di memori, dan susun sedemikian rupa sehingga pertanyaan dapat dijawab tanpa melihat setiap kata. Bagian Dua memakai susunan paling sederhana yang memenuhinya, yaitu *list* terurut.

12.5.1 Soal 12.4 — *List* terurut berisi pasangan dan pencarian biner

> **SOAL.** Write a class `DictionaryListImpl`. Its constructor reads the dictionary file and stores it in an `ArrayList`, each entry of which pairs a word with its signature; write a class `WordSig` for these pairs, with `private` fields for the word and the signature, and let it implement `Comparable<WordSig>`. The list must be sorted, once, in the *numerical* order of the signatures (which is not the alphabetical order of the signature strings). The method `signatureToWords` must use binary search, `Collections.binarySearch`, to find the words of a signature. Binary search returns the index of *one* match; the method must return *all* matching words, so scan above and below the index it finds. For example, `signatureToWords("4663")` must give the thirteen or so words that a large dictionary has for 4663, and `signatureToWords("43556")` must give `hello` (and `gekko`, if the dictionary has it).
>
> *Dengan kata lain:* tulislah `DictionaryListImpl` yang membaca kamus ke dalam `ArrayList` berisi pasangan kata dan signature (kelas `WordSig` yang mengimplementasikan `Comparable<WordSig>`), mengurutkannya sekali menurut nilai *numerik* signature, dan menjawab `signatureToWords` dengan `Collections.binarySearch`. Karena pencarian biner hanya menemukan *satu* kecocokan, pindai ke atas dan ke bawah indeksnya untuk mengumpulkan *semua* kata yang cocok.

Dengan bahasa kita sendiri. Ubah kamus menjadi kartu-kartu, masing-masing bertuliskan signature dan kata, lalu urutkan menurut signature, seperti buku telepon diurutkan menurut nama; kartu-kartu dengan signature yang sama lalu berdampingan. Untuk menjawab pertanyaan, temukan satu kartu dengan signature yang benar dengan berulang kali membelah dua bagian *list* tempat kartu itu mungkin berada, lalu kumpulkan tetangganya ke atas dan ke bawah selama signature-nya sama.

Rancangan. Tiga gagasan membentuk rancangan ini.

Pertama, urutannya. `Collections.sort` dan `Collections.binarySearch` menanyakan kepada elemen-elemennya mana yang lebih dulu, melalui `Comparable<WordSig>`: `compareTo(WordSig other)` mengembalikan bilangan negatif, nol, atau bilangan positif [5]. Soal meminta urutan *numerik*, dan kata itu menyembunyikan jebakan: `String.compareTo` membandingkan karakter demi karakter, sehingga menaruh `"22"` sebelum `"3"`, padahal $22 > 3$. Mengonversi ke bilangan punya jebakannya sendiri (lihat penyelesaian keliru). Obat yang bersih memakai sebuah fakta tentang bilangan tanpa nol di depan, dan signature kita (angka 2 sampai 9) tidak pernah memilikinya: *lebih sedikit angka berarti lebih kecil*, dan panjang yang sama dibandingkan persis seperti untai angkanya:

$$a < b \Leftrightarrow |a| < |b| \ \text{ atau } \ (|a| = |b| \ \text{ dan } \ a <_{\text{str}} b) \tag{12.4}$$

dengan $|a|$ banyaknya angka pada a dan $<_{\text{str}}$ urutan string biasa. Persamaan (12.4) tidak memerlukan konversi dan berlaku untuk signature sepanjang apa pun.

Kedua, pencariannya. Pencarian biner menyimpan dua indeks, `low` dan `high`, dan melihat kartu di tengah: bila signature-nya lebih kecil daripada yang dicari, kartu-kartu yang dicari terletak di

kanannya (`low` bergeser melewati tengah); bila lebih besar, di kirinya (`high` bergeser sebelum tengah); bila sama, kecocokan ditemukan. Setiap tengokan membelah dua bagian yang masih harus dicari [6], sehingga untuk n kartu paling banyak

$$\lceil \log_2(n+1) \rceil \tag{12.5}$$

kartu yang diperiksa, dengan $\lceil x \rceil$ berarti "bulatkan ke atas". Untuk 93 kartu kita, itu 7; untuk 200.000 kartu, 18; untuk sejuta, 20. Pencarian ini tidak pernah melihat sebagian besar *list*.

Ketiga, tetangganya. Pencarian biner berhenti pada kartu cocok mana pun yang pertama kali ditemuinya. **Tabel 12.4** memperlihatkan eksekusi sebenarnya untuk 4663 pada kamus kita: empat tengokan, berakhir di indeks 40, `goof`, di *tengah* blok kartu 4663 (indeks 38 sampai 45). Berjalan ke bawah menemukan `good` dan `gone`; berjalan ke atas menemukan `home` dua kali (dari `home` dan `Home`), `hone`, `hood`, dan `hoof`; penelusuran berhenti di `help` (4357) dan `iron` (4766).

Tabel 12.4 Pencarian biner untuk 4663 di antara 93 kartu terurut dari `words.txt`. Empat tengokan menemukan satu kecocokan; sisanya terletak di indeks 38 sampai 45, di sekitarnya.

Tengokan	`low`	`high`	Tengah	Kartu di tengah	Keputusan
1	0	92	46	`4766:iron`	lebih besar daripada 4663: `high = 45`
2	0	45	22	`228:bat`	lebih kecil (3 angka): `low = 23`
3	23	45	34	`2665:book`	lebih kecil: `low = 35`
4	35	45	40	`4663:goof`	sama: ditemukan indeks 40

Secara keseluruhan, membangun *list* memakan satu pengurutan, dan satu pencarian memakan satu pencarian biner ditambah satu langkah per kecocokan:

$$T_{\text{build}}(n) = O(n \log n), \qquad T_{\text{list}}(n) = O(\bar{m} \log n + k) \quad \text{per pencarian,} \tag{12.6}$$

dengan k banyaknya kata yang cocok dan $\bar{m}$ panjang signature, yaitu biaya terburuk setiap perbandingan.

Penyelesaian Java. **Kode 12.9** adalah kelas pasangannya; **Kode 12.10** kamusnya. `DictionaryListImpl` menyatakan `implements Dictionary`; antarmuka itu, yang hanya satu metode, dirancang pada Soal 12.5.

Kode 12.9 `WordSig.java`: signature dan kata, dibandingkan menurut urutan numerik Persamaan (12.4).

```java
// WordSig.java - Soal 12.4: kata yang dipasangkan dengan signature-nya, diurutkan menurut
// nilai NUMERIK signature.
package predictive;

public class WordSig implements Comparable<WordSig> {

    private final String signature;
    private final String word;

    public WordSig(String signature, String word) {
        this.signature = signature;
        this.word = word;
    }

    public String getSignature() {
        return signature;
    }

    public String getWord() {
        return word;
    }

    /**
     * Urutan numerik dua untai angka tanpa nol di depan: bilangan yang lebih
     * pendek lebih kecil; panjang sama dibandingkan angka demi angka, persis
     * yang dilakukan String.compareTo. Tidak perlu konversi ke bilangan,
     * sehingga signature sepanjang apa pun dibandingkan dengan benar.
     */
    @Override
    public int compareTo(WordSig other) {
        if (signature.length() != other.signature.length()) {
            return Integer.compare(signature.length(), other.signature.length());
        }
        return signature.compareTo(other.signature);
    }

    @Override
    public String toString() {
        return signature + ":" + word;
    }
}
```

Kode 12.10 `DictionaryListImpl.java`: kamus sebagai `ArrayList<WordSig>` terurut, dicari dengan `Collections.binarySearch` dan penelusuran ke kedua sisi.

```
// DictionaryListImpl.java - Soal 12.4: kamus di memori sebagai ArrayList terurut
// berisi pasangan WordSig, dicari dengan pencarian biner.
package predictive;

import java.io.File;
import java.io.IOException;
import java.nio.charset.StandardCharsets;
import java.util.ArrayList;
import java.util.Collections;
import java.util.HashSet;
import java.util.Scanner;
import java.util.Set;

public class DictionaryListImpl implements Dictionary {

    private final ArrayList<WordSig> dictionary = new ArrayList<>();

    /** Membaca setiap kata sah dari berkas sekali, lalu mengurutkan pasangannya sekali. */
    public DictionaryListImpl(String path) throws IOException {
        try (Scanner in = new Scanner(new File(path), StandardCharsets.UTF_8)) {
            while (in.hasNextLine()) {
                String word = in.nextLine().trim().toLowerCase();
                if (PredictivePrototype.isValidWord(word)) {
                    add(word);
                }
            }
        }
        Collections.sort(dictionary);
    }

    private void add(String word) {
        dictionary.add(new WordSig(PredictivePrototype.wordToSignature(word), word));
    }

    @Override
    public Set<String> signatureToWords(String signature) {
        Set<String> result = new HashSet<>();
        // pasangan "tiruan": pencarian biner hanya membandingkan signature
        int index = Collections.binarySearch(dictionary, new WordSig(signature, ""));
        if (index < 0) {
            return result;                      // tidak ditemukan: -(titik sisip) - 1
        }
        // binarySearch menemukan SALAH SATU yang cocok; sisanya adalah tetangganya
        for (int i = index; i >= 0 && matches(i, signature); i--) {
            result.add(dictionary.get(i).getWord());
        }
```

Kode 12.10 (Lanjutan) *DictionaryListImpl.java: kamus sebagai ArrayList<WordSig> terurut, dicari dengan Collections.binarySearch dan penelusuran ke kedua sisi.*

```
        for (int i = index + 1; i < dictionary.size() && matches(i, signature); i++) {
            result.add(dictionary.get(i).getWord());
        }
        return result;
    }

    private boolean matches(int i, String signature) {
        return dictionary.get(i).getSignature().equals(signature);
    }

    /** Banyaknya pasangan yang disimpan (duplikat dalam berkas disimpan dua kali). */
    public int size() {
        return dictionary.size();
    }
}
```

Kode 12.11 menguji kelas itu dan memperagakan dua penyelesaian keliru yang dibahas di bawah.

Kode 12.11 `ListCheck.java`: pemeriksaan Soal 12.4, kedua urutan signature berdampingan, dan dua varian keliru.

```
// ListCheck.java - pemeriksaan Soal 12.4, dengan dua penyelesaian keliru yang menggoda:
// hanya menyimpan temuan pertama pencarian biner, dan mengurutkan signature sebagai
double.
package predictive;

import java.nio.file.Files;
import java.nio.file.Path;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.List;
import java.util.Set;
import java.util.TreeSet;

public class ListCheck {

    /** Kata sah dari words.txt sebagai pasangan WordSig, diurutkan menurut urutan yang
diberikan. */
    static List<WordSig> load(Comparator<WordSig> order) throws Exception {
        List<WordSig> list = new ArrayList<>();
        for (String line : Files.readAllLines(Path.of("words.txt"))) {
            String word = line.trim().toLowerCase();
            if (PredictivePrototype.isValidWord(word)) {
                list.add(new WordSig(PredictivePrototype.wordToSignature(word), word));
            }
        }
        list.sort(order);
        return list;
    }

    /** Menggoda tetapi salah: hanya mengembalikan pasangan tempat pencarian biner
mendarat. */
    static Set<String> firstHitOnly(List<WordSig> list, String signature) {
        int i = Collections.binarySearch(list, new WordSig(signature, ""));
        return (i < 0) ? Set.of() : Set.of(list.get(i).getWord());
    }

    /** Menggoda tetapi salah: membandingkan signature sebagai nilai double. */
    static final Comparator<WordSig> BY_DOUBLE = (a, b) -> Double.compare(
            Double.parseDouble(a.getSignature()), Double.parseDouble(b.getSignature()));

    private static int passed = 0;
```

Kode 12.11 (Lanjutan) *ListCheck.java: pemeriksaan Soal 12.4, kedua urutan signature berdampingan, dan dua varian keliru.*

```
    private static void check(String name, Object expected, Object actual) {
        if (!java.util.Objects.equals(expected, actual)) {
            throw new AssertionError(name + ": expected " + expected + " but got " +
actual);
        }
        passed++;
        System.out.println("PASS " + name);
    }

    private static String sorted(Set<String> words) {
        return new TreeSet<>(words).toString();
    }

    public static void main(String[] args) throws Exception {
        DictionaryListImpl d = new DictionaryListImpl("words.txt");
        check("size", 93, d.size());
        check("4663", "[gone, good, goof, home, hone, hood, hoof]",
sorted(d.signatureToWords("4663")));
        check("43556", "[hello]", sorted(d.signatureToWords("43556")));
        check("5477", "[kiss, lips]", sorted(d.signatureToWords("5477")));
        check("single key", "[a]", sorted(d.signatureToWords("2")));
        check("empty", "[]", sorted(d.signatureToWords("")));
        check("no such key", "[]", sorted(d.signatureToWords("1")));
        check("absent", "[]", sorted(d.signatureToWords("22222")));

        List<WordSig> list = load(Comparator.naturalOrder());
        System.out.println("first pairs      : " + list.subList(0, 8));
        List<WordSig> three = new ArrayList<>(List.of(
                new WordSig("7", "p"), new WordSig("22", "ab"), new WordSig("3", "d")));
        Collections.sort(three);
        System.out.println("numerical order  : " + three);
        three.sort(Comparator.comparing(WordSig::getSignature));
        System.out.println("String order     : " + three);

        System.out.println("first hit only   : " + firstHitOnly(list, "4663"));

        List<WordSig> longs = new ArrayList<>(List.of(
                new WordSig("22222222222222222", "aaaaaaaaaaaaaaaa"),
                new WordSig("22222222222222223", "aaaaaaaaaaaaaaad"),
                new WordSig("22222222222222224", "aaaaaaaaaaaaaaag")));
        longs.sort(BY_DOUBLE);
        System.out.println("as doubles equal : "
                + (BY_DOUBLE.compare(longs.get(0), longs.get(2)) == 0));
        int i = Collections.binarySearch(longs, new WordSig("22222222222222222", ""),
BY_DOUBLE);
        System.out.println("search ...222 by double lands on " + longs.get(i));
        System.out.println("All " + passed + " checks passed.");
    }
}
```

```
PASS size
PASS 4663
PASS 43556
PASS 5477
PASS single key
PASS empty
PASS no such key
PASS absent
first pairs      : [2:a, 23:be, 43:he, 43:id, 43:if, 46:go, 46:in, 47:is]
numerical order  : [3:d, 7:p, 22:ab]
String order     : [22:ab, 3:d, 7:p]
first hit only   : [goof]
as doubles equal : true
search ...222 by double lands on 22222222222222223:aaaaaaaaaaaaaaad
All 8 checks passed.
```

Baris `first pairs` memperlihatkan *list* terurut yang diawali signature satu angka milik `a`, dan baris `numerical order` serta `String order` memperlihatkan perbedaan yang menjadi pokok Persamaan (12.4).

Penyelesaian Python. Python tidak memiliki `Comparable`; pengurutan dan modul `bisect` memakai `<`, yang memanggil metode `__lt__`. `WordSig` pada **Kode 12.12** mendefinisikannya dengan membandingkan tuple `(len(signature), signature)`, yang dibandingkan elemen demi elemen: Persamaan (12.4) dalam satu baris. Lagi pula, `bisect.bisect_left` mengembalikan posisi *pertama* tempat kunci dapat disisipkan, yang untuk kunci yang ada adalah kartu pertama bloknya, sehingga **Kode 12.13** cukup berjalan *ke atas*. **Kode 12.14** mengulang uji dan varian kelirunya.

Kode 12.12 `word_sig.py`: kelas pasangan dalam Python; `__lt__` adalah satu-satunya yang diperlukan pengurutan dan `bisect`.

```
# word_sig.py - Soal 12.4: kata yang dipasangkan dengan signature-nya, diurutkan menurut
# nilai NUMERIK signature.


class WordSig:
    __slots__ = ("signature", "word")

    def __init__(self, signature, word):
        self.signature = signature
        self.word = word

    def sort_key(self):
        """Urutan numerik tanpa konversi: yang lebih pendek dahulu, lalu angka demi
angka."""
        return (len(self.signature), self.signature)

    def __lt__(self, other):            # hanya itu yang diperlukan sort() dan bisect
        return self.sort_key() < other.sort_key()

    def __repr__(self):
        return f"{self.signature}:{self.word}"
```

Kode 12.13 `dictionary_list_impl.py`: *list* terurut dalam Python, dicari dengan `bisect_left` dan penelusuran ke atas.

```python
# dictionary_list_impl.py - Soal 12.4: kamus di memori sebagai list terurut
# berisi pasangan WordSig, dicari dengan pencarian biner.
from bisect import bisect_left

from predictive.dictionary import Dictionary
from predictive.prototype import is_valid_word, word_to_signature
from predictive.word_sig import WordSig


class DictionaryListImpl(Dictionary):

    def __init__(self, path):
        """Baca setiap kata sah dari berkas sekali, lalu urutkan pasangannya sekali."""
        self._dictionary = []
        with open(path, encoding="utf-8") as f:
            for line in f:
                word = line.strip().lower()
                if is_valid_word(word):
                    self._add(word)
        self._dictionary.sort()

    def _add(self, word):
        self._dictionary.append(WordSig(word_to_signature(word), word))

    def signature_to_words(self, signature):
        result = set()
        # bisect_left mengembalikan posisi PERTAMA yang pasangannya tidak lebih kecil
        i = bisect_left(self._dictionary, WordSig(signature, ""))
        while i < len(self._dictionary) and self._dictionary[i].signature == signature:
            result.add(self._dictionary[i].word)
            i += 1
        return result

    def __len__(self):
        """Banyaknya pasangan yang disimpan (duplikat dalam berkas disimpan dua kali)."""
        return len(self._dictionary)
```

Kode 12.14 `list_check.py`: uji Soal 12.4 dalam Python, dengan dua varian keliru yang sama.

```python
# list_check.py - uji Soal 12.4, dengan dua penyelesaian keliru yang menggoda:
# hanya menyimpan temuan pertama pencarian biner, dan mengurutkan signature sebagai float.
import sys
import unittest
from bisect import bisect_left

from predictive.dictionary_list_impl import DictionaryListImpl
from predictive.prototype import is_valid_word, word_to_signature
from predictive.word_sig import WordSig


def load():
    with open("words.txt", encoding="utf-8") as f:
        words = [line.strip().lower() for line in f]
    return sorted(WordSig(word_to_signature(w), w) for w in words if is_valid_word(w))


def first_hit_only(pairs, signature):
    """Menggoda tetapi salah: hanya mengembalikan pasangan tempat pencarian biner
mendarat."""
    lo, hi = 0, len(pairs) - 1
    while lo <= hi:                       # pencarian buku teks: berhenti pada kecocokan
MANA PUN
        mid = (lo + hi) // 2
        if pairs[mid].signature == signature:
            return {pairs[mid].word}
        if pairs[mid] < WordSig(signature, ""):
            lo = mid + 1
        else:
            hi = mid - 1
    return set()


def by_float(pair):
    """Menggoda tetapi salah: membandingkan signature sebagai nilai float."""
    return float(pair.signature)


class TestListImpl(unittest.TestCase):
    @classmethod
    def setUpClass(cls):
        cls.d = DictionaryListImpl("words.txt")

    def test_size(self):
        self.assertEqual(93, len(self.d))
```

Kode 12.14 (Lanjutan) *list_check.py: uji Soal 12.4 dalam Python, dengan dua varian keliru yang sama.*

```
    def test_lookups(self):
        self.assertEqual(["gone", "good", "goof", "home", "hone", "hood", "hoof"],
                         sorted(self.d.signature_to_words("4663")))
        self.assertEqual({"hello"}, self.d.signature_to_words("43556"))
        self.assertEqual({"kiss", "lips"}, self.d.signature_to_words("5477"))
        self.assertEqual({"a"}, self.d.signature_to_words("2"))

    def test_nothing_matches(self):
        for sig in ("", "1", "22222"):
            self.assertEqual(set(), self.d.signature_to_words(sig))


if __name__ == "__main__":
    pairs = load()
    print("first pairs      :", pairs[:8])
    three = sorted([WordSig("7", "p"), WordSig("22", "ab"), WordSig("3", "d")])
    print("numerical order  :", three)
    print("str order        :", sorted(three, key=lambda p: p.signature))
    print("first hit only   :", first_hit_only(pairs, "4663"))
    longs = sorted([WordSig("22222222222222222", "aaaaaaaaaaaaaaaaa"),
                    WordSig("22222222222222223", "aaaaaaaaaaaaaaaad"),
                    WordSig("22222222222222224", "aaaaaaaaaaaaaaaag")], key=by_float)
    print("as floats equal  :", by_float(longs[0]) == by_float(longs[2]))
    i = bisect_left(longs, float("22222222222222223"), key=by_float)
    print("search ...223 by float lands on", longs[i])
    sys.stdout.flush()
    unittest.main(verbosity=2)
```

```
first pairs      : [2:a, 23:be, 43:he, 43:id, 43:if, 46:go, 46:in, 47:is]
numerical order  : [3:d, 7:p, 22:ab]
str order        : [22:ab, 3:d, 7:p]
first hit only   : {'goof'}
as floats equal  : True
search ...223 by float lands on 22222222222222222:aaaaaaaaaaaaaaaaa
test_lookups (__main__.TestListImpl.test_lookups) ... ok
test_nothing_matches (__main__.TestListImpl.test_nothing_matches) ... ok
test_size (__main__.TestListImpl.test_size) ... ok

----------------------------------------------------------------------
Ran 3 tests in 0.000s

OK
```

Baris demi baris. Pada **Kode 12.9**, kedua *field* bersifat `private final`: mengubah signature sebuah kartu di dalam *list* terurut akan diam-diam merusak urutan yang menjadi sandaran pencarian biner. `compareTo` membandingkan panjang dengan `Integer.compare`, yang tidak dapat meluap seperti pengurangan, dan baru memakai `String.compareTo` untuk panjang yang sama. Pada **Kode 12.10**, konstruktor membaca baris seperti prototipe dan mengurutkan *sekali*, setelah semua kartu masuk; mengurutkan setelah setiap kartu akan memakan $O(n^2 \log n)$. `new WordSig(signature, "")` adalah *probe* (kartu penjajak) yang katanya tidak penting, karena `compareTo` hanya melihat signature. `Collections.binarySearch` mengembalikan indeks sebuah kecocokan, atau $-(\text{titik sisip}) - 1$ bila tidak ada (angka -1 menjaga agar "tidak ditemukan" tetap negatif bahkan untuk titik sisip 0). Gelung pertama berjalan turun dari `index`, gelung kedua naik dari `index + 1`, dan `HashSet` melaporkan `home` sekali saja. Pada **Kode 12.13**, `bisect_left` memberikan kandidat pertama dan gelung `while` mengumpulkan signature yang sama; `__len__` memungkinkan kita menulis `len(d)`.

Penyelesaian keliru yang menggoda. Godaan pertama adalah berhenti di tempat pencarian biner berhenti. `firstHitOnly` melakukannya, dan menjawab `[goof]` untuk 4663: satu kata dari

tujuh. Uji dengan `hello`, yang sendirian di bloknya, tidak akan menyadarinya; selalu ujilah signature yang memiliki beberapa kata.

Godaan kedua muncul dalam kode yang cermat: membandingkan signature sebagai bilangan, `Double.parseDouble(a.getSignature())` terhadap `Double.parseDouble(b.getSignature())`. Sebuah `double` hanya menampung sekitar 15 sampai 16 angka desimal, padahal signature sepanjang katanya, dan `internationalization` terdiri atas 20 huruf. Lewat dari 16 angka, signature yang berbeda menjadi `double` yang *sama*: eksekusi memperlihatkan tiga signature 17 angka yang berakhiran 2, 3, dan 4 yang dianggap `equal`. Pencarian biner lalu mendarat di `...223` ketika diminta `...222`, dan penelusuran kita, yang membandingkan string, tidak menemukan apa pun padahal `...222` ada di dalam *list*. `float` di Python runtuh dengan cara yang sama; `long` hanya menggeser batasnya ke 18 angka. Persamaan (12.4) tepat dan murah.

> **UJI DIRI.** Pada **Tabel 12.4**, apa keputusan tengokan pertama untuk signature 5477 (kiss, lips), dan apakah jawabannya bergantung pada tempat pencarian itu mendarat?
>
> *Jawaban.* Indeks 46 berisi `4766:iron`, yang lebih kecil daripada 5477 (panjang sama, `4` sebelum `5`), sehingga `low = 47`, dan paling banyak $\lceil \log_2 47 \rceil = 6$ tengokan lagi menyusul. Di mana pun pencarian mendarat di blok dua kartu itu, penelusuran ke kedua sisi mengumpulkan kartu yang lain, sehingga jawabannya tetap `{kiss, lips}`; itulah sebabnya penelusuran berjalan ke dua arah.

12.5.2 Soal 12.5 — Antarmuka `Dictionary`

> **SOAL.** Based on the design of `DictionaryListImpl`, create and document a Java interface `Dictionary`. It will be implemented by the faster dictionaries of Part Three.
>
> *Dengan kata lain:* berdasarkan rancangan `DictionaryListImpl`, buat dan dokumentasikan antarmuka Java `Dictionary`, yang kelak diimplementasikan oleh kamus-kamus yang lebih cepat pada Bagian Tiga.

Dengan bahasa kita sendiri. Apa yang diperlukan seorang *pemakai* kamus? Hanya bertanya "kata apa saja yang cocok dengan signature ini?". Bagaimana kata-kata disimpan adalah urusan setiap implementasi. Karena itu, antarmukanya cukup memuat satu pertanyaan itu, dan dokumentasinya harus menyatakan dengan tepat jawabannya dalam setiap keadaan, termasuk keadaan yang canggung.

Rancangan. Langkah 3 dari Subbab 12.2 dalam praktik: untuk setiap anggota publik `DictionaryListImpl`, tanyakan apakah seorang *pemakai* memerlukannya. `signatureToWords`: ya. Konstruktor: tidak; antarmuka tidak memiliki konstruktor, dan pembuatan objek adalah satu-satunya tempat kelas konkret disebut. `size()`: tidak. `ArrayList<WordSig>`: jelas tidak, sebab kalau ya, *map* dan pohon terpaksa memilikinya juga. Lalu keadaan-keadaan canggung ditetapkan secara tertulis: tidak pernah `null`, tanpa duplikat, tanpa urutan tertentu, dan signature yang memuat selain angka 2 sampai 9 tidak cocok dengan apa pun. Karena pohon pada Soal 12.8 juga mengembalikan prefiks, kontraknya membolehkan hal itu. Kontrak yang tepat inilah yang memungkinkan tiga kelas yang sangat berbeda saling menggantikan.

Penyelesaian Java. **Kode 12.15**.

Kode 12.15 `Dictionary.java`: satu metode, didokumentasikan dengan Javadoc, termasuk keadaan kosong dan tidak sah.

```java
// Dictionary.java - Soal 12.5: satu janji yang dibuat setiap kamus.
package predictive;

import java.util.Set;

/**
 * Kamus untuk teks prediktif: ia menjawab "kata apa saja yang dapat dimaksud
 * oleh deretan tekanan tombol ini?". Implementasinya menentukan cara kata disimpan.
 */
public interface Dictionary {

    /**
     * Mengembalikan kata-kata yang cocok dengan sebuah signature papan tombol.
     *
     * @param signature tombol yang ditekan, masing-masing angka 2 sampai 9, misalnya
"4663"
     * @return kata-kata huruf kecil yang cocok, tanpa duplikat dan tanpa urutan
     *         tertentu; himpunan kosong (tidak pernah null) bila tidak ada yang cocok atau
bila
     *         signature memuat karakter yang bukan angka 2 sampai 9.
     *         Implementasi yang mencocokkan prefiks juga mengembalikan
     *         signature.length() huruf pertama dari kata yang lebih panjang.
     */
    Set<String> signatureToWords(String signature);
}
```

Penyelesaian Python. **Kode 12.16** menuliskan kontraknya sebagai *abstract base class* (kelas dasar abstrak, Bab 7): kelas turunan `ABC` yang metodenya ditandai `@abstractmethod`. Python menolak membuat objek dari kelas yang masih memiliki metode abstrak, sehingga setiap implementasi wajib menyediakan `signature_to_words`.

Kode 12.16 `dictionary.py`: kontrak yang sama sebagai kelas dasar abstrak dengan *docstring*.

```python
# dictionary.py - Soal 12.5: satu janji yang dibuat setiap kamus.
from abc import ABC, abstractmethod


class Dictionary(ABC):
    """Kamus untuk teks prediktif: ia menjawab "kata apa saja yang dapat dimaksud
    oleh deretan tekanan tombol ini?". Subkelasnya menentukan cara kata disimpan."""

    @abstractmethod
    def signature_to_words(self, signature):
        """Kembalikan kata-kata yang cocok dengan sebuah signature papan tombol.

        signature: tombol yang ditekan, masing-masing angka 2 sampai 9, mis. "4663".
        Mengembalikan himpunan kata huruf kecil (tidak pernah None): kosong bila tidak
        ada yang cocok atau bila signature memuat karakter yang bukan
        angka 2 sampai 9. Implementasi pencocok prefiks juga mengembalikan
        len(signature) huruf pertama dari kata yang lebih panjang.
        """
```

Baris demi baris. Setiap metode dalam antarmuka Java otomatis bersifat `public` dan `abstract`, sehingga deklarasinya cukup berupa signature metode dan titik koma; `@param` dan `@return` menjadi halaman Javadoc yang sudah kita kenal (Bab 6). Antarmuka dengan tepat satu metode abstrak adalah *functional interface*: lambda seperti `sig -> Set.of("home")` sudah merupakan `Dictionary` yang lengkap, dan Soal 12.10 memanfaatkannya untuk menguji model GUI. Di Python, badan metode abstrak hanya berupa *docstring*-nya, yang merupakan pernyataan yang sah.

Penyelesaian keliru yang menggoda. Antarmuka yang menggoda hanya menyalin wajah publik implementasi pertama:

```
public interface Dictionary {
    ArrayList<WordSig> getDictionary();
    int size();
    Set<String> signatureToWords(String signature);
}
```

Antarmuka ini terkompilasi, dan *list* mengimplementasikannya dengan mudah. Lalu datanglah *map* yang tidak punya `ArrayList<WordSig>` untuk dikembalikan: ia harus membangunnya pada setiap panggilan atau mengembalikan `null`, sehingga merusak setiap pemanggil yang memercayai antarmuka itu. *Representasi* implementasi pertama telah bocor ke dalam kontrak. Rancanglah antarmuka dari kebutuhan pemakainya, bukan dari *field* pengimplementasi pertamanya [7].

> **UJI DIRI.** Seseorang mengusulkan untuk menambahkan `String wordToSignature(String word)` ke dalam `Dictionary`. Apakah itu gagasan yang baik?
>
> *Jawaban.* Tidak. Signature sebuah kata bergantung pada papan tombol, bukan pada cara kamus menyimpan kata, sehingga setiap implementasi akan berisi kode yang sama. Tempatnya adalah satu metode bantu statis bersama, `PredictivePrototype.wordToSignature`, seperti pada **Gambar 12.2**.

12.5.3 Soal 12.6 — Program *list*, dan pengukuran waktu pertama

> **SOAL.** Write the command-line program `Sigs2WordsList` for `DictionaryListImpl`, with the same output as `Sigs2WordsProto`. Then compare the time that `Sigs2WordsList` and `Sigs2WordsProto` take to finish with the same large input. Can the difference be made noticeable? Write down the data you used and the timings. On Linux and macOS, the `time` command reports how long a command takes; use its `real` (elapsed) time.
>
> *Dengan kata lain:* tulislah program baris perintah `Sigs2WordsList` di atas `DictionaryListImpl` dengan keluaran yang sama seperti `Sigs2WordsProto`, lalu bandingkan waktu kedua program untuk masukan besar yang sama. Dapatkah perbedaannya dibuat terasa? Catat data dan hasil pengukurannya; di Linux dan macOS, pakailah waktu `real` dari perintah `time`.

Dengan bahasa kita sendiri. Pasang program baris perintah yang sama di atas *list*, lalu adu dengan prototipe: pertanyaan yang sama, kamus yang sama, pertanyaan yang banyak, kamus yang besar.

Rancangan. Programnya adalah **Kode 12.6** dengan pencarian diganti sebuah `Dictionary` yang dibuat pada baris pertamanya. Untuk adu cepat kita memerlukan kamus besar, dari `MakeDictionary`, dan banyak signature, dari teks sungguhan yang diubah oleh `Words2SigProto`, persis seperti yang dihasilkan orang yang sedang mengetik. Untuk melihat *ke mana* waktunya pergi, sebuah program terpisah mengukur kedua fasenya secara terpisah: membangun kamus sekali, dan menjawab 2.000 pencarian.

Penyelesaian Java. **Kode 12.17** adalah programnya; satu-satunya perbedaannya dari versi Map dan Tree pada Soal 12.9 adalah baris yang ditandai.

Kode 12.17 `Sigs2WordsList.java`: program baris perintah di atas `Dictionary` apa pun; yang ini membuat `DictionaryListImpl`.

```java
// Sigs2WordsList.java - Soal 12.6: Sigs2WordsProto dengan kamus yang disimpan.
package predictive;

import java.io.IOException;
import java.util.TreeSet;

public class Sigs2WordsList {

    public static void main(String[] args) throws IOException {
        Dictionary d = new DictionaryListImpl(PredictivePrototype.DICTIONARY);   // satu-satunya baris yang berbeda
        if (args.length == 0) {
            System.out.println("USAGE: java predictive.Sigs2WordsList sig1 sig2 ... sigN");
            return;
        }
        for (String sig : args) {
            System.out.println(sig + ": " + new TreeSet<>(d.signatureToWords(sig)));
        }
    }
}
```

Dengan `words.txt`, program ini mencetak baris-baris prototipe, uji pertama bagi kamus yang baru:

```
4663: [gone, good, goof, home, hone, hood, hoof]
43556: [hello]
96753: [world]
69: [my, ow, ox]
6263: [mane, name, oboe]
47: [is]
```

Kode 12.18 menulis kamus besar itu. Untuk menghasilkan kata acak yang *sama* di mana pun dan dalam kedua bahasa, program ini memakai *linear congruential generator* (pembangkit kongruensial linear) miliknya sendiri, yang menghitung setiap bilangan dari bilangan sebelumnya:

$$x_{i+1} = (1103515245\, x_i + 12345) \bmod 2^{31} \tag{12.7}$$

dengan benih $x_0 = 12345$ dan konstanta-konstanta dari contoh pembangkit dalam standar bahasa C. Bit *rendah* pembangkit semacam ini terkenal buruk: tiga bit terendahnya berulang dengan periode delapan [8]. Versi pertama program ini memakai `next() % 8` untuk memilih panjang kata dan menghasilkan 199.998 kata sepanjang 6 dari 200.000; perbaikannya, seperti pada `rand()` di C, adalah hanya memakai bit-bit tinggi, `state >> 16`.

Kode 12.18 `MakeDictionary.java`: 200.000 kata rekaan yang dapat dilafalkan, sama di setiap komputer, ditulis ke `big_words.txt`.

```
// MakeDictionary.java - menulis kamus buatan yang besar untuk percobaan pengukuran waktu.
// Benih yang sama menghasilkan berkas yang sama di Java dan di Python.
package predictive;

import java.io.IOException;
import java.io.PrintWriter;
import java.nio.charset.StandardCharsets;

public class MakeDictionary {

    private static final String CONSONANTS = "bcdfghjklmnpqrstvwxyz";
    private static final String VOWELS = "aeiou";
    private static long state = 12345;              // benihnya

    /**
     * Pembangkit kongruensial linear, seperti rand() di C: bilangan berikutnya dalam
     * 0 .. 32767. Hanya bit tinggi yang dipakai; bit rendah terlalu cepat berulang.
     */
    static int next() {
        state = (state * 1103515245L + 12345L) & 0x7fffffffL;
        return (int) (state >> 16);
    }

    /** Kata yang dapat dilafalkan, 3 sampai 10 huruf: konsonan dan vokal berselang-seling.
*/
    static String nextWord() {
        int length = 3 + next() % 8;
        boolean consonant = next() % 2 == 0;
        StringBuilder sb = new StringBuilder(length);
        for (int i = 0; i < length; i++) {
            sb.append(consonant ? CONSONANTS.charAt(next() % 21) : VOWELS.charAt(next() %
5));
            consonant = !consonant;
        }
        return sb.toString();
    }

    public static void main(String[] args) throws IOException {
        int n = (args.length > 0) ? Integer.parseInt(args[0]) : 200_000;
        String file = (args.length > 1) ? args[1] : "big_words.txt";
        String last = "";
        try (PrintWriter out = new PrintWriter(file, StandardCharsets.UTF_8)) {
            for (int i = 0; i < n; i++) {
                last = nextWord();
                out.println(last);
            }
        }
        System.out.println("wrote " + n + " words to " + file + "; the last one is " +
last);
```

Kode 12.18 (Lanjutan) *MakeDictionary.java: 200.000 kata rekaan yang dapat dilafalkan, sama di setiap komputer, ditulis ke big_words.txt.*

```
    }
}
```

```
wrote 200000 words to big_words.txt; the last one is ofad
```

Kode 12.19 mengukur keempat kamus bab ini sekaligus, agar Soal 12.9 dapat memakainya lagi: waktu membangun masing-masing, dan yang terbaik dari tiga eksekusi 2.000 pencarian. Prototipe, yang tidak membangun apa pun, hanya diberi 5 pencarian; 2.000 akan memakan waktu bermenit-menit.

Kode 12.19 `TimingComparison.java`: waktu pembangunan dan waktu pencarian keempat kamus pada berkas yang sama dan 2.000 signature yang sama.

```
// TimingComparison.java - Soal 12.6 dan 12.9: keempat kamus diukur
// pada berkas kamus yang sama dan signature yang sama.
package predictive;

import java.nio.charset.StandardCharsets;
import java.nio.file.Files;
import java.nio.file.Path;
import java.util.ArrayList;
import java.util.List;
import java.util.Set;
import java.util.function.Function;

public class TimingComparison {

    interface Builder {
        Dictionary build(String path) throws Exception;
    }

    /** Yang terbaik dari tiga eksekusi semua pencarian, dalam milidetik; sekaligus
menghitung hasilnya. */
    static double timeLookups(Function<String, Set<String>> lookup, List<String> sigs,
long[] found) {
        double best = Double.MAX_VALUE;
        for (int run = 0; run < 3; run++) {
            long count = 0;
            long t0 = System.nanoTime();
            for (String s : sigs) {
                count += lookup.apply(s).size();
            }
            best = Math.min(best, (System.nanoTime() - t0) / 1e6);
            found[0] = count;
        }
        return best;
    }

    static void row(String name, String buildMs, int queries, double lookupMs, long found)
{
        System.out.printf("%-6s %10s %8d %12.2f %12.2f %8d%n",
                name, buildMs, queries, lookupMs, 1000.0 * lookupMs / queries, found);
    }
```

Kode 12.19 (Lanjutan) *TimingComparison.java: waktu pembangunan dan waktu pencarian keempat kamus pada berkas yang sama dan 2.000 signature yang sama.*

```
    public static void main(String[] args) throws Exception {
        String path = PredictivePrototype.DICTIONARY;
        List<String> words = new ArrayList<>();
        for (String line : Files.readAllLines(Path.of(path), StandardCharsets.UTF_8)) {
            String w = line.trim().toLowerCase();
            if (PredictivePrototype.isValidWord(w)) {
                words.add(w);
            }
        }
        List<String> sigs = new ArrayList<>();          // 2000 signature dari entri yang
benar-benar ada
        for (int k = 0; k < 2000; k++) {
            sigs.add(PredictivePrototype.wordToSignature(words.get((int) ((k * 7919L) %
words.size()))));
        }
        System.out.println("dictionary " + path + ": " + words.size() + " words");
        System.out.println("impl     build ms  queries  lookups ms  us/lookup   found");

        long[] found = new long[1];
        List<String> five = sigs.subList(0, 5);          // prototipe terlalu lambat untuk
2000
        double ms = timeLookups(PredictivePrototype::signatureToWords, five, found);
        row("proto", "-", five.size(), ms, found[0]);

        String[] names = {"list", "map", "tree"};
        Builder[] builders = {DictionaryListImpl::new, DictionaryMapImpl::new,
DictionaryTreeImpl::new};
        for (int i = 0; i < names.length; i++) {
            long t0 = System.nanoTime();
            Dictionary d = builders[i].build(path);
            double buildMs = (System.nanoTime() - t0) / 1e6;
            ms = timeLookups(d::signatureToWords, sigs, found);
            row(names[i], String.format("%.1f", buildMs), sigs.size(), ms, found[0]);
        }
    }
}
```

Jalankan dengan kamus besar (`PREDICTIVE_DICT=big_words.txt java predictive.TimingComparison`):

```
dictionary big_words.txt: 200000 words
impl     build ms  queries  lookups ms  us/lookup   found
proto           -        5      311.42    62283.18      19
list        419.6     2000        5.41        2.70    5332
map         307.0     2000        0.45        0.22    5332
tree        956.1     2000       27.14       13.57    7496
```

Kolom `us/lookup` adalah mikrodetik per pencarian. Prototipe memerlukan 62.283, kira-kira seperenam belas detik; *list* memerlukan 2,70, lebih dari dua puluh ribu kali lebih sedikit. *List* membayar sekali, 420 milidetik untuk membaca dan mengurutkan 200.000 kartu, harga kira-kira tujuh pencarian prototipe, dan sesudah itu selalu unggul. Kolom `found` (5.332 kata untuk *list* dan *map*) adalah satu pemeriksaan lagi bahwa keduanya memberikan jawaban yang sama.

Penyelesaian Python. **Kode 12.20**, **Kode 12.21**, dan **Kode 12.22** adalah ketiga program yang sama; `make_dictionary.py` menulis berkas yang identik dengan berkas Java, bita demi bita, karena keduanya mengikuti Persamaan (12.7) dengan benih yang sama.

Kode 12.20 `sigs2words_list.py`: program *list* dalam Python.

```python
# sigs2words_list.py - Soal 12.6: sigs2words_proto dengan kamus yang disimpan.
import sys

from predictive.dictionary_list_impl import DictionaryListImpl
from predictive.prototype import DICTIONARY


def main(args):
    d = DictionaryListImpl(DICTIONARY)                  # satu-satunya baris yang berbeda
    if not args:
        print("USAGE: python -m predictive.sigs2words_list sig1 sig2 ... sigN")
        return
    for sig in args:
        print(sig + ": [" + ", ".join(sorted(d.signature_to_words(sig))) + "]")


if __name__ == "__main__":
    main(sys.argv[1:])
```

```
4663: [gone, good, goof, home, hone, hood, hoof]
43556: [hello]
96753: [world]
69: [my, ow, ox]
6263: [mane, name, oboe]
47: [is]
```

Kode 12.21 `make_dictionary.py`: kamus rekaan yang sama dalam Python.

```python
# make_dictionary.py - menulis kamus buatan yang besar untuk percobaan pengukuran waktu.
# Benih yang sama menghasilkan berkas yang sama di Java dan di Python.
import sys

CONSONANTS = "bcdfghjklmnpqrstvwxyz"
VOWELS = "aeiou"
_state = 12345                                  # benihnya


def next_number():
    """Pembangkit kongruensial linear, seperti rand() di C: bilangan berikutnya dalam
    0 .. 32767. Hanya bit tinggi yang dipakai; bit rendah terlalu cepat berulang."""
    global _state
    _state = (_state * 1103515245 + 12345) & 0x7FFFFFFF
    return _state >> 16


def next_word():
    """Kata yang dapat dilafalkan, 3 sampai 10 huruf: konsonan dan vokal berselang-
seling."""
    length = 3 + next_number() % 8
    consonant = next_number() % 2 == 0
    letters = []
    for _ in range(length):
        letters.append(CONSONANTS[next_number() % 21] if consonant else
VOWELS[next_number() % 5])
        consonant = not consonant
    return "".join(letters)


def main(args):
    n = int(args[0]) if len(args) > 0 else 200_000
    file = args[1] if len(args) > 1 else "big_words.txt"
    last = ""
    with open(file, "w", encoding="utf-8", newline="\n") as out:
        for _ in range(n):
            last = next_word()
            out.write(last + "\n")
    print(f"wrote {n} words to {file}; the last one is {last}")


if __name__ == "__main__":
    main(sys.argv[1:])
```

```
wrote 200000 words to big_words.txt; the last one is ofad
```

Kode 12.22 `timing_comparison.py`: keempat kamus diukur dalam Python.

```
# timing_comparison.py - Soal 12.6 dan 12.9: keempat kamus diukur
# pada berkas kamus yang sama dan signature yang sama.
import time

from predictive.dictionary_list_impl import DictionaryListImpl
from predictive.dictionary_map_impl import DictionaryMapImpl
from predictive.dictionary_tree_impl import DictionaryTreeImpl
from predictive.prototype import DICTIONARY, is_valid_word, signature_to_words,
word_to_signature


def time_lookups(lookup, sigs):
    """Yang terbaik dari tiga eksekusi semua pencarian, dalam milidetik; sekaligus
menghitung hasilnya."""
    best, found = float("inf"), 0
    for _ in range(3):
        t0 = time.perf_counter()
        found = sum(len(lookup(s)) for s in sigs)
        best = min(best, (time.perf_counter() - t0) * 1e3)
    return best, found


def row(name, build_ms, queries, lookup_ms, found):
    print(f"{name:<6} {build_ms:>10} {queries:8d} {lookup_ms:12.2f} "
          f"{1000.0 * lookup_ms / queries:12.2f} {found:8d}")


def main():
    with open(DICTIONARY, encoding="utf-8") as f:
        words = [w for w in (line.strip().lower() for line in f) if is_valid_word(w)]
    sigs = [word_to_signature(words[(k * 7919) % len(words)]) for k in range(2000)]
    print(f"dictionary {DICTIONARY}: {len(words)} words")
    print("impl     build ms  queries  lookups ms  us/lookup   found")

    five = sigs[:5]                              # prototipe terlalu lambat untuk 2000
    ms, found = time_lookups(signature_to_words, five)
    row("proto", "-", len(five), ms, found)

    for name, cls in (("list", DictionaryListImpl), ("map", DictionaryMapImpl),
                      ("tree", DictionaryTreeImpl)):
        t0 = time.perf_counter()
        d = cls(DICTIONARY)
        build_ms = f"{(time.perf_counter() - t0) * 1e3:.1f}"
        ms, found = time_lookups(d.signature_to_words, sigs)
        row(name, build_ms, len(sigs), ms, found)
```

Kode 12.22 (Lanjutan) *timing_comparison.py: keempat kamus diukur dalam Python.*

```
if __name__ == "__main__":
    main()
```

```
dictionary big_words.txt: 200000 words
impl     build ms  queries  lookups ms  us/lookup   found
proto           -        5     1177.68    235536.63      19
list       1077.7     2000       12.19         6.10    5332
map         373.5     2000        0.72         0.36    5332
tree       1874.9     2000       21.93        10.97    7496
```

Angka Python lebih besar, seperti yang wajar bagi sebuah interpreter, tetapi rasionya bercerita sama: 235.537 lawan 6,10 mikrodetik, hampir empat puluh ribu kali.

Adu cepat di baris perintah. Soal meminta perintah `time`, yang mengukur seluruh program, termasuk waktu mulainya. Kita mengubah 53 kata dari `sample_text.txt` (ada di folder bab ini) menjadi signature lalu memberikan semuanya kepada setiap program:

```
export PREDICTIVE_DICT=big_words.txt
SIGS=$(java predictive.Words2SigProto $(cat sample_text.txt) | sed -n 's/^output: //p')
time java predictive.Sigs2WordsProto $SIGS > /dev/null
time java predictive.Sigs2WordsList  $SIGS > /dev/null
```

Baris kedua hanya menyimpan angka-angka sesudah `output:`, dan `> /dev/null` membuang jawabannya, karena yang penting hanya waktunya. **Tabel 12.5** mengumpulkan waktu `real` untuk keempat program bab ini. (Di Windows, `Measure-Command { ... }` pada PowerShell melakukan hal yang sama; lihat `README.md` di folder itu.)

Tabel 12.5 Waktu berlalu setiap program baris perintah untuk 53 signature dari `sample_text.txt` dengan kamus 200.000 kata, sebagaimana dilaporkan `time`. Setiap eksekusi mencakup memulai program dan membangun kamusnya.

Program	Java, `real`	Python, `real`
`Sigs2WordsProto` / `sigs2words_proto`	4.073 s	15.092 s
`Sigs2WordsList` / `sigs2words_list`	1.022 s	1.031 s
`Sigs2WordsMap` / `sigs2words_map`	0.905 s	0.432 s
`Sigs2WordsTree` / `sigs2words_tree`	2.581 s	1.991 s

Jadi perbedaannya terasa: empat detik lawan satu di Java, lima belas lawan satu di Python. Namun satu detik milik *list* hampir seluruhnya untuk memulai dan membangun; 53 pencariannya sendiri hanya sekitar 150 mikrodetik. `time` tidak dapat memisahkan kedua fase itu, dan karena itulah **Kode 12.19** mengukurnya terpisah. Ringkasan yang jujur: *untuk segelintir pertanyaan, pembangunan mendominasi dan prototipe hanya beberapa kali lebih lambat; untuk ribuan pertanyaan, pencarian mendominasi dan prototipe puluhan ribu kali lebih lambat.*

Baris demi baris. Pada **Kode 12.18**, `state` bertipe `long` karena $1103515245 \cdot x_i$ meluap dari `int`, dan `& 0x7fffffffL` menyimpan 31 bit terendah, yaitu sisa bagi modulo 2^{31}. `nextWord` menyelang-nyelingkan konsonan dan vokal, sehingga katanya dapat dilafalkan dan signature-nya bertabrakan seperti kata sungguhan. Pada **Kode 12.19**, antarmuka bersarang `Builder` (“membangun `Dictionary` dari sebuah *path*, mungkin melempar eksepsi”) dipenuhi oleh referensi konstruktor `DictionaryListImpl::new` dan kedua saudaranya, sehingga satu gelung membangun dan mengukur ketiganya. `timeLookups` menerima pencarian sebagai `Function<String, Set<String>>`, di sini `d::signatureToWords`, dan menyimpan yang terbaik dari tiga eksekusi untuk meredam pengaruh kompilasi *just-in-time*. `(k * 7919) % words.size()` menyebar 2.000 pilihan ke seluruh kamus; 7919 adalah bilangan prima, sehingga pilihannya tidak cepat berulang.

Penyelesaian keliru yang menggoda. Percobaan yang menggoda adalah contoh dalam pernyataan soal yang dijalankan apa adanya: `time java predictive.Sigs2WordsList 4663 43556 96753 69 6263 47` dengan `words.txt`, dilawankan dengan prototipe. Pada mesin yang dipakai untuk buku ini, keduanya melaporkan `real 0m0.162s` dan `real 0m0.163s`, dan orang pun menyimpulkan bahwa *list* tidak lebih cepat. Yang terukur adalah waktu mulai mesin virtual Java; sembilan puluh tiga kata dan enam pertanyaan tidak dapat membedakan kedua struktur. Pengukuran baru bermakna bila besaran yang membedakan para kandidat mendominasi segala hal lainnya.

> **UJI DIRI.** Dengan angka-angka pada tabel waktu Java, setelah berapa pencarian pembangunan *list* terbayar dibandingkan prototipe, dan setelah berapa pencarian dibandingkan *map*?
>
> *Jawaban.* Dibandingkan prototipe: pembangunan memakan 420 ms dan setiap pencarian prototipe kira-kira 62 ms, jadi setelah $420/62 \approx 7$ pencarian. Dibandingkan *map*, tidak pernah terbayar: *map* dibangun lebih cepat (307 ms) *dan* menjawab lebih cepat (0,22 µs lawan 2,70 µs). Keunggulan *list* atas *map* terletak di tempat lain: *list* dapat mendaftar signature secara urut, sedangkan *hash map* tidak.

12.6 Bagian Tiga: *Map* dan Pohon

List terurut menjawab dalam segelintir perbandingan, tetapi memiliki dua kelemahan: menyisipkan sebuah kata menggeser rata-rata separuh kartu, pekerjaan $O(n)$, dan ia hanya menjawab signature yang *utuh*, padahal telepon prediktif seharusnya menyarankan kata selagi kata itu diketik. Bagian Tiga menghilangkan keduanya.

12.6.1 Soal 12.7 — *Map* bernilai jamak

> **SOAL.** Write a class `DictionaryMapImpl` that implements `Dictionary` and stores the dictionary in a generic *multi-valued map*: a map from each signature to the collection of its words. Like the list, it must find the words of a signature quickly, but without scanning to either side of an index; unlike the list, it must also allow new words to be inserted efficiently. Choose a `Map` implementation from the Java Collections API, explain how it works, and justify the choice. The constructor fills the map; `signatureToWords` returns, as a `Set<String>`, only the whole words whose length equals the signature's. Do not use `WordSig`.
>
> *Dengan kata lain:* tulislah `DictionaryMapImpl` yang menyimpan kamus dalam *map* generik bernilai jamak, dari setiap signature ke koleksi kata-katanya. Pencariannya harus cepat tanpa memindai ke kedua sisi indeks, dan penyisipan kata baru juga harus efisien. Pilih implementasi `Map` dari Java Collections API, jelaskan cara kerjanya, dan beri alasannya. `signatureToWords` hanya mengembalikan kata utuh yang panjangnya sama dengan signature; jangan memakai `WordSig`.

Dengan bahasa kita sendiri. Alih-alih sederet kartu, simpanlah sebuah lemari berlaci, setiap laci berlabel sebuah signature dan berisi kata-katanya. Untuk menjawab, buka laci berlabel yang tepat; untuk menambah kata, temukan lacinya (buat bila belum ada) lalu masukkan kata itu.

Rancangan. Tipenya sudah berbicara: `Map<String, Set<String>>`, dari signature ke himpunan kata. Nilainya berupa *himpunan* karena satu signature memiliki beberapa kata dan sebuah kata tidak boleh muncul dua kali dalam satu laci. `Map` yang mana? Java menawarkan dua implementasi utama [5]:

- `HashMap` menghitung sebuah bilangan dari kunci, yaitu *hash code*-nya, yang memilih sebuah *bucket* (ember) dalam larik internal; pencarian pergi ke ember itu dan membandingkan beberapa kunci yang ada di sana. Rata-rata biayanya sama berapa pun ukuran *map*, $O(1)$ *harapan* [6], asalkan kode *hash*-nya tersebar baik, seperti kode *hash* `String`. Kunci keluar tanpa urutan yang berguna.
- `TreeMap` menyimpan kuncinya dalam pohon pencarian seimbang yang diurutkan oleh `compareTo`: $O(\log n)$ per pencarian atau penyisipan, dan kunci keluar secara urut, sehingga juga dapat menjawab pertanyaan seperti "semua kunci di antara dua nilai ini".

Kita hanya pernah bertanya "apa isi laci berlabel *s*?" dan "masukkan kata ini ke lacinya", dan tidak satu pun memerlukan urutan, jadi pilihannya `HashMap`. (Latihan 12.3 memperlihatkan apa yang

akan diberikan `TreeMap`.) Dengan memperhitungkan penghitungan *hash* atas $\bar{m}$ karakter signature,

$$T_{\text{map}}(n) = O(\bar{m} + k) \text{ harapan per pencarian}, \qquad T_{\text{insert}} = O(\bar{m}) \text{ harapan}, \tag{12.8}$$

dan tidak satu pun bergantung pada n. Laci yang tidak ada menghasilkan himpunan kosong, sebagaimana dituntut kontrak pada **Kode 12.15**, dan laci itu sendiri tidak pernah diserahkan, hanya *view* hanya-baca atasnya; penyelesaian keliru memperlihatkan alasannya.

Penyelesaian Java. **Kode 12.23** adalah kamusnya; **Kode 12.24** mengujinya.

Kode 12.23 `DictionaryMapImpl.java`: `HashMap` dari signature ke himpunan kata, dengan `addWord` publik untuk penyisipan cepat dan hasil yang hanya-baca.

```
// DictionaryMapImpl.java - Soal 12.7: kamus sebagai map bernilai jamak,
// signature -> himpunan kata, disimpan dalam HashMap.
package predictive;

import java.io.File;
import java.io.IOException;
import java.nio.charset.StandardCharsets;
import java.util.Collections;
import java.util.HashMap;
import java.util.HashSet;
import java.util.Map;
import java.util.Scanner;
import java.util.Set;

public final class DictionaryMapImpl implements Dictionary {

    // HashMap: pencarian atau penyisipan memakan waktu harapan O(1), berapa pun
    // banyaknya signature; kita tidak pernah memerlukan signature dalam urutan.
    private final Map<String, Set<String>> dictionary = new HashMap<>();

    public DictionaryMapImpl(String path) throws IOException {
        try (Scanner in = new Scanner(new File(path), StandardCharsets.UTF_8)) {
            while (in.hasNextLine()) {
                String word = in.nextLine().trim().toLowerCase();
                if (PredictivePrototype.isValidWord(word)) {
                    addWord(word);
                }
            }
        }
    }

    /** Menyisipkan satu kata: temukan (atau buat) himpunan signature-nya, lalu tambahkan.
*/
    public void addWord(String word) {
        String signature = PredictivePrototype.wordToSignature(word);
        dictionary.computeIfAbsent(signature, s -> new HashSet<>()).add(word);
    }

    @Override
    public Set<String> signatureToWords(String signature) {
        Set<String> words = dictionary.get(signature);
        if (words == null) {
            return Collections.emptySet();
        }
        return Collections.unmodifiableSet(words);   // view hanya-baca, tanpa menyalin
    }
```

Kode 12.23 (Lanjutan) *DictionaryMapImpl.java: HashMap dari signature ke himpunan kata, dengan addWord publik untuk penyisipan cepat dan hasil yang hanya-baca.*

```
    /** Banyaknya signature yang berbeda (kunci map). */
    public int signatureCount() {
        return dictionary.size();
    }
}
```

Kode 12.24 `MapCheck.java`: pemeriksaan Soal 12.7, termasuk upaya mengosongkan himpunan yang dikembalikan, dan kelas *map* dengan dua metode keliru.

```
// MapCheck.java - pemeriksaan Soal 12.7, dengan dua penyelesaian keliru yang menggoda:
// put() yang menimpa, dan getter yang menyerahkan himpunan milik map sendiri.
package predictive;

import java.util.HashMap;
import java.util.HashSet;
import java.util.Map;
import java.util.Set;
import java.util.TreeSet;

public class MapCheck {

    /** Menggoda tetapi salah, dua kali. */
    static final class LeakyMap {
        private final Map<String, Set<String>> map = new HashMap<>();

        void addWordOverwriting(String word) {        // salah 1: himpunan baru setiap kali
            Set<String> s = new HashSet<>();
            s.add(word);
            map.put(PredictivePrototype.wordToSignature(word), s);
        }

        void addWord(String word) {
            map.computeIfAbsent(PredictivePrototype.wordToSignature(word), k -> new
HashSet<>()).add(word);
        }

        Set<String> signatureToWords(String sig) {    // salah 2: himpunan milik map
sendiri
            return map.getOrDefault(sig, new HashSet<>());
        }
    }

    private static int passed = 0;

    private static void check(String name, Object expected, Object actual) {
        if (!java.util.Objects.equals(expected, actual)) {
            throw new AssertionError(name + ": expected " + expected + " but got " +
actual);
        }
        passed++;
        System.out.println("PASS " + name);
    }
```

Kode 12.24 (Lanjutan) *MapCheck.java: pemeriksaan Soal 12.7, termasuk upaya mengosongkan himpunan yang dikembalikan, dan kelas map dengan dua metode keliru.*

```
    public static void main(String[] args) throws Exception {
        DictionaryMapImpl d = new DictionaryMapImpl("words.txt");
        check("signatures", 61, d.signatureCount());
        check("4663", "[gone, good, goof, home, hone, hood, hoof]",
                new TreeSet<>(d.signatureToWords("4663")).toString());
        check("absent", "[]", d.signatureToWords("22222").toString());
        check("whole words only", "[work]", d.signatureToWords("9675").toString());
        d.addWord("hoop");
        check("added hoop", true, d.signatureToWords("4667").contains("hoop"));
        try {
            d.signatureToWords("4663").clear();
            System.out.println("clear() was allowed");
        } catch (UnsupportedOperationException e) {
            System.out.println("clear() refused: UnsupportedOperationException");
        }
        check("still 7 words", 7, d.signatureToWords("4663").size());

        LeakyMap leaky = new LeakyMap();
        for (String w : new String[] {"gone", "good", "home"}) {
            leaky.addWordOverwriting(w);
        }
        System.out.println("overwriting put  : " + leaky.signatureToWords("4663"));
        LeakyMap leaky2 = new LeakyMap();
        for (String w : new String[] {"gone", "good", "home"}) {
            leaky2.addWord(w);
        }
        leaky2.signatureToWords("4663").clear();       // pemanggil yang "sekadar
merapikan"
        System.out.println("after caller clear: " + leaky2.signatureToWords("4663"));
        System.out.println("All " + passed + " checks passed.");
    }
}
```

```
PASS signatures
PASS 4663
PASS absent
PASS whole words only
PASS added hoop
clear() refused: UnsupportedOperationException
PASS still 7 words
overwriting put  : [home]
after caller clear: []
All 6 checks passed.
```

Ke-93 baris sah menghasilkan 61 laci. Pemeriksaan `whole words only` memperlihatkan bahwa *map*, seperti *list*, hanya menjawab signature yang lengkap (`work` untuk 9675, tanpa sepotong pun dari `world`), dan `clear() refused` memperlihatkan *view* hanya-baca sedang bekerja.

Penyelesaian Python. `dict` di Python adalah *hash map*. `setdefault(key, default)` mengembalikan nilai di bawah `key`, setelah lebih dulu menyimpan `default` bila kuncinya belum ada: "temukan atau buat lacinya". Hasilnya dikembalikan sebagai `frozenset`, salinan yang imutabel; **Kode 12.26** memuat ujinya.

Kode 12.25 `dictionary_map_impl.py`: *map* bernilai jamak dalam Python, mengembalikan `frozenset`.

```python
# dictionary_map_impl.py - Soal 12.7: kamus sebagai map bernilai jamak,
# signature -> himpunan kata, disimpan dalam dict (hash map Python).
from predictive.dictionary import Dictionary
from predictive.prototype import is_valid_word, word_to_signature


class DictionaryMapImpl(Dictionary):

    def __init__(self, path):
        # dict: pencarian atau penyisipan memakan waktu harapan O(1)
        self._dictionary = {}
        with open(path, encoding="utf-8") as f:
            for line in f:
                word = line.strip().lower()
                if is_valid_word(word):
                    self.add_word(word)

    def add_word(self, word):
        """Sisipkan satu kata: temukan (atau buat) himpunan signature-nya, lalu
tambahkan."""
        self._dictionary.setdefault(word_to_signature(word), set()).add(word)

    def signature_to_words(self, signature):
        # frozenset adalah salinan hanya-baca: pemanggil tidak dapat mengubah kamus
        return frozenset(self._dictionary.get(signature, ()))

    def signature_count(self):
        """Banyaknya signature yang berbeda (kunci dict)."""
        return len(self._dictionary)
```

Kode 12.26 `map_check.py`: uji Soal 12.7 dalam Python, dan dua metode keliru yang sama.

```
# map_check.py - uji Soal 12.7, dengan dua penyelesaian keliru yang menggoda:
# penugasan yang menimpa, dan getter yang menyerahkan himpunan milik dict sendiri.
import sys
import unittest

from predictive.dictionary_map_impl import DictionaryMapImpl
from predictive.prototype import word_to_signature


class LeakyMap:
    """Menggoda tetapi salah, dua kali."""

    def __init__(self):
        self.map = {}

    def add_word_overwriting(self, word):           # salah 1: himpunan baru setiap kali
        self.map[word_to_signature(word)] = {word}

    def add_word(self, word):
        self.map.setdefault(word_to_signature(word), set()).add(word)

    def signature_to_words(self, sig):              # salah 2: himpunan milik dict sendiri
        return self.map.get(sig, set())


class TestMapImpl(unittest.TestCase):
    def setUp(self):
        self.d = DictionaryMapImpl("words.txt")

    def test_counts_and_lookups(self):
        self.assertEqual(61, self.d.signature_count())
        self.assertEqual(["gone", "good", "goof", "home", "hone", "hood", "hoof"],
                         sorted(self.d.signature_to_words("4663")))
        self.assertEqual(frozenset(), self.d.signature_to_words("22222"))
        self.assertEqual({"work"}, self.d.signature_to_words("9675"))

    def test_add_word(self):
        self.d.add_word("hoop")
        self.assertIn("hoop", self.d.signature_to_words("4667"))

    def test_result_is_read_only(self):
        with self.assertRaises(AttributeError):     # frozenset tidak memiliki clear()
            self.d.signature_to_words("4663").clear()
        self.assertEqual(7, len(self.d.signature_to_words("4663")))
```

Kode 12.26 (Lanjutan) *map_check.py: uji Soal 12.7 dalam Python, dan dua metode keliru yang sama.*

```
if __name__ == "__main__":
    leaky = LeakyMap()
    for w in ("gone", "good", "home"):
        leaky.add_word_overwriting(w)
    print("overwriting =     :", leaky.signature_to_words("4663"))
    leaky2 = LeakyMap()
    for w in ("gone", "good", "home"):
        leaky2.add_word(w)
    leaky2.signature_to_words("4663").clear()       # pemanggil yang "sekadar merapikan"
    print("after caller clear:", leaky2.signature_to_words("4663"))
    sys.stdout.flush()
    unittest.main(verbosity=2)
```

```
overwriting =     : {'home'}
after caller clear: set()
test_add_word (__main__.TestMapImpl.test_add_word) ... ok
test_counts_and_lookups (__main__.TestMapImpl.test_counts_and_lookups) ... ok
test_result_is_read_only (__main__.TestMapImpl.test_result_is_read_only) ... ok

----------------------------------------------------------------------
Ran 3 tests in 0.001s

OK
```

Baris demi baris. Pada **Kode 12.23**, kelasnya `final` karena konstruktornya memanggil metode publik `addWord`: dalam sebuah subkelas, `addWord` yang di-*override* dapat berjalan sebelum *field* subkelas itu ada [7]. `computeIfAbsent(signature, s -> new HashSet<>())` mengembalikan lacinya, setelah lebih dulu membuat `HashSet` kosong bila belum ada, dan `.add(word)` memasukkan katanya. `get` mengembalikan `null` untuk kunci yang tidak ada, yang kita ubah menjadi `Collections.emptySet()` yang imutabel dan dipakai bersama. `Collections.unmodifiableSet(words)` membungkus laci dalam sebuah *view* yang membaca langsung dari himpunan aslinya dan melempar `UnsupportedOperationException` dari setiap metode yang mengubah; membuatnya berbiaya $O(1)$. Pustaka Python tidak memiliki *view* semacam itu untuk himpunan, sehingga **Kode 12.25** menyalin k kata itu ke sebuah `frozenset`, tetap murah.

Penyelesaian keliru yang menggoda. `LeakyMap` di kedua program pemeriksa memuat dua kesalahan yang diundang soal ini. Yang pertama terletak pada penyisipan. Metode `addWordOverwriting` pada hakikatnya menjalankan

```
map.put(signature, new HashSet<>(Set.of(word)));
```

untuk setiap kata, yang *mengganti* lacinya, sehingga hanya kata terakhir dari setiap signature yang bertahan: setelah gone, good, dan home, laci 4663 hanya berisi `[home]`. Yang kedua terletak pada pencarian: mengembalikan himpunan milik *map* sendiri sama dengan menyerahkan kunci lemari kepada setiap pemanggil. Pemanggil yang "sekadar merapikan" dengan memanggil `clear()` pada hasilnya mengosongkan laci *di dalam kamus*, dan pencarian 4663 berikutnya tidak menemukan apa pun, padahal tidak ada metode kamus yang dipanggil di antaranya. Seperti pada soal enkapsulasi di Bab 8: jangan pernah menyerahkan objek internal yang dapat berubah; serahkan *view* atau salinan.

> **UJI DIRI.** `signatureToWords` kita mengembalikan *view* yang tidak dapat diubah di Java dan *salinan* di Python. Sebutkan satu hal yang dapat diamati pemanggil yang berbeda di antara keduanya.
>
> *Jawaban. View* membaca langsung dari laci aslinya, sehingga bila seseorang memanggil `addWord` setelah pemanggil menerima *view* itu, *view* tersebut ikut memperlihatkan kata baru. Salinan adalah potret sesaat dan tidak berubah. (Keduanya menolak diubah oleh pemanggil: *view* melempar `UnsupportedOperationException`, dan `frozenset` sama sekali tidak memiliki `clear` atau `add`.)

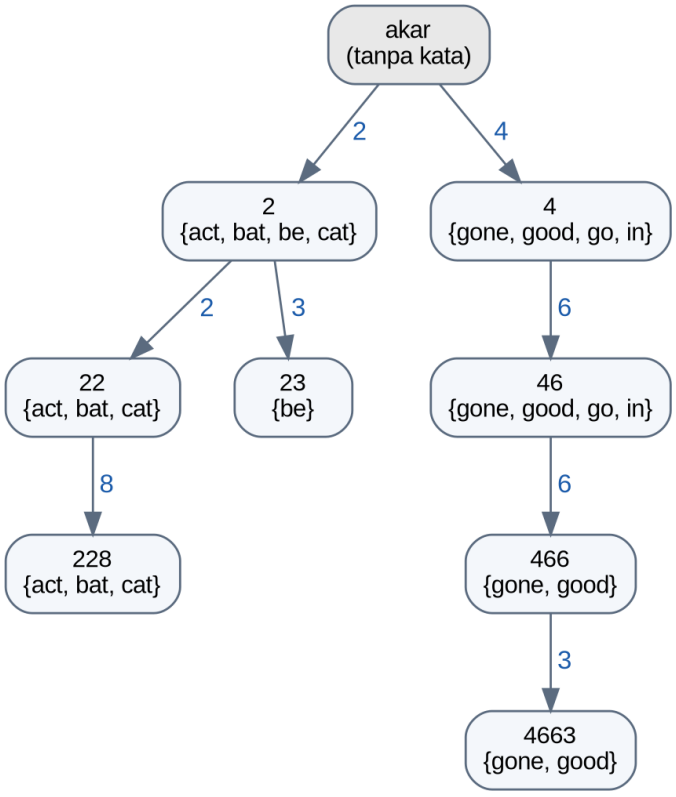
12.6.2 Soal 12.8 — Pohon tombol yang juga mencocokkan prefiks

> **SOAL.** Write a class `DictionaryTreeImpl` that implements `Dictionary` and stores the dictionary in a tree of your own. Searching must be fast (like the list) and inserting must be fast (like the map). In addition, the tree must find words when only the first part of the signature, a *prefix*, is known, so that users can see the word they intend while they are still typing it. Each node has up to eight branches, one for each key 2 to 9; the path from the root to a node spells a signature or the beginning of one; each node stores the collection of words

> whose signature begins with that path. The root stores no words. The constructor takes the path of the dictionary file. `signatureToWords` returns, as a `Set<String>`, the matching words and prefixes of words, each cut to the length of the signature. To save memory, store whole words as read from the file, and write a helper that trims all the words of a collection.
>
> *Dengan kata lain:* tulislah `DictionaryTreeImpl`, kamus dalam pohon buatan sendiri yang cepat dicari dan cepat disisipi, dan yang juga menemukan kata ketika baru bagian awal signature-nya, *prefiks*, yang diketahui. Setiap simpul memiliki paling banyak delapan cabang, satu untuk setiap tombol 2 sampai 9; lintasan dari akar ke sebuah simpul mengeja (awal) sebuah signature, dan setiap simpul menyimpan kata-kata yang signature-nya diawali lintasan itu. Akar tidak menyimpan kata. `signatureToWords` mengembalikan kata dan prefiks kata yang cocok, masing-masing dipotong sepanjang signature; simpanlah kata utuh dan tulis metode bantu untuk memotongnya.

Dengan bahasa kita sendiri. Bayangkan jalan-jalan. Dari akar, delapan jalan berangkat, berlabel 2 sampai 9; masing-masing menuju persimpangan tempat paling banyak delapan jalan berangkat lagi. Kata `good` disimpan dengan menyusuri 4, 6, 6, 3 dan meninggalkan salinannya di setiap persimpangan yang dilalui: `4`, `46`, `466`, dan `4663`. Untuk menjawab "4663", susurilah jalan-jalan yang sama dan lihat apa yang ditinggalkan di persimpangan terakhir. Karena kata ditinggalkan di *setiap* persimpangan, persimpangan `466` mengenal `good`, `gone`, dan `home` walaupun signature mereka lebih panjang: begitulah prefiks dicocokkan. Setelah dipotong sepanjang pertanyaan, `good` dan `goof` sama-sama menjadi `goo`, dan himpunan hanya menyimpan satu.

Rancangan. **Gambar 12.3** menggambar pohon untuk delapan kata: act, bat, be, cat, go, gone, good, dan in. Label pada sisi adalah tombol; setiap kotak memperlihatkan lintasan yang menuju ke sana dan kata-kata yang tersimpan di sana.



Gambar 12.3 Kamus pohon untuk delapan kata. Setiap kata disimpan di setiap simpul pada lintasan signature-nya, sehingga simpul `46` memuat keempat kata yang diawali tombol 4 dan 6.

Bacalah tiga pertanyaan dari gambar itu: `"228"` menemukan `{act, bat, cat}`; `"46"` menemukan `{gone, good, go, in}`, yang setelah dipotong menjadi dua huruf menjadi `{go, in}`; dan `"229"` tidak menemukan jalan 9 sesudah 22, sehingga jawabannya kosong.

Pohon semacam ini, yang lintasannya mengeja kunci, disebut *trie* (dari kata re*trie*val, pengambilan kembali) [9]. Strukturnya rekursif seperti pohon biner di Bab 8, dengan simpul berupa himpunan kata ditambah delapan anak yang berupa `null` atau simpul, dan metodenya mengikuti *resep delegasi* dari Subbab 8.2, seperti diperlihatkan **Tabel 12.6**.

Tabel 12.6 Resep delegasi dari Bab 8 yang diterapkan pada dua metode rekursif kamus pohon.

Pertanyaan	`insert(signature, pos, word)`	`find(signature, pos)`
Kasus dasar	`pos == length`: tidak ada lagi yang harus dikerjakan	`pos == length`: jawabannya simpul ini
Tugas saya sendiri	buat anak untuk tombol `signature[pos]` bila belum ada; tambahkan `word` ke anak itu	cari anak untuk tombol `signature[pos]`; `null` bila tidak ada atau bukan tombol
Delegasi	`child.insert(signature, pos + 1, word)`	`child.find(signature, pos + 1)`

Tombol `'2'` menuju anak 0 dan `'9'` menuju anak 7. Menyisipkan kata sepanjang m huruf mengunjungi m simpul, dan menemukan simpul sebuah signature sepanjang ℓ mengunjungi ℓ simpul, berapa pun ukuran kamusnya. Namun sesudah itu jawabannya dibangun: setiap kata yang tersimpan di simpul itu dipotong dan dimasukkan ke hasil, dan simpul-simpul di dekat akar menyimpan banyak kata. Dengan W_s kata di simpul milik s,

$$T_{\text{tree}}(s) = O(\ell + W_s\,\ell) \text{ per pencarian}, \qquad \text{memori} = O\left(\sum_{w} |w|\right) \text{ referensi kata}, \tag{12.9}$$

karena setiap kata w disimpan di $|w|$ simpul pada lintasannya. Ke-90 kata kita yang berbeda memiliki 354 huruf, jadi 354 referensi di 120 simpul; 165.929 kata rekaan yang berbeda memerlukan 1.183.435 referensi. String-nya sendiri tidak disalin: setiap simpul merujuk objek `String` yang sama, dan itulah penghematan memori yang diminta soal.

Penyelesaian Java. Kode 12.27.

Kode 12.27 `DictionaryTreeImpl.java`: *trie* dengan delapan anak per simpul, diisi dan dicari melalui delegasi; hasilnya dipotong sepanjang signature.

```java
// DictionaryTreeImpl.java - Soal 12.8: kamus sebagai pohon dengan paling banyak
// delapan cabang per simpul, satu untuk setiap tombol 2..9; ia juga mencocokkan prefiks.
package predictive;

import java.io.File;
import java.io.IOException;
import java.nio.charset.StandardCharsets;
import java.util.Collections;
import java.util.HashSet;
import java.util.Scanner;
import java.util.Set;

public final class DictionaryTreeImpl implements Dictionary {

    // kata utuh yang signature-nya diawali lintasan dari akar sampai ke sini
    private final Set<String> words = new HashSet<>();
    // children[0] adalah cabang untuk tombol '2', ..., children[7] untuk tombol '9'
    private final DictionaryTreeImpl[] children = new DictionaryTreeImpl[8];

    /** Membangun akar dan menyisipkan setiap kata sah dari berkas. */
    public DictionaryTreeImpl(String path) throws IOException {
        try (Scanner in = new Scanner(new File(path), StandardCharsets.UTF_8)) {
            while (in.hasNextLine()) {
                String word = in.nextLine().trim().toLowerCase();
                if (PredictivePrototype.isValidWord(word)) {
                    insert(word);
                }
            }
        }
    }

    private DictionaryTreeImpl() {                // simpul dalam yang kosong
    }

    /** Menyisipkan satu kata di sepanjang lintasan signature-nya. Akar tidak menyimpan apa
pun. */
    public void insert(String word) {
        insert(PredictivePrototype.wordToSignature(word), 0, word);
    }
```

Kode 12.27 (Lanjutan) *DictionaryTreeImpl.java: trie dengan delapan anak per simpul, diisi dan dicari melalui delegasi; hasilnya dipotong sepanjang signature.*

```
    private void insert(String signature, int pos, String word) {
        if (pos == signature.length()) {
            return;                                  // seluruh signature sudah terpakai
        }
        int i = childIndex(signature.charAt(pos));
        if (children[i] == null) {
            children[i] = new DictionaryTreeImpl();
        }
        children[i].words.add(word);                 // lintasan anak saya adalah prefiks
dari word
        children[i].insert(signature, pos + 1, word);    // delegasi: sisanya tugas anak
itu
    }

    /** Simpul di ujung lintasan yang dieja signature, atau null. */
    private DictionaryTreeImpl find(String signature, int pos) {
        if (pos == signature.length()) {
            return this;
        }
        int i = childIndex(signature.charAt(pos));
        if (i < 0 || children[i] == null) {
            return null;                             // tidak ada tombol itu, atau tidak ada
kata yang lewat sini
        }
        return children[i].find(signature, pos + 1);
    }

    private static int childIndex(char key) {
        return (key >= '2' && key <= '9') ? key - '2' : -1;
    }

    @Override
    public Set<String> signatureToWords(String signature) {
        DictionaryTreeImpl node = find(signature, 0);
        if (node == null) {
            return Collections.emptySet();
        }
        return trim(node.words, signature.length());
    }

    /** n huruf pertama setiap kata; prefiks yang sama melebur menjadi satu. */
    private static Set<String> trim(Set<String> words, int n) {
        Set<String> result = new HashSet<>();
        for (String w : words) {
            result.add(w.substring(0, n));
        }
        return result;
    }
```

Kode 12.27 (Lanjutan) *DictionaryTreeImpl.java: trie dengan delapan anak per simpul, diisi dan dicari melalui delegasi; hasilnya dipotong sepanjang signature.*

```
    /** Banyaknya simpul, akar termasuk. */
    public int nodeCount() {
        int count = 1;
        for (DictionaryTreeImpl child : children) {
            if (child != null) {
                count += child.nodeCount();
            }
        }
        return count;
    }
}
```

Kode 12.28 menguji kata utuh dan prefiks, serta memperagakan penyelesaian keliru di bawah.

Kode 12.28 `TreeCheck.java`: pemeriksaan Soal 12.8 dan pohon yang menyimpan kata hanya di ujung lintasannya.

```
// TreeCheck.java - pemeriksaan Soal 12.8, dengan penyelesaian keliru yang menggoda, yang
// menyimpan setiap kata hanya di ujung lintasannya.
package predictive;

import java.util.HashSet;
import java.util.Set;
import java.util.TreeSet;

public class TreeCheck {

    /** Menggoda tetapi salah: kata hanya disimpan di simpul TERAKHIR lintasannya. */
    static final class EndOnlyTree {
        private final Set<String> words = new HashSet<>();
        private final EndOnlyTree[] children = new EndOnlyTree[8];

        void insert(String word) {
            EndOnlyTree node = this;
            for (char key : PredictivePrototype.wordToSignature(word).toCharArray()) {
                int i = key - '2';
                if (node.children[i] == null) {
                    node.children[i] = new EndOnlyTree();
                }
                node = node.children[i];
            }
            node.words.add(word);
        }

        Set<String> signatureToWords(String signature) {
            EndOnlyTree node = this;
            for (char key : signature.toCharArray()) {
                node = node.children[key - '2'];
                if (node == null) {
                    return Set.of();
                }
            }
            return node.words;
        }
    }

    private static int passed = 0;
```

Kode 12.28 (Lanjutan) *TreeCheck.java: pemeriksaan Soal 12.8 dan pohon yang menyimpan kata hanya di ujung lintasannya.*

```
    private static void check(String name, Object expected, Object actual) {
        if (!java.util.Objects.equals(expected, actual)) {
            throw new AssertionError(name + ": expected " + expected + " but got " +
actual);
        }
        passed++;
        System.out.println("PASS " + name);
    }

    private static String sorted(Set<String> words) {
        return new TreeSet<>(words).toString();
    }

    public static void main(String[] args) throws Exception {
        DictionaryTreeImpl d = new DictionaryTreeImpl("words.txt");
        check("4663", "[gone, good, goof, home, hone, hood, hoof]",
sorted(d.signatureToWords("4663")));
        check("96753", "[worke, world]", sorted(d.signatureToWords("96753")));
        check("9675", "[work, worl]", sorted(d.signatureToWords("9675")));
        check("46", "[go, ho, in]", sorted(d.signatureToWords("46")));
        check("4", "[g, h, i]", sorted(d.signatureToWords("4")));
        check("empty", "[]", sorted(d.signatureToWords("")));
        check("no such key", "[]", sorted(d.signatureToWords("41")));
        check("too long", "[]", sorted(d.signatureToWords("4663333")));
        System.out.println("nodes            : " + d.nodeCount());
        d.insert("hoop");
        check("insert hoop", "[hoop]", sorted(d.signatureToWords("4667")));

        EndOnlyTree wrong = new EndOnlyTree();
        for (String w : new String[] {"work", "worked", "world", "worlds"}) {
            wrong.insert(w);
        }
        System.out.println("end-only 9675    : " + sorted(wrong.signatureToWords("9675")));
        System.out.println("end-only 96753   : " +
sorted(wrong.signatureToWords("96753")));
        System.out.println("All " + passed + " checks passed.");
    }
}
```

```
PASS 4663
PASS 96753
PASS 9675
PASS 46
PASS 4
PASS empty
PASS no such key
PASS too long
nodes            : 121
PASS insert hoop
end-only 9675    : [work]
end-only 96753   : [world]
All 9 checks passed.
```

Jawabannya sama dengan *list* dan *map* untuk 4663; untuk 96753 pohon menambahkan `worke` (dari `worked` dan `worker`), untuk 9675 ia memberikan `work` dan `worl`, dan untuk tombol tunggal 4 huruf-huruf `g`, `h`, dan `i`. Signature yang lebih panjang daripada setiap lintasan tidak menemukan simpul (`too long`).

Penyelesaian Python. **Kode 12.29** memisahkan simpul, kelas privat kecil `_Node`, dari `DictionaryTreeImpl` yang publik, yang memegang akarnya. Rekursinya tinggal di `_Node`, dan **Kode 12.30** memuat ujinya.

Kode 12.29 `dictionary_tree_impl.py`: *trie* yang sama dalam Python; `_Node` menjalankan rekursinya dan `_trim` berupa *set comprehension*.

```
# dictionary_tree_impl.py - Soal 12.8: kamus sebagai pohon dengan paling banyak
# delapan cabang per simpul, satu untuk setiap tombol 2..9; ia juga mencocokkan prefiks.
from predictive.dictionary import Dictionary
from predictive.prototype import is_valid_word, word_to_signature


def _child_index(key):
    return ord(key) - ord("2") if "2" <= key <= "9" else -1


class _Node:
    __slots__ = ("words", "children")

    def __init__(self):
        self.words = set()           # kata utuh yang signature-nya diawali lintasan saya
        self.children = [None] * 8   # children[0] adalah tombol '2', ..., children[7]
adalah '9'

    def insert(self, signature, pos, word):
        if pos == len(signature):
            return                                  # seluruh signature sudah terpakai
        i = _child_index(signature[pos])
        if self.children[i] is None:
            self.children[i] = _Node()
        self.children[i].words.add(word)            # lintasan anak saya adalah prefiks
dari word
        self.children[i].insert(signature, pos + 1, word)   # delegasi

    def find(self, signature, pos):
        """Simpul di ujung lintasan yang dieja signature, atau None."""
        if pos == len(signature):
            return self
        i = _child_index(signature[pos])
        if i < 0 or self.children[i] is None:
            return None
        return self.children[i].find(signature, pos + 1)

    def node_count(self):
        return 1 + sum(c.node_count() for c in self.children if c is not None)


class DictionaryTreeImpl(Dictionary):
```

Kode 12.29 (Lanjutan) *dictionary_tree_impl.py: trie yang sama dalam Python; _Node menjalankan rekursinya dan _trim berupa set comprehension.*

```
    def __init__(self, path):
        """Bangun akar dan sisipkan setiap kata sah dari berkas."""
        self._root = _Node()                          # akar tidak menyimpan kata
        with open(path, encoding="utf-8") as f:
            for line in f:
                word = line.strip().lower()
                if is_valid_word(word):
                    self.insert(word)

    def insert(self, word):
        self._root.insert(word_to_signature(word), 0, word)

    def signature_to_words(self, signature):
        node = self._root.find(signature, 0)
        if node is None:
            return set()
        return self._trim(node.words, len(signature))

    @staticmethod
    def _trim(words, n):
        """n huruf pertama setiap kata; prefiks yang sama melebur menjadi satu."""
        return {w[:n] for w in words}

    def node_count(self):
        return self._root.node_count()
```

Kode 12.30 `tree_check.py`: uji Soal 12.8 dalam Python, dan pohon yang hanya menyimpan di ujung.

```
# tree_check.py - uji Soal 12.8, dengan penyelesaian keliru yang menggoda, yang
# menyimpan setiap kata hanya di ujung lintasannya.
import sys
import unittest

from predictive.dictionary_tree_impl import DictionaryTreeImpl
from predictive.prototype import word_to_signature


class EndOnlyTree:
    """Menggoda tetapi salah: kata hanya disimpan di simpul TERAKHIR lintasannya."""

    def __init__(self):
        self.words = set()
        self.children = [None] * 8

    def insert(self, word):
        node = self
        for key in word_to_signature(word):
            i = ord(key) - ord("2")
            if node.children[i] is None:
                node.children[i] = EndOnlyTree()
            node = node.children[i]
        node.words.add(word)

    def signature_to_words(self, signature):
        node = self
        for key in signature:
            node = node.children[ord(key) - ord("2")]
            if node is None:
                return set()
        return node.words


class TestTreeImpl(unittest.TestCase):
    @classmethod
    def setUpClass(cls):
        cls.d = DictionaryTreeImpl("words.txt")

    def check(self, expected, signature):
        self.assertEqual(expected, sorted(self.d.signature_to_words(signature)))
```

Kode 12.30 (Lanjutan) *tree_check.py: uji Soal 12.8 dalam Python, dan pohon yang hanya menyimpan di ujung.*

```
    def test_whole_words_and_prefixes(self):
        self.check(["gone", "good", "goof", "home", "hone", "hood", "hoof"], "4663")
        self.check(["worke", "world"], "96753")
        self.check(["work", "worl"], "9675")
        self.check(["go", "ho", "in"], "46")
        self.check(["g", "h", "i"], "4")

    def test_nothing_matches(self):
        for sig in ("", "41", "4663333"):
            self.check([], sig)

    def test_insert(self):
        d = DictionaryTreeImpl("words.txt")
        d.insert("hoop")
        self.assertEqual({"hoop"}, d.signature_to_words("4667"))


if __name__ == "__main__":
    print("nodes            :", DictionaryTreeImpl("words.txt").node_count())
    wrong = EndOnlyTree()
    for w in ("work", "worked", "world", "worlds"):
        wrong.insert(w)
    print("end-only 9675    :", sorted(wrong.signature_to_words("9675")))
    print("end-only 96753   :", sorted(wrong.signature_to_words("96753")))
    sys.stdout.flush()
    unittest.main(verbosity=2)
```

```
nodes            : 121
end-only 9675    : ['work']
end-only 96753   : ['world']
test_insert (__main__.TestTreeImpl.test_insert) ... ok
test_nothing_matches (__main__.TestTreeImpl.test_nothing_matches) ... ok
test_whole_words_and_prefixes (__main__.TestTreeImpl.test_whole_words_and_prefixes) ... ok

----------------------------------------------------------------------
Ran 3 tests in 0.001s

OK
```

Kedua bahasa membangun 121 simpul yang sama (akar dan 120 lainnya) untuk kamus kita.

Baris demi baris. Pada **Kode 12.27**, kedua *field* itulah seluruh simpulnya: `words`, sebuah `HashSet<String>`, dan `children`, delapan referensi yang semula `null`. Konstruktor privat membuat simpul dalam yang kosong, yang tidak dapat dibuat oleh pihak luar. `insert(String word)` yang publik menghitung signature sekali dan memulai `insert` rekursif yang privat di posisi 0 pada akar; metode itu adalah **Tabel 12.6** kolom pertama, dan ia menambahkan kata ke *anak*, tidak pernah ke `this`, sehingga akar tetap kosong. `find` adalah kolom kedua. `trim` membangun himpunan baru berisi `w.substring(0, n)`; kata-kata yang berbagi prefiks melebur karena hasilnya berupa himpunan. `nodeCount` adalah satu delegasi lagi. Di Python, `[None] * 8` membuat delapan anak kosong, `ord(key) - ord("2")` adalah indeks anak, dan `{w[:n] for w in words}`, sebuah *set comprehension*, adalah seluruh `trim`.

Penyelesaian keliru yang menggoda. Rancangan yang menggoda menyimpan setiap kata hanya di tempat lintasannya *berakhir*, seperti *map*. `EndOnlyTree` menjawab signature utuh (`world` untuk 96753), tetapi untuk prefiks 9675, ketika `world` baru separuh diketik, ia hanya menjawab `[work]`: `world`, `worked`, dan `worlds` lewat tanpa meninggalkan jejak, dan tujuan pohon itu hilang. Kesalahan sebaliknya, menambahkan kata juga ke `this`, memenuhi akar dengan seluruh kamus

dan membuat `signatureToWords("")` mengembalikan `{""}`; itulah sebabnya ada aturan "akar tidak menyimpan kata".

> **UJI DIRI.** Sisipkan hanya `home` dan `hood` ke dalam pohon kosong. Berapa simpul yang dimilikinya (termasuk akar), dan apa yang dikembalikan `signatureToWords("466")`?
>
> *Jawaban.* Lintasannya 4-6-6-3 dan 4-6-6-3 lagi (`hood` juga bersignature 4663), jadi ada 4 simpul di bawah akar, 5 seluruhnya, dan masing-masing dari keempatnya menyimpan `{home, hood}`. Dipotong menjadi tiga huruf, keduanya menjadi `hom` dan `hoo`, sehingga jawabannya `{hom, hoo}`.

12.6.3 Soal 12.9 — Satu baris yang diubah, dan adu cepat lengkap

> **SOAL.** It should be possible to change just one line of `Sigs2WordsList` so that it works with any implementation of `Dictionary`. Write `Sigs2WordsMap` and `Sigs2WordsTree` that way. Compare the time that `Sigs2WordsMap` and `Sigs2WordsTree` take with large inputs. Can the difference between `Sigs2WordsList` and `Sigs2WordsMap`, or between `Sigs2WordsTree` and `Sigs2WordsMap`, be made noticeable? Write down the data and the timings.
>
> *Dengan kata lain:* dengan hanya mengubah satu baris `Sigs2WordsList`, buatlah `Sigs2WordsMap` dan `Sigs2WordsTree`, lalu bandingkan waktunya untuk masukan besar. Dapatkah perbedaan *list* dengan *map*, atau pohon dengan *map*, dibuat terasa? Catat data dan hasil pengukurannya.

Dengan bahasa kita sendiri. Inilah hasil dari Soal 12.5: `Sigs2WordsList` mendeklarasikan `Dictionary d` dan hanya memakai `d.signatureToWords`, sehingga kelas yang dibuat pada baris pertamanya adalah satu-satunya ikatannya dengan *list*. Lalu adu cepat dari Soal 12.6 dijalankan dengan keempat peserta.

Penyelesaian Java. **Kode 12.31** dan **Kode 12.32** berbeda dari **Kode 12.17** pada baris yang ditandai (dan, tentu saja, pada nama kelas dan pesan cara pakainya).

Kode 12.31 `Sigs2WordsMap.java`: program baris perintah di atas `DictionaryMapImpl`.

```
// Sigs2WordsMap.java - Soal 12.9: Sigs2WordsProto dengan kamus yang disimpan.
package predictive;

import java.io.IOException;
import java.util.TreeSet;

public class Sigs2WordsMap {

    public static void main(String[] args) throws IOException {
        Dictionary d = new DictionaryMapImpl(PredictivePrototype.DICTIONARY);   // satu-
satunya baris yang berbeda
        if (args.length == 0) {
            System.out.println("USAGE: java predictive.Sigs2WordsMap sig1 sig2 ... sigN");
            return;
        }
        for (String sig : args) {
            System.out.println(sig + ": " + new TreeSet<>(d.signatureToWords(sig)));
        }
    }
}
```

Kode 12.32 `Sigs2WordsTree.java`: program baris perintah di atas `DictionaryTreeImpl`.

```java
// Sigs2WordsTree.java - Soal 12.9: Sigs2WordsProto dengan kamus yang disimpan.
package predictive;

import java.io.IOException;
import java.util.TreeSet;

public class Sigs2WordsTree {

    public static void main(String[] args) throws IOException {
        Dictionary d = new DictionaryTreeImpl(PredictivePrototype.DICTIONARY);   // satu-satunya baris yang berbeda
        if (args.length == 0) {
            System.out.println("USAGE: java predictive.Sigs2WordsTree sig1 sig2 ... sigN");
            return;
        }
        for (String sig : args) {
            System.out.println(sig + ": " + new TreeSet<>(d.signatureToWords(sig)));
        }
    }
}
```

Dengan `words.txt`, `java predictive.Sigs2WordsMap 4663 43556 96753 69 6263 47` mencetak enam baris yang sama seperti *list* dan prototipe:

```
4663: [gone, good, goof, home, hone, hood, hoof]
43556: [hello]
96753: [world]
69: [my, ow, ox]
6263: [mane, name, oboe]
47: [is]
```

dan `java predictive.Sigs2WordsTree` dengan signature yang sama memperlihatkan prefiks:

```
4663: [gone, good, goof, home, hone, hood, hoof]
43556: [hello]
96753: [worke, world]
69: [my, ow, ox]
6263: [mane, name, oboe]
47: [ir, is]
```

Jawaban pohon berbeda pada dua baris, sebagaimana mestinya: `worke` untuk 96753 (dari `worked` dan `worker`) dan `ir` untuk 47 (dari `iron` dan `ironed`).

Penyelesaian Python. **Kode 12.33** dan **Kode 12.34**; keluarannya identik dengan keluaran Java.

Kode 12.33 `sigs2words_map.py`: program *map* dalam Python.

```python
# sigs2words_map.py - Soal 12.9: sigs2words_proto dengan kamus yang disimpan.
import sys

from predictive.dictionary_map_impl import DictionaryMapImpl
from predictive.prototype import DICTIONARY


def main(args):
    d = DictionaryMapImpl(DICTIONARY)                    # satu-satunya baris yang berbeda
    if not args:
        print("USAGE: python -m predictive.sigs2words_map sig1 sig2 ... sigN")
        return
    for sig in args:
        print(sig + ": [" + ", ".join(sorted(d.signature_to_words(sig))) + "]")


if __name__ == "__main__":
    main(sys.argv[1:])
```

Kode 12.34 `sigs2words_tree.py`: program pohon dalam Python.

```
# sigs2words_tree.py - Soal 12.9: sigs2words_proto dengan kamus yang disimpan.
import sys

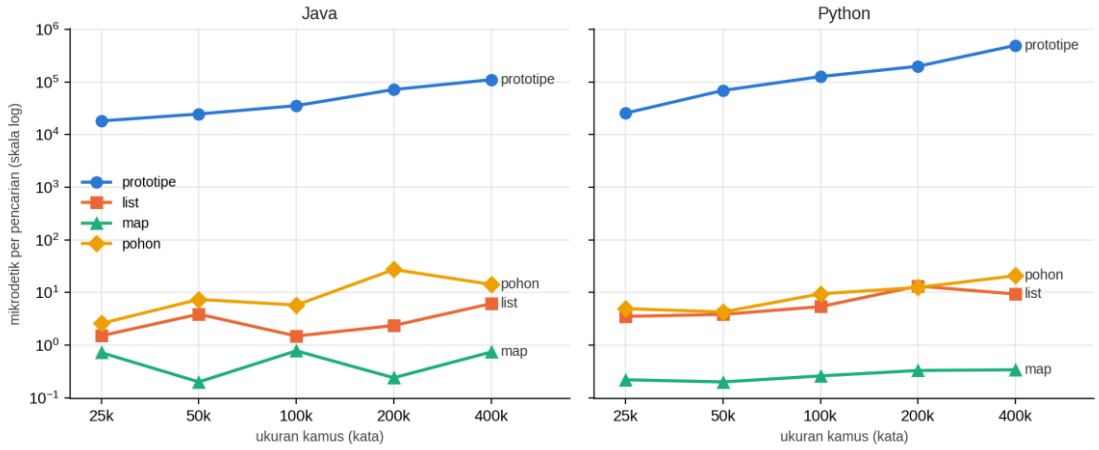
from predictive.dictionary_tree_impl import DictionaryTreeImpl
from predictive.prototype import DICTIONARY


def main(args):
    d = DictionaryTreeImpl(DICTIONARY)                  # satu-satunya baris yang berbeda
    if not args:
        print("USAGE: python -m predictive.sigs2words_tree sig1 sig2 ... sigN")
        return
    for sig in args:
        print(sig + ": [" + ", ".join(sorted(d.signature_to_words(sig))) + "]")


if __name__ == "__main__":
    main(sys.argv[1:])
```

```
4663: [gone, good, goof, home, hone, hood, hoof]
43556: [hello]
96753: [world]
69: [my, ow, ox]
6263: [mane, name, oboe]
47: [is]
4663: [gone, good, goof, home, hone, hood, hoof]
43556: [hello]
96753: [worke, world]
69: [my, ow, ox]
6263: [mane, name, oboe]
47: [ir, is]
```

Pengukurannya. Tabel waktu pada Soal 12.6 sudah mencakup keempat kamus, dan **Tabel 12.5** keempat program baris perintah. Untuk melihat bagaimana masing-masing *berskala*, kami membangkitkan kamus berukuran 25.000 sampai 400.000 kata (`MakeDictionary` menerima jumlah kata dan nama berkas sebagai argumen) dan menjalankan kedua program pengukur pada masing-masing. **Gambar 12.4** memplot waktu per pencarian terhadap ukuran pada sumbu logaritmik, sehingga garis yang naik 45 derajat berarti "sebanding dengan n" dan garis datar berarti "tidak bergantung pada n".



Gambar 12.4 Mikrodetik per pencarian terhadap ukuran kamus, pada sumbu logaritmik, diukur dengan `TimingComparison` (kiri) dan `timing_comparison.py` (kanan). Prototipe naik bersama ukurannya; *map* tetap datar; *list* dan pohon berada di antaranya.

Gambar ini menegaskan rumus-rumusnya, dengan derau pengukuran sungguhan di mesin yang dipakai bersama (titik-titik Java melompat ketika kompilasi *just-in-time* dan *garbage collection* ikut campur). Kata yang enam belas kali lebih banyak membuat prototipe kira-kira enam kali lebih lama di Java dan sembilan belas kali di Python, mendekati Persamaan (12.3). *Map* tetap datar di bawah satu mikrodetik, seperti dijanjikan Persamaan (12.8). *List* merayap naik perlahan, log$n$ dari Persamaan (12.6), dengan 15 sampai 19 perbandingan yang masing-masing menyentuh bagian memori yang berbeda. Pohon juga merayap naik dan berada *di atas list*, sebagaimana dijelaskan Persamaan (12.9): untuk signature utuh ia juga mengumpulkan dan memotong setiap kata yang lebih panjang yang diawali signature itu, sehingga ia menjawab pertanyaan yang lebih besar, seperti diperlihatkan kolom `found`-nya (7.496 lawan 5.332).

Tabel 12.7 merangkum seluruh perbandingan.

Tabel 12.7 Keempat kamus dibandingkan. n adalah banyaknya kata, $\bar{m}$ panjang rata-ratanya, k banyaknya kecocokan, ℓ panjang signature, dan W_s banyaknya kata yang tersimpan di simpulnya.

Kamus	Pembangunan	Pencarian (signature utuh)	Menyisipkan satu kata	Prefiks?	Keluaran terurut?
Prototipe	tidak ada	$O(n\bar{m})$, membaca berkas	menyunting berkas	bisa, dengan memindai	bisa, dengan mengurutkan
List terurut	$O(n \log n)$	$O(\bar{m} \log n + k)$	$O(n)$ (menggeser)	ya, dengan pencarian rentang (Latihan 12.3)	ya
Hash map	$O(n\bar{m})$ harapan	$O(\bar{m} + k)$ harapan	$O(\bar{m})$ harapan	tidak	tidak
Trie	$O\left(\sum_w \lvert w \rvert\right)$	$O(\ell + W_s \ell)$	$O(\lvert w \rvert)$	ya, bawaan	dengan menelusuri anak menurut urutan tombol

Jadi, jawaban atas soal ini: *ya*, *list* lawan *map* terasa bedanya bila pertanyaannya banyak dan kamusnya besar (di sini *map* lebih dari sepuluh kali lebih cepat per pencarian, dan juga lebih cepat dibangun), begitu pula pohon lawan *map*, dengan *map* unggul untuk signature utuh. Namun pohon menjawab pertanyaan yang sama sekali tidak dapat dijawab *map*, yaitu kata apa yang mungkin sedang diketik pengguna *saat ini*, dan itulah sebabnya telepon memakai *trie* dan GUI pada Soal 12.10 memakai pohon.

Baris demi baris. Ketiga program Java hanya berbeda pada kelas sesudah `new`. Itulah "memprogram terhadap antarmuka": karena `Dictionary` adalah satu-satunya tipe yang dikenal sisa kode, sisa kode itu tidak mungkin bergantung pada hal yang membedakan implementasi-implementasinya. Python tidak menuliskan tipe deklarasi, tetapi ketiga modul hanya memanggil `signature_to_words`, disiplin yang sama berdasarkan konvensi.

Penyelesaian keliru yang menggoda. Jalan pintasnya mendeklarasikan tipe konkret, `DictionaryListImpl d = new DictionaryListImpl(...)`, dan kelak seseorang memanggil `d.size()` karena metode itu tersedia. Kini pergantian memerlukan dua perubahan, dan perubahan kedua tidak terkompilasi untuk *map*, yang metodenya `signatureCount()`. Deklarasikan variabel dengan tipe antarmuka setiap kali antarmuka itu sudah cukup.

> **UJI DIRI.** Pada tabel Java di Soal 12.6, pohon memerlukan 13,57 µs per pencarian dan *map* 0,22. Bila telepon menanyai kamus sekali untuk setiap tekanan tombol dan pengguna menekan empat tombol per detik, apakah perbedaan itu penting?
>
> *Jawaban.* Tidak: empat pencarian masing-masing 14 µs berarti 56 µs kerja per detik, jauh di bawah apa pun yang dapat disadari manusia. Perbedaan enam puluh kali lipat penting bagi

> program yang mengajukan jutaan pertanyaan, bukan bagi orang yang bertanya empat kali per detik; di situ, sifat lain, seperti prefiks milik pohon, yang menentukan.

12.7 Bagian Empat: Papan Tombol dalam MVC

12.7.1 Soal 12.10 — GUI teks prediktif

> **SOAL.** In a new package `predictivegui`, write a graphical user interface for the classes of Parts One to Three, using the Model–View–Controller method, so that the user interface is separate from the program code and can be changed without changing it. The *model* encapsulates a `Dictionary` object: it has a constructor that loads the dictionary and a method that queries it with signatures. It may store one dictionary and a list of the words entered so far, but it must not store signatures or spaces. The *controller* is the keypad: twelve buttons. The *view* is a text window above the keypad that shows the message; it must not be possible to type into it directly. Pressing key 2 gives one of a, b or c. Pressing `*` cycles through the words that match. Pressing `0` completes the current word and starts a new one after a space. Pressing `#` deletes the last letter of the current word. The GUI must work with any implementation of `Dictionary`, without changing the existing classes.
>
> *Dengan kata lain:* dalam paket baru `predictivegui`, tulislah GUI untuk kelas-kelas Bagian Satu sampai Tiga dengan metode Model–View–Controller. Model membungkus sebuah `Dictionary`, memuat kamus lewat konstruktor, dan menanyainya dengan signature; model boleh menyimpan satu kamus dan daftar kata yang sudah dimasukkan, tetapi tidak boleh menyimpan signature atau spasi. Controller-nya adalah dua belas tombol, view-nya jendela teks di atas papan tombol yang tidak dapat diketiki langsung. `*` berpindah di antara kata yang cocok, `0` menyelesaikan kata dan memulai kata baru sesudah spasi, `#` menghapus huruf terakhir. GUI harus bekerja dengan implementasi `Dictionary` apa pun tanpa mengubah kelas yang sudah ada.

Dengan bahasa kita sendiri. Kita sedang membangun sebuah telepon kecil. Menekan 4, 6, 6, 3 menampilkan `gone`; `*` menggantinya dengan `good`, `goof`, lalu `home`; `0` menetapkan `home` dan menyisakan spasi; `#` menarik kembali satu huruf. Apa yang harus diingat di antara tekanan tombol menjadi milik model; tombol-tombol hanya melaporkan tombol mana yang ditekan; jendela teks hanya menampilkan apa yang dikatakan model.

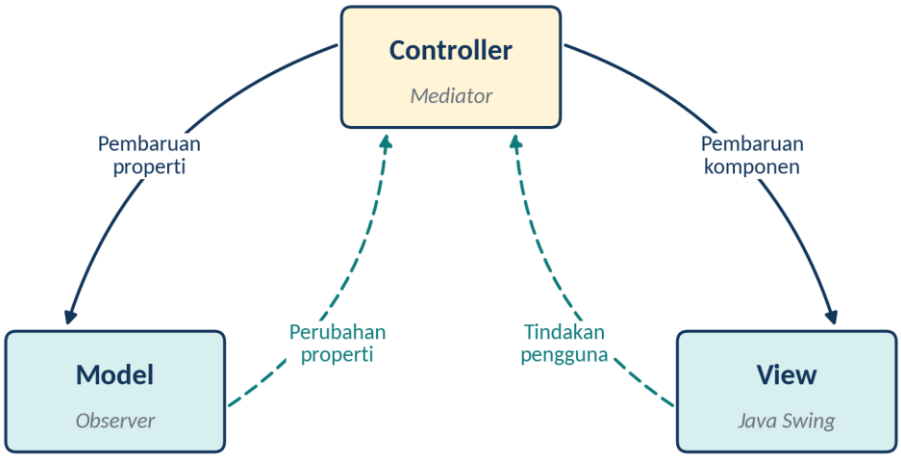
Rancangan. Langkah 4 dari Subbab 12.2 lebih dulu: *apa yang harus diingat model?* Kata-kata yang sudah selesai (`home`), kandidat untuk kata saat ini (`gone`, `good`, `goof`, `home`), dan kandidat mana yang sedang ditampilkan. Tidak lebih: angka-angka yang ditekan tidak perlu disimpan, karena kata yang ditampilkan memiliki satu huruf per tombol yang ditekan, sehingga signature-nya *adalah* angka-angka yang ditekan itu. Begitulah kita menaati “tidak ada signature di dalam model” tanpa kehilangan apa pun. **Tabel 12.8** mendaftar apa yang dilakukan setiap tombol.

Tabel 12.8 Tiga *field* model (kata-kata yang selesai, kandidat, indeks kandidat yang ditampilkan) dan pengaruh setiap tombol terhadapnya. Tidak ada signature yang pernah disimpan.

Tombol	Pengaruhnya pada model
`2`–`9`	kandidat baru = kata-kata untuk (signature kata yang ditampilkan) + tombol; bila tidak ada, tombolnya diabaikan
`*`	tampilkan kandidat berikutnya; sesudah yang terakhir, kembali ke yang pertama
`0`	tambahkan kata yang ditampilkan ke kata-kata yang selesai; tidak ada kandidat lagi
`#`	kandidat baru = kata-kata untuk signature kata yang ditampilkan tanpa angka terakhirnya
`1`, lainnya	tidak ada

Tombol yang tidak cocok dengan apa pun diabaikan: menampilkan angka sebagai gantinya akan memaksa model menyimpannya, dan telepon yang menolak tombol yang mustahil berperilaku sebagaimana telepon. Dengan kamus pohon, sebuah tombol hanya ditolak bila tidak ada satu kata pun yang diawali tombol-tombol yang sudah ditekan; sesudah 4, 6, 6, kandidatnya adalah prefiks `gon`, `goo`, `hom`, `hon`, `hoo`, dan `inn`.

Selebihnya adalah struktur MVC dari Bab 11 [10], yang digambarkan pada **Gambar 12.5**. Tindakan pengguna pada tombol-tombol view sampai ke controller; controller memanggil `model.press(key)`; model berubah dan memberi tahu setiap *listener* yang terdaftar; view, salah satu *listener* itu, meminta teksnya kepada model dan menampilkannya.



Gambar 12.5 Pola Model–View–Controller. Controller menjadi perantara: ia mengubah tindakan pengguna menjadi pembaruan model, dan perubahan model sampai ke view.

Untuk pemberitahuan, model menyimpan `List<Runnable>` berisi *listener* dan menjalankan masing-masing setelah setiap perubahan. Kode yang lebih lama, termasuk versi pertama program ini, sebagai gantinya menurunkan `java.util.Observable`, yang sudah *deprecated* sejak Java 9 (“masih ada, jangan dipakai dalam kode baru”) karena memaksa model menurunkan sebuah kelas dan tidak memberi tahu *listener* apa yang berubah [5].

Gambar 12.6 memperlihatkan jendela versi pertama itu, sekaligus sebuah peringatan: tombol 9 berlabel `vwxy`, padahal **Gambar 12.1** menaruh w, x, y, dan z pada 9 dan v pada 8. Label itu diketik manual di controller, terpisah dari huruf-huruf yang dipakai kode signature, dan tidak ada uji yang membandingkan keduanya. Controller kita mengambil labelnya dari satu tabel di samping tombol-tombolnya, sesuai dengan `keyOf`.



Gambar 12.6 Jendela versi awal program ini, dibangun ulang dengan label tombol yang diketik tangan, sebelum tombol apa pun ditekan. Label tombol 9 keliru: tertulis `vwxy`, bukan `wxyz`.

Penyelesaian Java. Empat kelas, satu untuk setiap peran, ditambah sebuah uji. Model, **Kode 12.35**, hanya bergantung pada antarmuka `Dictionary`; konstruktor keduanya adalah kemudahan yang membangun pohon.

Kode 12.35 `DictionaryModel.java`: modelnya. Ia menyimpan kata-kata yang selesai, kandidat, dan sebuah indeks, serta menghitung ulang signature dari kata yang ditampilkan.

```java
// DictionaryModel.java - Soal 12.10: model GUI papan tombol. Ia memegang
// kamus, kata-kata yang selesai, dan kandidat untuk kata yang sedang diketik.
package predictivegui;

import java.io.IOException;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import java.util.Set;
import java.util.TreeSet;

import predictive.Dictionary;
import predictive.DictionaryTreeImpl;
import predictive.PredictivePrototype;

public class DictionaryModel {

    private final Dictionary dictionary;
    private final List<String> message = new ArrayList<>();      // hanya kata yang selesai
    private List<String> candidates = new ArrayList<>();         // kecocokan untuk kata
saat ini
    private int index = 0;                                        // kandidat yang sedang
ditampilkan
    private final List<Runnable> listeners = new ArrayList<>();

    /** Model di atas Dictionary apa pun: uji memberikan kamus tiruan yang kecil. */
    public DictionaryModel(Dictionary dictionary) {
        this.dictionary = dictionary;
    }

    /** Model program sebenarnya: pohon pencocok prefiks yang dibaca dari berkas. */
    public DictionaryModel(String path) throws IOException {
        this(new DictionaryTreeImpl(path));
    }

    /** Mendaftarkan kode yang dijalankan setelah setiap perubahan (metode update milik
view). */
    public void addListener(Runnable listener) {
        listeners.add(listener);
    }

    /** Menanyai kamus: kecocokan sebuah signature, menurut abjad. */
    public List<String> lookUp(String signature) {
        Set<String> found = dictionary.signatureToWords(signature);
        return new ArrayList<>(new TreeSet<>(found));
    }
```

Kode 12.35 (Lanjutan) *DictionaryModel.java: modelnya. Ia menyimpan kata-kata yang selesai, kandidat, dan sebuah indeks, serta menghitung ulang signature dari kata yang ditampilkan.*

```
    /** Satu tekanan tombol: '2'..'9' mengetik, '*' kecocokan berikutnya, '0' mengakhiri
kata, '#' menghapus. */
    public void press(char key) {
        switch (key) {
            case '0' -> finishWord();
            case '*' -> nextCandidate();
            case '#' -> deleteLetter();
            default -> typeKey(key);
        }
        for (Runnable listener : listeners) {
            listener.run();                          // setiap view menggambar ulang
dirinya
        }
    }

    private String currentWord() {
        return candidates.isEmpty() ? "" : candidates.get(index);
    }

    private void typeKey(char key) {
        if (key < '2' || key > '9') {
            return;                                  // '1' dan tombol tak dikenal tidak
berbuat apa-apa
        }
        // signature tidak pernah disimpan: ia dihitung ulang dari kata yang ditampilkan
        List<String> found = lookUp(PredictivePrototype.wordToSignature(currentWord()) +
key);
        if (!found.isEmpty()) {                      // tombol yang tidak cocok dengan apa
pun diabaikan
            candidates = found;
            index = 0;
        }
    }

    private void nextCandidate() {
        if (!candidates.isEmpty()) {
            index = (index + 1) % candidates.size(); // sesudah yang terakhir, kembali ke
yang pertama
        }
    }

    private void finishWord() {
        if (!candidates.isEmpty()) {
            message.add(currentWord());
            candidates = new ArrayList<>();
            index = 0;
        }
    }
```

Kode 12.35 (Lanjutan) *DictionaryModel.java: modelnya. Ia menyimpan kata-kata yang selesai, kandidat, dan sebuah indeks, serta menghitung ulang signature dari kata yang ditampilkan.*

```
    private void deleteLetter() {
        String word = currentWord();
        if (word.isEmpty()) {
            return;
        }
        String shorter = PredictivePrototype.wordToSignature(word).substring(0,
word.length() - 1);
        candidates = shorter.isEmpty() ? new ArrayList<>() : lookUp(shorter);
        index = 0;
    }

    /** Kata-kata yang selesai diikuti kata saat ini, bila ada. */
    public List<String> getWords() {
        List<String> all = new ArrayList<>(message);
        if (!candidates.isEmpty()) {
            all.add(currentWord());
        }
        return Collections.unmodifiableList(all);
    }

    /** Teks yang ditampilkan: kata dipisah spasi; kata yang selesai diakhiri satu spasi.
*/
    public String getText() {
        StringBuilder sb = new StringBuilder(String.join(" ", message));
        if (!message.isEmpty()) {
            sb.append(' ');
        }
        return sb.append(currentWord()).toString();
    }
}
```

View, **Kode 12.36**, adalah panel dengan area teks yang tidak dapat disunting pengguna; `update()` menyalin teks model ke dalamnya.

Kode 12.36 `MessageView.java`: view, area teks hanya-baca di dalam panel bergulir.

```java
// MessageView.java - Soal 12.10: view. Ia menampilkan teks model dan
// tidak dapat disunting dengan mengetik.
package predictivegui;

import java.awt.BorderLayout;

import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;

public final class MessageView extends JPanel {

    private final DictionaryModel model;
    private final JTextArea message = new JTextArea(5, 20);

    public MessageView(DictionaryModel model) {
        super(new BorderLayout());
        this.model = model;
        message.setEditable(false);                  // hanya papan tombol yang boleh
mengubah teks
        message.setLineWrap(true);
        message.setWrapStyleWord(true);
        message.setText("predictive text: enter text with 8 keys.");
        add(new JScrollPane(message), BorderLayout.CENTER);
    }

    /** Dipanggil model setelah setiap perubahan. */
    public void update() {
        message.setText(model.getText());
    }

    /** Apa yang sekarang ditampilkan view (dipakai uji-mandiri). */
    public String getText() {
        return message.getText();
    }
}
```

Controller, **Kode 12.37**, membangun dua belas tombol dari sebuah tabel dan memberi masing-masing *listener* yang memanggil `model.press` dengan tombolnya sendiri.

Kode 12.37 `KeypadController.java`: controller, dua belas tombol dalam kisi 4 kali 3 yang dibangun dari satu tabel tombol dan label.

```
// KeypadController.java - Soal 12.10: controller. Dua belas tombol dalam kisi
// 4 x 3; setiap tombol meneruskan tombolnya ke model dan tidak berbuat lain.
package predictivegui;

import java.awt.GridLayout;
import java.util.HashMap;
import java.util.Map;

import javax.swing.JButton;
import javax.swing.JPanel;

public final class KeypadController extends JPanel {

    private static final String[][] KEYS = {
        {"1", ""}, {"2", "abc"}, {"3", "def"},
        {"4", "ghi"}, {"5", "jkl"}, {"6", "mno"},
        {"7", "pqrs"}, {"8", "tuv"}, {"9", "wxyz"},
        {"*", "next"}, {"0", "space"}, {"#", "delete"}};

    private final Map<Character, JButton> buttons = new HashMap<>();

    public KeypadController(DictionaryModel model) {
        super(new GridLayout(4, 3));
        for (String[] k : KEYS) {
            char key = k[0].charAt(0);              // variabel baru pada setiap putaran
gelung
            JButton button = new JButton("<html><center>" + k[0] + "<br>" + k[1] +
"</center></html>");
            button.addActionListener(e -> model.press(key));
            button.setEnabled(key != '1');          // tombol 1 tidak memuat huruf
            buttons.put(key, button);
            add(button);
        }
    }

    /** Tombol milik sebuah kunci (dipakai uji-mandiri untuk mengekliknya). */
    public JButton buttonFor(char key) {
        return buttons.get(key);
    }
}
```

Kode 12.38 merangkai semuanya. Tanpa argumen, program ini membuka jendela; dengan `--selftest`, ia membangun komponen yang sama tanpa jendela, mengeklik tombol-tombol contoh dari kode, dan mencetak apa yang ditampilkan view setelah setiap klik.

Kode 12.38 `PredictiveGUI.java`: program utama, dengan mode `--selftest` yang mengeklik tombol dari kode.

```
// PredictiveGUI.java - Soal 12.10: merangkai model, view, dan controller.
// Jalankan dengan --selftest untuk mengeklik tombol dari kode, tanpa membuka jendela.
package predictivegui;

import java.awt.BorderLayout;
import java.io.IOException;

import javax.swing.JFrame;
import javax.swing.SwingUtilities;

import predictive.PredictivePrototype;

public class PredictiveGUI {

    public static void main(String[] args) throws IOException {
        DictionaryModel model = new DictionaryModel(PredictivePrototype.DICTIONARY);
        if (args.length > 0 && args[0].equals("--selftest")) {
            MessageView view = new MessageView(model);
            KeypadController keypad = new KeypadController(model);
            model.addListener(view::update);
            selfTest(view, keypad);
            return;
        }
        SwingUtilities.invokeLater(() -> {           // objek Swing hidup di thread
peristiwa
            MessageView view = new MessageView(model);
            KeypadController keypad = new KeypadController(model);
            model.addListener(view::update);
            JFrame frame = new JFrame("Predictive Text");
            frame.add(view, BorderLayout.CENTER);
            frame.add(keypad, BorderLayout.SOUTH);
            frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            frame.setSize(260, 340);
            frame.setVisible(true);
        });
    }

    /** Mengeklik tombol-tombol contoh buku dan mencetak apa yang ditampilkan view. */
    static void selfTest(MessageView view, KeypadController keypad) {
        String presses = "4663***0" + "96753#0";
        System.out.println("start      -> \"" + view.getText() + "\"");
        for (char key : presses.toCharArray()) {
            keypad.buttonFor(key).doClick();         // persis yang dilakukan klik tetikus
            System.out.println("press " + key + "    -> \"" + view.getText() + "\"");
        }
    }
}
```

`java predictivegui.PredictiveGUI --selftest`, dijalankan dari folder `java`, mencetak:

```
start      -> "predictive text: enter text with 8 keys."
press 4    -> "g"
press 6    -> "go"
press 6    -> "gon"
press 3    -> "gone"
press *    -> "good"
press *    -> "goof"
press *    -> "home"
press 0    -> "home "
press 9    -> "home w"
press 6    -> "home wo"
press 7    -> "home wor"
press 5    -> "home work"
press 3    -> "home worke"
press #    -> "home work"
press 0    -> "home work "
```

Bacalah keluaran itu sebagai film layar: `g` sesudah 4; `gon`, yang pertama dari enam prefiks, sesudah 4, 6, 6; kata-kata 4663 sesudah angka 3, dilalui dengan `*` sampai `home`; `0` menetapkannya. Pada kata kedua, `3` memberikan `worke` (dari `worked`), `#` menarik kembali huruf `e` dan model mencari 9675 dari awal, yang kandidat pertamanya `work`.

Kode 12.39 menguji model sendirian. Karena `Dictionary` hanya memiliki satu metode, lambda di atas sebuah `Map` kecil sudah merupakan kamus tiruan yang lengkap, dan ujinya tidak memerlukan berkas.

Kode 12.39 `ModelCheck.java`: uji model sendirian, dengan kamus tiruan yang ditulis sebagai lambda.

```
// ModelCheck.java - Soal 12.10: uji untuk model saja, dengan kamus
// tiruan yang ditulis sebagai lambda. Tanpa jendela, tanpa tombol.
package predictivegui;

import java.util.Map;
import java.util.Set;

import predictive.Dictionary;

public class ModelCheck {

    private static int passed = 0;

    private static void check(String name, Object expected, Object actual) {
        if (!java.util.Objects.equals(expected, actual)) {
            throw new AssertionError(name + ": expected " + expected + " but got " +
actual);
        }
        passed++;
        System.out.println("PASS " + name);
    }

    private static void type(DictionaryModel m, String keys) {
        for (char k : keys.toCharArray()) {
            m.press(k);
        }
    }

    public static void main(String[] args) {
        Map<String, Set<String>> table = Map.of(
                "4", Set.of("g", "h", "i"), "43", Set.of("he", "if"),
                "435", Set.of("hel"), "4355", Set.of("hell"), "43556", Set.of("hello"),
                "9", Set.of("w"), "96", Set.of("wo", "yo"));
        Dictionary fake = sig -> table.getOrDefault(sig, Set.of());   // Dictionary hanya
memiliki satu metode

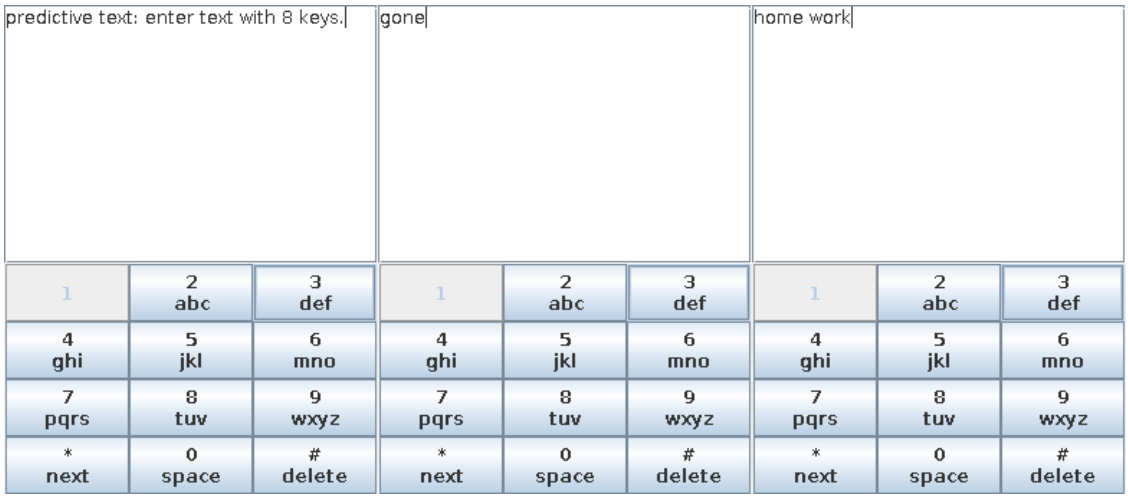
        DictionaryModel m = new DictionaryModel(fake);
        int[] updates = {0};
        m.addListener(() -> updates[0]++);
```

Kode 12.39 (Lanjutan) *ModelCheck.java: uji model sendirian, dengan kamus tiruan yang ditulis sebagai lambda.*

```
        type(m, "43");
        check("first match", "he", m.getText());
        m.press('*');
        check("star cycles", "if", m.getText());
        m.press('*');
        check("star wraps", "he", m.getText());
        type(m, "556");
        check("hello", "hello", m.getText());
        m.press('7');
        check("no match: key ignored", "hello", m.getText());
        m.press('0');
        check("0 finishes a word", "hello ", m.getText());
        check("words", "[hello]", m.getWords().toString());
        m.press('0');
        check("0 again adds nothing", "[hello]", m.getWords().toString());
        type(m, "96*");
        check("second word", "hello yo", m.getText());
        type(m, "##");
        check("delete back to nothing", "hello ", m.getText());
        m.press('#');
        check("delete on empty", "hello ", m.getText());
        check("one update per press", 16, updates[0]);
        System.out.println("All " + passed + " checks passed.");
    }
}
```

```
PASS first match
PASS star cycles
PASS star wraps
PASS hello
PASS no match: key ignored
PASS 0 finishes a word
PASS words
PASS 0 again adds nothing
PASS second word
PASS delete back to nothing
PASS delete on empty
PASS one update per press
All 12 checks passed.
```

Gambar 12.7 memperlihatkan jendela yang sesungguhnya, dibuka pada layar virtual pada tiga saat.



Gambar 12.7 Jendela Swing dari **Kode 12.38** pada awal (kiri), sesudah menekan 4, 6, 6, 3 (tengah), dan sesudah `* * * 0 9 6 7 5 3 #` (kanan). Tombol 1 dinonaktifkan karena tidak memuat huruf.

Penyelesaian Python. Keempat peran yang sama. Model, **Kode 12.40**, tidak mengimpor apa pun dari Tkinter; view dan controller mengimpor `tkinter` [11] *di dalam* konstruktornya, sehingga model dan uji-mandirinya juga berjalan di tempat yang tidak memasang Tkinter (sebagian sistem Linux menyertakan Python tanpa Tkinter). **Kode 12.41** adalah view-nya, **Kode 12.42** controller-nya, dan **Kode 12.44** menguji model dengan kamus tiruan.

Kode 12.40 `dictionary_model.py`: model dalam Python; `from_file` adalah konstruktor alternatif, ditulis sebagai *class method*.

```
# dictionary_model.py - Soal 12.10: model GUI papan tombol. Ia memegang
# kamus, kata-kata yang selesai, dan kandidat untuk kata yang sedang diketik.
from predictive.dictionary_tree_impl import DictionaryTreeImpl
from predictive.prototype import word_to_signature


class DictionaryModel:

    def __init__(self, dictionary):
        """Model di atas Dictionary apa pun: uji memberikan kamus tiruan yang kecil."""
        self._dictionary = dictionary
        self._message = []            # hanya kata yang selesai
        self._candidates = []         # kecocokan untuk kata saat ini
        self._index = 0               # kandidat yang sedang ditampilkan
        self._listeners = []

    @classmethod
    def from_file(cls, path):
        """Model program sebenarnya: pohon pencocok prefiks yang dibaca dari berkas."""
        return cls(DictionaryTreeImpl(path))

    def add_listener(self, listener):
        """Daftarkan fungsi yang dipanggil setelah setiap perubahan (update milik view)."""
        self._listeners.append(listener)

    def look_up(self, signature):
        """Tanyai kamus: kecocokan sebuah signature, menurut abjad."""
        return sorted(self._dictionary.signature_to_words(signature))

    def press(self, key):
        """Satu tekanan tombol: '2'..'9' mengetik, '*' kecocokan berikutnya, '0' mengakhiri
kata, '#' menghapus."""
        if key == "0":
            self._finish_word()
        elif key == "*":
            self._next_candidate()
        elif key == "#":
            self._delete_letter()
        else:
            self._type_key(key)
        for listener in self._listeners:
            listener()                                # setiap view menggambar ulang dirinya

    def _current_word(self):
        return self._candidates[self._index] if self._candidates else ""
```

Kode 12.40 (Lanjutan) *dictionary_model.py: model dalam Python; from_file adalah konstruktor alternatif, ditulis sebagai class method.*

```
    def _type_key(self, key):
        if not "2" <= key <= "9":
            return                                  # '1' dan tombol tak dikenal tidak
berbuat apa-apa
        # signature tidak pernah disimpan: ia dihitung ulang dari kata yang ditampilkan
        found = self.look_up(word_to_signature(self._current_word()) + key)
        if found:                                   # tombol yang tidak cocok dengan apa
pun diabaikan
            self._candidates, self._index = found, 0

    def _next_candidate(self):
        if self._candidates:
            self._index = (self._index + 1) % len(self._candidates)

    def _finish_word(self):
        if self._candidates:
            self._message.append(self._current_word())
            self._candidates, self._index = [], 0

    def _delete_letter(self):
        word = self._current_word()
        if not word:
            return
        shorter = word_to_signature(word)[:-1]
        self._candidates = self.look_up(shorter) if shorter else []
        self._index = 0

    def get_words(self):
        """Kata-kata yang selesai diikuti kata saat ini, bila ada."""
        words = list(self._message)
        if self._candidates:
            words.append(self._current_word())
        return tuple(words)                         # tuple tidak dapat diubah pemanggil

    def get_text(self):
        """Teks yang ditampilkan: kata dipisah spasi; kata yang selesai diakhiri satu
spasi."""
        text = " ".join(self._message)
        if self._message:
            text += " "
        return text + self._current_word()
```

Kode 12.41 `message_view.py`: view, *widget* `Text` Tkinter yang hanya dibuka kuncinya selagi teks model ditulis ke dalamnya.

```
# message_view.py - Soal 12.10: view. Ia menampilkan teks model dan
# tidak dapat disunting dengan mengetik.


class MessageView:

    def __init__(self, parent, model):
        import tkinter as tk                        # diimpor hanya bila jendela dibangun
        self._model = model
        self._text = tk.Text(parent, width=24, height=6, wrap="word")
        self._text.pack(fill="both", expand=True)
        self._show("predictive text: enter text with 8 keys.")

    def _show(self, text):
        self._text.configure(state="normal")        # buka kunci, ganti, kunci lagi
        self._text.delete("1.0", "end")
        self._text.insert("1.0", text)
        self._text.configure(state="disabled")      # hanya papan tombol yang boleh
mengubah teks

    def update(self):
        """Dipanggil model setelah setiap perubahan."""
        self._show(self._model.get_text())

    def get_text(self):
        return self._text.get("1.0", "end-1c")
```

Kode 12.42 `keypad_controller.py`: controller; `make_command` memberi setiap tombol sebuah fungsi yang mengingat tombolnya sendiri.

```
# keypad_controller.py - Soal 12.10: controller. Dua belas tombol dalam kisi
# 4 x 3; setiap tombol meneruskan tombolnya ke model dan tidak berbuat lain.

KEYS = [("1", ""), ("2", "abc"), ("3", "def"),
        ("4", "ghi"), ("5", "jkl"), ("6", "mno"),
        ("7", "pqrs"), ("8", "tuv"), ("9", "wxyz"),
        ("*", "next"), ("0", "space"), ("#", "delete")]


def make_command(model, key):
    """Fungsi yang dipanggil sebuah tombol. Setiap panggilan make_command memiliki 'key'
sendiri."""
    return lambda: model.press(key)


class KeypadController:

    def __init__(self, parent, model):
        import tkinter as tk                        # diimpor hanya bila jendela dibangun
        frame = tk.Frame(parent)
        frame.pack(fill="both")
        self._buttons = {}
        for n, (key, letters) in enumerate(KEYS):
            button = tk.Button(frame, text=key + "\n" + letters, width=6,
                               command=make_command(model, key))
            if key == "1":
                button.configure(state="disabled")  # tombol 1 tidak memuat huruf
            button.grid(row=n // 3, column=n % 3, sticky="nsew")
            self._buttons[key] = button

    def button_for(self, key):
        return self._buttons[key]
```

Kode 12.43 `predictive_gui.py`: program utama; `--selftest` menekan tombol-tombol melalui fungsi `make_command` yang sama dengan yang dipakai tombol.

```
# predictive_gui.py - Soal 12.10: merangkai model, view, dan controller.
# Jalankan dengan --selftest untuk menekan tombol dari kode, tanpa membuka jendela.
import sys

from predictive.prototype import DICTIONARY
from predictivegui.dictionary_model import DictionaryModel
from predictivegui.keypad_controller import make_command


class TextView:
    """View pengganti untuk uji-mandiri: ia menyimpan teks alih-alih menggambarnya."""

    def __init__(self, model):
        self._model = model
        self.text = "predictive text: enter text with 8 keys."

    def update(self):
        self.text = self._model.get_text()


def self_test(model):
    view = TextView(model)
    model.add_listener(view.update)
    print('start      -> "' + view.text + '"')
    for key in "4663***0" + "96753#0":
        make_command(model, key)()                  # persis yang dipanggil klik tombol
        print("press " + key + '    -> "' + view.text + '"')


def main(args):
    model = DictionaryModel.from_file(DICTIONARY)
    if args and args[0] == "--selftest":
        self_test(model)
        return
    import tkinter as tk
    from predictivegui.keypad_controller import KeypadController
    from predictivegui.message_view import MessageView
    root = tk.Tk()
    root.title("Predictive Text")
    view = MessageView(root, model)
    KeypadController(root, model)
    model.add_listener(view.update)
    root.mainloop()
```

Kode 12.43 (Lanjutan) *predictive_gui.py: program utama; --selftest menekan tombol-tombol melalui fungsi make_command yang sama dengan yang dipakai tombol.*

```
if __name__ == "__main__":
    main(sys.argv[1:])
```

`python -m predictivegui.predictive_gui --selftest` mencetak film yang sama dengan versi Java:

```
start      -> "predictive text: enter text with 8 keys."
press 4    -> "g"
press 6    -> "go"
press 6    -> "gon"
press 3    -> "gone"
press *    -> "good"
press *    -> "goof"
press *    -> "home"
press 0    -> "home "
press 9    -> "home w"
press 6    -> "home wo"
press 7    -> "home wor"
press 5    -> "home work"
press 3    -> "home worke"
press #    -> "home work"
press 0    -> "home work "
```

Kode 12.44 `model_check.py`: uji model sendirian; kamus tiruannya adalah objek apa saja yang memiliki metode `signature_to_words`.

```
# model_check.py - Soal 12.10: uji untuk model saja, dengan kamus
# tiruan. Tanpa jendela, tanpa tombol.
import unittest

from predictivegui.dictionary_model import DictionaryModel

TABLE = {"4": {"g", "h", "i"}, "43": {"he", "if"}, "435": {"hel"}, "4355": {"hell"},
         "43556": {"hello"}, "9": {"w"}, "96": {"wo", "yo"}}


class FakeDictionary:
    """Apa saja yang memiliki signature_to_words() sudah cukup: duck typing."""

    def signature_to_words(self, signature):
        return TABLE.get(signature, set())


class TestModel(unittest.TestCase):
    def setUp(self):
        self.m = DictionaryModel(FakeDictionary())
        self.updates = 0

        def count():
            self.updates += 1
        self.m.add_listener(count)

    def type(self, keys):
        for k in keys:
            self.m.press(k)

    def test_typing_and_cycling(self):
        self.type("43")
        self.assertEqual("he", self.m.get_text())
        self.type("*")
        self.assertEqual("if", self.m.get_text())
        self.type("*")
        self.assertEqual("he", self.m.get_text())            # berputar kembali
        self.type("556")
        self.assertEqual("hello", self.m.get_text())
        self.type("7")
        self.assertEqual("hello", self.m.get_text())         # tidak cocok: tombol
diabaikan
```

Kode 12.44 (Lanjutan) *model_check.py: uji model sendirian; kamus tiruannya adalah objek apa saja yang memiliki metode signature_to_words.*

```
    def test_finishing_and_deleting(self):
        self.type("435560")
        self.assertEqual("hello ", self.m.get_text())
        self.assertEqual(("hello",), self.m.get_words())
        self.type("0")
        self.assertEqual(("hello",), self.m.get_words())     # 0 lagi tidak menambah apa
pun
        self.type("96*")
        self.assertEqual("hello yo", self.m.get_text())
        self.type("###")
        self.assertEqual("hello ", self.m.get_text())         # penghapusan berhenti pada
kosong

    def test_one_update_per_press(self):
        self.type("43*0#")
        self.assertEqual(5, self.updates)


if __name__ == "__main__":
    unittest.main(verbosity=2)
```

```
test_finishing_and_deleting (__main__.TestModel.test_finishing_and_deleting) ... ok
test_one_update_per_press (__main__.TestModel.test_one_update_per_press) ... ok
test_typing_and_cycling (__main__.TestModel.test_typing_and_cycling) ... ok

----------------------------------------------------------------------
Ran 3 tests in 0.000s

OK
```

Gambar 12.8 memperlihatkan jendela Tkinter pada ketiga saat yang sama.



Gambar 12.8 Jendela Tkinter dari **Kode 12.43** pada ketiga saat yang sama seperti pada **Gambar 12.7**.

Baris demi baris. Pada **Kode 12.35**, *field*-nya sesuai dengan **Tabel 12.8**. `press` beralih ke satu metode privat kecil per tombol lalu menjalankan setiap *listener*, ada perubahan atau tidak; uji `one update per press` mengunci perilaku itu (Latihan 12.8 memperbaikinya). Dalam `typeKey`, `wordToSignature(currentWord()) + key` berarti "angka-angka sejauh ini, ditambah yang baru". `lookUp` mengurutkan jawabannya lewat `TreeSet` dan menyalinnya ke `ArrayList`, karena model memerlukan posisi. `nextCandidate` berputar kembali dengan `%`, dan `deleteLetter` membuang angka terakhir dengan `substring` lalu mencari signature yang lebih pendek, mengosongkan kandidat bila tidak ada yang tersisa. `getWords` mengembalikan *list* yang tidak dapat diubah.

Pada **Kode 12.36**, `setEditable(false)` menjadikan model satu-satunya sumber teks. Pada **Kode 12.37**, `char key = k[0].charAt(0);` membuat variabel *baru* pada setiap putaran, yang boleh ditangkap lambda `e -> model.press(key)` karena tidak pernah ditugasi ulang (Soal 12.16 membahas baris ini). Label HTML memberi setiap tombol dua baris yang diratakan ke tengah, dan `buttonFor` memungkinkan uji-mandiri memanggil `doClick()`, yang persis melakukan apa yang dilakukan klik tetikus. Pada **Kode 12.38**, jendela dibangun di dalam `SwingUtilities.invokeLater(...)`, karena komponen Swing menjadi milik *event dispatch thread* [12]; uji-mandiri, yang tidak memiliki jendela maupun *thread* peristiwa, membangunnya di *thread* utama. View didaftarkan oleh program utama, bukan oleh konstruktornya sendiri: objek yang menyerahkan `this` sebelum konstruktornya selesai dapat dipanggil dalam keadaan setengah jadi.

Di Python, `from_file` adalah `@classmethod`, cara lazim menulis konstruktor kedua, dan `get_words` mengembalikan `tuple`. *Widget* `Text` Tkinter tidak memiliki mode "hanya-baca tetapi dapat ditulis program", sehingga `_show` membuka kuncinya, mengganti teksnya, lalu menguncinya lagi. `grid(row=n // 3, column=n % 3)` menata dua belas tombol menjadi empat baris berisi tiga, dan `make_command(model, key)` memberi masing-masing fungsinya sendiri (Soal 12.16 menjelaskan alasannya).

Penyelesaian keliru yang menggoda. Godaan pertama adalah menjadikan area teks *sebagai* model: membaca kembali pesan dengan `getText()`, mengubahnya, lalu menulisnya dengan `setText()`. Akibatnya `*` harus menebak kandidat dari teks, `#` harus menebak di mana kata dimulai, area teks harus tetap dapat disunting sehingga pengguna dapat merusak satu-satunya salinan data, dan tidak ada yang dapat diuji tanpa jendela. Godaan kedua menyimpan `currentSig` di samping kata-kata, seperti rancangan lama. Selain melanggar aturan, dua *field* yang menggambarkan satu fakta dapat tidak sepakat, misalnya setelah `#` yang hanya memperbarui salah satunya; satu *field* yang dihitung ulang bila diperlukan tidak mungkin tidak sepakat.

> **UJI DIRI.** Mulai dari model kosong dengan kamus pohon dari `words.txt`, apa yang ditampilkan view setelah tekanan `4 3 5 5 6 0 # 9 6`?
>
> *Jawaban.* `hello wo`. Tombol 4, 3, 5, 5, 6 memberikan `hello` (satu-satunya kandidat untuk 43556), `0` menetapkannya, dan `#` tidak berbuat apa-apa, karena tidak ada kata saat ini yang dapat dihapus hurufnya (kata-kata yang selesai tidak disunting). Lalu 9 memberikan `w` dan 6 memberikan `wo`, kandidat pertama untuk 96 menurut abjad.

12.8 Soal Baru yang Dikerjakan: Koleksi dan Generik

Proyek teks prediktif telah melatih *list*, himpunan, *map*, `Comparable`, antarmuka, dan GUI. Tiga soal berikut membawa tiga gagasan Bab 9 lebih jauh, masing-masing di toko Profitina Laundry: memilih koleksi dengan mengukurnya, menulis kelas generik sendiri, dan mengurutkan objek menurut beberapa kunci sekaligus.

12.8.1 Soal 12.11 — Memilih koleksi dengan mengukurnya

> **SOAL.** At the pickup counter of Profitina Laundry, every order has a number, and the shop keeps the numbers of the orders that are ready (100,000 of them on a busy day at a large branch). A scanner reads the number on a customer's slip, and the program must say at once whether that order is ready. At the end of the day, the numbers must be printed in increasing order; and when the scanner reads the same slip several times, the log of scans should keep

> each number once, in the order of its first scan. Choose a collection for each of the three jobs, justify the choice, and confirm it with measurements.
>
> *Dengan kata lain:* di loket pengambilan Profitina Laundry, pemindai membaca nomor pesanan dan program harus segera menjawab apakah pesanan itu sudah siap (ada 100.000 nomor pada hari yang sibuk). Di akhir hari, nomor-nomor itu dicetak urut naik, dan catatan pindaian menyimpan setiap nomor sekali saja, dalam urutan pindaian pertamanya. Pilih koleksi untuk ketiga tugas itu, beri alasannya, dan pastikan dengan pengukuran.

Dengan bahasa kita sendiri. Tiga tugas, tiga pertanyaan. *Apakah nomor ini ada?*, ditanyakan ribuan kali. *Semua nomor secara urut*, sekali sehari. *Buang ulangan tetapi pertahankan urutan kemunculan pertama*, untuk catatan pindaian.

Rancangan. Langkah 2 dari Subbab 12.2 mencocokkan setiap tugas dengan sebuah struktur (**Tabel 12.9**); lalu programnya mengukur.

Tabel 12.9 Tiga tugas loket pengambilan dan koleksi-koleksi yang dapat mengerjakannya. Biayanya per operasi kecuali disebutkan lain.

Tugas	Kandidat	Biaya tugas	Putusan
"Apakah x ada?"	`ArrayList` / `list`	$O(n)$: melihat setiap elemen	terlalu lambat untuk banyak pertanyaan
"Apakah x ada?"	`HashSet` / `set`	$O(1)$ harapan	pilihannya
"Apakah x ada?"	`TreeSet` / *list* terurut + `bisect`	$O(\log n)$	cukup baik, dan sekaligus terurut
"Semua secara urut"	`TreeSet` / `sorted(...)`	penelusuran urut, $O(n)$	pilihannya
"Unik, urutan kemunculan pertama"	`LinkedHashSet` / `dict.fromkeys`	$O(1)$ harapan per butir	pilihannya
"Unik, urutan kemunculan pertama"	`ArrayList` + `contains`	$O(n)$ per butir, $O(n^2)$ seluruhnya	penyelesaian keliru yang menggoda

Python tidak memiliki himpunan terurut; *list* terurut yang dicari dengan `bisect`, atau `sorted(the_set)` untuk sebuah daftar, menggantikannya. Sejak Python 3.7 sebuah `dict` mengingat urutan penyisipan, sehingga `list(dict.fromkeys(items))` membuang ulangan dengan urutan kemunculan pertama.

Penyelesaian Java. **Kode 12.45** membangun 100.000 nomor pesanan genap dalam urutan acak, mengajukan 10.000 pertanyaan (kira-kira separuhnya tentang nomor ganjil yang tidak ada), dan mengukur waktu setiap kandidat; program ini juga mencetak urutan penelusuran tiga himpunan dan mengukur dua cara membuang pindaian berulang.

Kode 12.45 `CollectionChoice.java`: uji keanggotaan pada tiga koleksi, urutan penelusuran tiga himpunan, dan dua cara membuang duplikat, semuanya diukur waktunya.

```java
// CollectionChoice.java - Soal 12.11: memilih koleksi menurut apa yang harus dikerjakannya,
// lalu mengukur. Profitina Laundry memeriksa nomor pesanan yang dipindai.
import java.util.ArrayList;
import java.util.Collections;
import java.util.HashSet;
import java.util.LinkedHashSet;
import java.util.List;
import java.util.Random;
import java.util.Set;
import java.util.TreeSet;
import java.util.function.IntPredicate;

public class CollectionChoice {

    /** Milidetik untuk semua uji keanggotaan; sekaligus menghitung yang ditemukan. */
    static double timeContains(IntPredicate contains, int[] queries, int[] hits) {
        long t0 = System.nanoTime();
        int h = 0;
        for (int q : queries) {
            if (contains.test(q)) {
                h++;
            }
        }
        hits[0] = h;
        return (System.nanoTime() - t0) / 1e6;
    }

    /** Menggoda tetapi salah: membuang duplikat dengan List.contains, O(n) per panggilan. */
    static List<Integer> dedupWithList(int[] ids) {
        List<Integer> seen = new ArrayList<>();
        for (int id : ids) {
            if (!seen.contains(id)) {
                seen.add(id);
            }
        }
        return seen;
    }
```

Kode 12.45 (Lanjutan) *CollectionChoice.java: uji keanggotaan pada tiga koleksi, urutan penelusuran tiga himpunan, dan dua cara membuang duplikat, semuanya diukur waktunya.*

```
    /** Hasil yang sama dengan LinkedHashSet: O(1) harapan per panggilan, urutan terjaga.
*/
    static List<Integer> dedupWithSet(int[] ids) {
        Set<Integer> seen = new LinkedHashSet<>();
        for (int id : ids) {
            seen.add(id);
        }
        return new ArrayList<>(seen);
    }

    public static void main(String[] args) {
        Random random = new Random(42);                // benih tetap: id yang sama pada
setiap eksekusi
        int n = 100_000;
        List<Integer> ids = new ArrayList<>();
        for (int i = 0; i < n; i++) {
            ids.add(100_000 + 2 * i);                    // hanya nomor pesanan genap
        }
        Collections.shuffle(ids, random);
        int[] queries = new int[10_000];
        for (int k = 0; k < queries.length; k++) {
            queries[k] = 100_000 + random.nextInt(2 * n);   // sekitar separuhnya ganjil:
tidak ada
        }

        List<Integer> list = new ArrayList<>(ids);
        Set<Integer> hash = new HashSet<>(ids);
        Set<Integer> tree = new TreeSet<>(ids);
        int[] hits = new int[1];
        System.out.println("contains: " + n + " ids, " + queries.length + " queries");
        System.out.printf("  ArrayList %9.2f ms%n", timeContains(list::contains, queries,
hits));
        int listHits = hits[0];
        System.out.printf("  HashSet   %9.2f ms%n", timeContains(hash::contains, queries,
hits));
        System.out.printf("  TreeSet   %9.2f ms%n", timeContains(tree::contains, queries,
hits));
        System.out.println("  same hits: " + (listHits == hits[0]) + " (" + hits[0] + ")");

        List<Integer> few = List.of(42, 7, 19, 7, 88, 3);
        System.out.println("order of " + few + ":");
        System.out.println("  HashSet       " + new HashSet<>(few));
        System.out.println("  LinkedHashSet " + new LinkedHashSet<>(few));
        System.out.println("  TreeSet       " + new TreeSet<>(few));
```

Kode 12.45 (Lanjutan) *CollectionChoice.java: uji keanggotaan pada tiga koleksi, urutan penelusuran tiga himpunan, dan dua cara membuang duplikat, semuanya diukur waktunya.*

```
        for (int m : new int[] {20_000, 40_000, 80_000}) {
            int[] scans = new int[m];
            for (int k = 0; k < m; k++) {
                scans[k] = random.nextInt(m / 2);        // banyak pindaian berulang
            }
            long t0 = System.nanoTime();
            List<Integer> a = dedupWithList(scans);
            long t1 = System.nanoTime();
            List<Integer> b = dedupWithSet(scans);
            long t2 = System.nanoTime();
            System.out.printf("dedup %6d scans: List %8.2f ms, LinkedHashSet %5.2f ms,
equal: %b%n",
                    m, (t1 - t0) / 1e6, (t2 - t1) / 1e6, a.equals(b));
        }
    }
}
```

```
contains: 100000 ids, 10000 queries
  ArrayList   1469.95 ms
  HashSet        5.45 ms
  TreeSet       15.75 ms
  same hits: true (4999)
order of [42, 7, 19, 7, 88, 3]:
  HashSet       [19, 3, 7, 88, 42]
  LinkedHashSet [42, 7, 19, 88, 3]
  TreeSet       [3, 7, 19, 42, 88]
dedup  20000 scans: List    85.85 ms, LinkedHashSet  7.73 ms, equal: true
dedup  40000 scans: List   278.55 ms, LinkedHashSet  8.15 ms, equal: true
dedup  80000 scans: List   964.64 ms, LinkedHashSet  6.37 ms, equal: true
```

Penyelesaian Python. **Kode 12.46** menjalankan percobaan yang sama dengan `list`, `set`, dan *list* terurut yang dicari dengan `bisect`.

Kode 12.46 `collection_choice.py`: percobaan yang sama dalam Python; *list* terurut dengan `bisect` menggantikan `TreeSet`.

```
# collection_choice.py - Soal 12.11: memilih koleksi menurut apa yang harus dikerjakannya,
# lalu mengukur. Profitina Laundry memeriksa nomor pesanan yang dipindai.
import random
import time
from bisect import bisect_left


def time_contains(contains, queries):
    """Milidetik untuk semua uji keanggotaan, dan banyaknya yang ditemukan."""
    t0 = time.perf_counter()
    hits = sum(1 for q in queries if contains(q))
    return (time.perf_counter() - t0) * 1e3, hits


def in_sorted(sorted_list, x):
    """Pencarian biner pada list terurut: pengganti TreeSet di Python."""
    i = bisect_left(sorted_list, x)
    return i < len(sorted_list) and sorted_list[i] == x


def dedup_with_list(ids):
    """Menggoda tetapi salah: membuang duplikat dengan 'in' pada list, O(n) per uji."""
    seen = []
    for i in ids:
        if i not in seen:
            seen.append(i)
    return seen


def dedup_with_dict(ids):
    """Hasil yang sama dengan kunci dict: O(1) harapan per uji, urutan terjaga."""
    return list(dict.fromkeys(ids))


def main():
    rng = random.Random(42)                      # benih tetap: id yang sama pada setiap
eksekusi
    n = 100_000
    ids = [100_000 + 2 * i for i in range(n)]    # hanya nomor pesanan genap
    rng.shuffle(ids)
    queries = [100_000 + rng.randrange(2 * n) for _ in range(10_000)]
```

Kode 12.46 (Lanjutan) *collection_choice.py: percobaan yang sama dalam Python; list terurut dengan bisect menggantikan TreeSet.*

```
    as_list, as_set, as_sorted = list(ids), set(ids), sorted(ids)
    print(f"contains: {n} ids, {len(queries)} queries")
    ms, list_hits = time_contains(as_list.__contains__, queries)
    print(f"  list      {ms:9.2f} ms")
    ms, hits = time_contains(as_set.__contains__, queries)
    print(f"  set       {ms:9.2f} ms")
    ms, hits = time_contains(lambda q: in_sorted(as_sorted, q), queries)
    print(f"  sorted    {ms:9.2f} ms")
    print(f"  same hits: {list_hits == hits} ({hits})")

    few = [42, 7, 19, 7, 88, 3]
    print(f"order of {few}:")
    print("  set           ", set(few))
    print("  dict.fromkeys ", list(dict.fromkeys(few)))
    print("  sorted(set)   ", sorted(set(few)))

    for m in (20_000, 40_000, 80_000):
        scans = [rng.randrange(m // 2) for _ in range(m)]   # banyak pindaian berulang
        t0 = time.perf_counter()
        a = dedup_with_list(scans)
        t1 = time.perf_counter()
        b = dedup_with_dict(scans)
        t2 = time.perf_counter()
        print(f"dedup {m:6d} scans: list {(t1 - t0) * 1e3:9.2f} ms, "
              f"dict {(t2 - t1) * 1e3:5.2f} ms, equal: {a == b}")


if __name__ == "__main__":
    main()
```

```
contains: 100000 ids, 10000 queries
  list        6462.39 ms
  set            1.30 ms
  sorted         4.93 ms
  same hits: True (4961)
order of [42, 7, 19, 7, 88, 3]:
  set           {3, 7, 42, 19, 88}
  dict.fromkeys  [42, 7, 19, 88, 3]
  sorted(set)    [3, 7, 19, 42, 88]
dedup  20000 scans: list    327.45 ms, dict  0.83 ms, equal: True
dedup  40000 scans: list   1669.68 ms, dict  1.87 ms, equal: True
dedup  80000 scans: list   6025.80 ms, dict  6.00 ms, equal: True
```

Pengukurannya berbicara terang: untuk 10.000 pertanyaan, `ArrayList` 1.470 ms lawan `HashSet` 5,5 ms, selisih kira-kira 270 kali; di Python, 6,5 *detik* lawan 1,3 ms, kira-kira 5.000 kali. Struktur terurut berada di antaranya (`TreeSet` 15,8 ms, `bisect` 4,9 ms). Semua kandidat menemukan jumlah yang sama (4.999 di Java, 4.961 di Python, yang bilangan acaknya berbeda dari Java untuk benih yang sama).

Membuang pindaian berulang memperlihatkan $O(n^2)$ lawan $O(n)$: setiap penggandaan jumlah pindaian melipatkan waktu versi `List` sebesar 3,2 dan 3,5 kali di Java serta 5,1 dan 3,6 kali di Python, mendekati faktor 4 dari hukum kuadrat, sedangkan `LinkedHashSet` dan `dict` tetap dalam beberapa milidetik, dengan hasil yang sama.

Baris demi baris. Pada **Kode 12.45**, `new Random(42)` menetapkan benihnya dan `Collections.shuffle` mengacak nomor-nomornya agar *list* tidak diuntungkan urutan yang kebetulan baik. `timeContains` menerima ujinya sebagai `IntPredicate`, yang dipasok oleh `list::contains`, `hash::contains`, dan `tree::contains`. Pindaian diambil dari `m / 2` nomor,

sehingga banyak yang berulang. Pada **Kode 12.46**, `as_set.__contains__` adalah metode di balik operator `in`, dan `in_sorted` adalah `bisect_left` ditambah satu perbandingan.

Contoh kecilnya layak ditengok sekali lagi. `HashSet` mencetak `[19, 3, 7, 88, 42]`, urutan ember, tidak terurut dan tidak pula sesuai urutan masuk, dan sama pada setiap eksekusi karena kode *hash* bilangan bulat kecil adalah bilangan itu sendiri, yang membuatnya menggoda untuk dipercaya. `set` di Python mencetak urutan lain lagi. Hanya koleksi yang mengikuti urutan penyisipan (`LinkedHashSet`, `dict.fromkeys`) dan yang terurut (`TreeSet`, `sorted`) yang *menjanjikan* urutan.

Penyelesaian keliru yang menggoda. Penyelesaian pertama yang ditulis semua orang adalah `if (!seen.contains(id)) seen.add(id);` dengan `ArrayList`: benar, dan seketika pada dua puluh pindaian. Namun `contains` menelusuri *list*, sehingga n pindaian memakan sampai $n(n+1)/2$ langkah, jumlah pada Persamaan (12.2): di bawah satu detik untuk 80.000 pindaian di Java, enam detik di Python, dan menjadi empat kali lipat setiap kali masukannya berlipat dua. Perbaikannya hanya mengganti satu tipe: `LinkedHashSet` mempertahankan urutan kemunculan pertama *dan* menjawab `contains` dalam waktu konstan.

> **UJI DIRI.** Loket juga memerlukan "pesanan siap yang nomornya antara 150.000 dan 150.100". Himpunan mana dari ketiganya yang dapat menjawab tanpa melihat semua 100.000 nomor?
>
> *Jawaban.* Hanya yang terurut. `TreeSet.subSet(150000, 150101)` menemukan awalnya dalam $O(\log n)$ langkah lalu menelusuri nomor-nomor yang cocok; di Python, dua panggilan `bisect_left` pada *list* terurut memberikan batas irisannya. `HashSet` tidak mengenal "di antara" dan harus menguji semua 100.000 nomor (atau semua 101 kandidat, satu `contains` untuk masing-masing, yang hanya mungkin karena nomornya bilangan bulat).

12.8.2 Soal 12.12 — Kelas generik dengan parameter tipe berbatas

> **SOAL.** A washing machine's drum is loaded from a bin. Write a generic class `Bin<T>` that holds items of one kind `T` up to a capacity in kilograms: `add(item)` puts the item in only if it still fits and reports whether it did, `load()` returns the current weight, and `heaviest()` returns the heaviest item, typed as `T`, not as `Object`. Every item must have a weight, so `T` must be restricted to types that have one. Also write a method `totalKg` that adds up the weight of a list of *any* kind of weighed thing (a list of shirts, a list of towels), and a generic method `maxOf` that returns the largest element of any list whose elements can be compared.
>
> *Dengan kata lain:* tulislah kelas generik `Bin<T>` berisi barang satu jenis `T` sampai kapasitas tertentu dalam kilogram, dengan `add`, `load`, dan `heaviest` yang mengembalikan `T`. Setiap barang harus memiliki berat, jadi `T` harus dibatasi. Tulis juga `totalKg` untuk *list* berisi benda berberat jenis apa pun, dan metode generik `maxOf` untuk elemen terbesar dari *list* yang elemennya dapat dibandingkan.

Dengan bahasa kita sendiri. Wadah kemeja hanya menerima kemeja, dan barang terberatnya adalah kemeja, lengkap dengan `owner`-nya, tanpa *cast*. Wadah itu memanggil `kg()` pada barang-barangnya, jadi ia harus tahu bahwa setiap `T` memilikinya; metode-metode bantunya sebaiknya menerima sebanyak mungkin jenis *list* yang masih aman.

Rancangan. "Setiap `T` memiliki berat" adalah *parameter tipe berbatas* (*bounded type parameter*), `<T extends Weighed>`, dengan `Weighed` sebuah antarmuka yang mendeklarasikan `double kg()`; di dalam kelas, `item.kg()` pun diizinkan. Metode-metode bantunya memakai dua perkakas lain dari Bab 9 [13]:

- `totalKg(List<? extends Weighed> things)` memakai *wildcard*. `List<Towel>` **bukan** `List<Weighed>` (penyelesaian keliru memperlihatkan alasannya); `List<? extends Weighed>`, "*list* berisi suatu tipe yang `Weighed`", menerima keduanya, dengan harga bahwa kita hanya boleh membaca darinya.
- `<T extends Comparable<? super T>> T maxOf(List<? extends T> xs)` adalah *metode generik*, yang `T`-nya disimpulkan pada setiap pemanggilan (`Integer`, `String`, ...); batasnya menyatakan bahwa sebuah `T` dapat dibandingkan dengan `T` atau dengan supertipenya.

Barang-barangnya berupa *record* (Java 16 ke atas): `record Shirt(String owner, double kg) implements Weighed { }` mendeklarasikan kelas final dengan *field* privat final, sebuah konstruktor, pengakses `owner()` dan `kg()`, serta `equals`, `hashCode`, dan `toString`, semuanya dibangkitkan otomatis; `kg()` persis yang dituntut antarmukanya.

Penyelesaian Java. Kode 12.47.

Kode 12.47 `GenericBin.java`: kelas generik berbatas `Bin<T extends Weighed>`, metode *wildcard* `totalKg`, dan metode generik `maxOf`, beserta pemeriksaannya.

```
// GenericBin.java - Soal 12.12: kelas generik dengan parameter tipe berbatas,
// metode wildcard, dan metode generik. Wadah mesin cuci menampung barang.
import java.util.ArrayList;
import java.util.List;

public class GenericBin {

    /** Apa saja yang memiliki berat. */
    interface Weighed {
        double kg();
    }

    record Shirt(String owner, double kg) implements Weighed { }

    record Towel(double kg) implements Weighed { }

    /** Wadah berisi barang SATU jenis T; T harus Weighed, jadi kg() boleh dipanggil. */
    static final class Bin<T extends Weighed> {
        private final double capacityKg;
        private final List<T> items = new ArrayList<>();

        Bin(double capacityKg) {
            this.capacityKg = capacityKg;
        }

        /** Menambahkan barang bila masih muat; mengembalikan apakah barang itu
ditambahkan. */
        boolean add(T item) {
            if (load() + item.kg() > capacityKg) {
                return false;
            }
            items.add(item);
            return true;
        }

        double load() {
            return totalKg(items);                  // List<T> adalah List<? extends
Weighed>
        }
```

Kode 12.47 (Lanjutan) *GenericBin.java: kelas generik berbatas Bin<T extends Weighed>, metode wildcard totalKg, dan metode generik maxOf, beserta pemeriksaannya.*

```
        /** Barang terberat, atau null untuk wadah kosong. */
        T heaviest() {
            T best = null;
            for (T item : items) {
                if (best == null || item.kg() > best.kg()) {
                    best = item;
                }
            }
            return best;
        }

        List<T> items() {
            return List.copyOf(items);              // salinan hanya-baca
        }
    }

    /** Berat total list berisi benda Weighed jenis APA PUN. */
    static double totalKg(List<? extends Weighed> things) {
        double sum = 0;
        for (Weighed w : things) {
            sum += w.kg();
        }
        return sum;
    }

    /** Elemen terbesar dari list apa pun yang elemennya dapat dibandingkan. */
    static <T extends Comparable<? super T>> T maxOf(List<? extends T> xs) {
        T best = xs.get(0);
        for (T x : xs) {
            if (x.compareTo(best) > 0) {
                best = x;
            }
        }
        return best;
    }

    private static int passed = 0;

    private static void check(String name, Object expected, Object actual) {
        if (!java.util.Objects.equals(expected, actual)) {
            throw new AssertionError(name + ": expected " + expected + " but got " +
actual);
        }
        passed++;
        System.out.println("PASS " + name);
    }
```

Kode 12.47 (Lanjutan) *GenericBin.java: kelas generik berbatas Bin<T extends Weighed>, metode wildcard totalKg, dan metode generik maxOf, beserta pemeriksaannya.*

```
    public static void main(String[] args) {
        Bin<Shirt> shirts = new Bin<>(1.0);
        check("add 0.4", true, shirts.add(new Shirt("Ani", 0.4)));
        check("add 0.5", true, shirts.add(new Shirt("Budi", 0.5)));
        check("0.3 does not fit", false, shirts.add(new Shirt("Citra", 0.3)));
        check("heaviest", "Budi", shirts.heaviest().owner());   // tanpa cast: heaviest()
adalah Shirt
        check("empty bin", null, new Bin<Towel>(5.0).heaviest());

        List<Towel> towels = List.of(new Towel(0.8), new Towel(1.2));
        check("totalKg of shirts", 0.9, totalKg(shirts.items()));
        check("totalKg of towels", 2.0, totalKg(towels));
        check("maxOf ints", 9, maxOf(List.of(3, 9, 4)));
        check("maxOf strings", "wash", maxOf(List.of("iron", "dry", "wash")));
        System.out.println("All " + passed + " checks passed.");
    }
}
```

```
PASS add 0.4
PASS add 0.5
PASS 0.3 does not fit
PASS heaviest
PASS empty bin
PASS totalKg of shirts
PASS totalKg of towels
PASS maxOf ints
PASS maxOf strings
All 9 checks passed.
```

Penyelesaian Python. Generik di Python berupa *type hint* (petunjuk tipe), yang diperiksa oleh perkakas seperti mypy [14] dan diabaikan ketika program berjalan. **Kode 12.48** menyatakan rancangan yang sama: `Weighed` adalah sebuah `Protocol` (kelas apa pun yang memiliki `kg()` memenuhi syarat), `T = TypeVar("T", bound=Weighed)` adalah parameter berbatasnya, `Bin(Generic[T])` kelas generiknya, dan `total_kg` menerima `Sequence[Weighed]`. Karena tidak ada yang ditegakkan saat program berjalan, `add` memeriksa sendiri `isinstance(item, Weighed)`, yang dimungkinkan oleh `@runtime_checkable`.

Kode 12.48 `generic_bin.py`: rancangan yang sama dengan `Protocol`, `TypeVar`, dan `Generic`, diperiksa mypy sebelum dijalankan dan oleh `isinstance` selagi berjalan.

```python
# generic_bin.py - Soal 12.12: kelas generik dengan variabel tipe berbatas,
# fungsi atas benda Weighed apa pun, dan fungsi generik.
# Petunjuk tipe diperiksa oleh pemeriksa tipe seperti mypy, bukan oleh Python sendiri.
import unittest
from dataclasses import dataclass
from typing import Generic, Optional, Protocol, Sequence, TypeVar, runtime_checkable


@runtime_checkable
class Weighed(Protocol):
    """Apa saja yang memiliki berat."""

    def kg(self) -> float: ...


@dataclass(frozen=True)
class Shirt:
    owner: str
    weight: float

    def kg(self) -> float:
        return self.weight


@dataclass(frozen=True)
class Towel:
    weight: float

    def kg(self) -> float:
        return self.weight


T = TypeVar("T", bound=Weighed)          # T boleh tipe apa pun yang Weighed


class Bin(Generic[T]):
    """Wadah berisi barang SATU jenis T."""

    def __init__(self, capacity_kg: float) -> None:
        self._capacity_kg = capacity_kg
        self._items: list[T] = []
```

Kode 12.48 (Lanjutan) *generic_bin.py: rancangan yang sama dengan Protocol, TypeVar, dan Generic, diperiksa mypy sebelum dijalankan dan oleh isinstance selagi berjalan.*

```
    def add(self, item: T) -> bool:
        """Tambahkan barang bila masih muat; kembalikan apakah barang itu ditambahkan."""
        if not isinstance(item, Weighed):    # Python tidak memeriksa apa pun kecuali kita
minta
            raise TypeError(f"not Weighed: {item!r}")
        if self.load() + item.kg() > self._capacity_kg:
            return False
        self._items.append(item)
        return True

    def load(self) -> float:
        return total_kg(self._items)

    def heaviest(self) -> Optional[T]:
        """Barang terberat, atau None untuk wadah kosong."""
        return max(self._items, key=lambda item: item.kg(), default=None)

    def items(self) -> tuple[T, ...]:
        return tuple(self._items)            # salinan hanya-baca


def total_kg(things: Sequence[Weighed]) -> float:
    """Berat total barisan berisi benda Weighed jenis APA PUN."""
    return sum(w.kg() for w in things)


C = TypeVar("C", int, str)               # tipe elemen yang diterima fungsi ini


def max_of(xs: Sequence[C]) -> C:
    """Elemen terbesar dari barisan yang tidak kosong."""
    best = xs[0]
    for x in xs:
        if x > best:
            best = x
    return best
```

Kode 12.48 (Lanjutan) *generic_bin.py: rancangan yang sama dengan Protocol, TypeVar, dan Generic, diperiksa mypy sebelum dijalankan dan oleh isinstance selagi berjalan.*

```
class TestGenericBin(unittest.TestCase):
    def test_bin(self) -> None:
        shirts: Bin[Shirt] = Bin(1.0)
        self.assertTrue(shirts.add(Shirt("Ani", 0.4)))
        self.assertTrue(shirts.add(Shirt("Budi", 0.5)))
        self.assertFalse(shirts.add(Shirt("Citra", 0.3)))
        heaviest = shirts.heaviest()
        self.assertEqual("Budi", heaviest.owner if heaviest else None)
        self.assertIsNone(Bin[Towel](5.0).heaviest())

    def test_wrong_item_is_refused_at_run_time(self) -> None:
        with self.assertRaises(TypeError):
            Bin[Towel](5.0).add("socks")  # type: ignore[arg-type]  # sengaja salah

    def test_functions(self) -> None:
        shirts = [Shirt("Ani", 0.4), Shirt("Budi", 0.5)]
        self.assertEqual(0.9, total_kg(shirts))
        self.assertEqual(2.0, total_kg([Towel(0.8), Towel(1.2)]))
        self.assertEqual(9, max_of([3, 9, 4]))
        self.assertEqual("wash", max_of(["iron", "dry", "wash"]))


if __name__ == "__main__":
    unittest.main(verbosity=2)
```

```
test_bin (__main__.TestGenericBin.test_bin) ... ok
test_functions (__main__.TestGenericBin.test_functions) ... ok
test_wrong_item_is_refused_at_run_time
(__main__.TestGenericBin.test_wrong_item_is_refused_at_run_time) ... ok

----------------------------------------------------------------------
Ran 3 tests in 0.000s

OK
```

Menjalankan pemeriksa tipe pada berkas itu, `python -m mypy generic_bin.py`, melaporkan:

```
Success: no issues found in 1 source file
```

Uji yang menambahkan string ke wadah handuk melakukannya *dengan sengaja*, untuk memeriksa penjaga saat berjalan; komentarnya `# type: ignore[arg-type]` memberi tahu mypy bahwa galat itu disengaja.

Baris demi baris. Pada **Kode 12.47**, `add` menolak barang yang akan melampaui kapasitas *sebelum* mengubah apa pun. `load()` meneruskan `List<T>` ke `totalKg`, yang diizinkan karena `T extends Weighed`. `heaviest()` mengembalikan `T`, sehingga ujinya dapat menulis `shirts.heaviest().owner()` tanpa *cast*, dan `items()` mengembalikan `List.copyOf` yang tidak dapat diubah. `maxOf` mengharapkan *list* yang tidak kosong, seperti `Collections.max`. Ujinya boleh membandingkan `0.4 + 0.5` dengan `0.9` secara persis karena keduanya menghasilkan `double` yang sama; pada umumnya, bandingkan jumlah bilangan titik-mengambang dengan suatu toleransi. Pada **Kode 12.48**, `@dataclass(frozen=True)` berperan sebagai *record*, `max(..., default=None)` mengembalikan `None` untuk wadah kosong, dan `C = TypeVar("C", int, str)` adalah variabel tipe yang *dibatasi pada pilihan tertentu* (*constrained*).

Penyelesaian keliru yang menggoda. Signature pertama yang wajar untuk metode bantunya adalah `static double totalKg(List<Weighed> things)`. Ia terkompilasi, dan berjalan bila dipanggil dengan `List<Weighed>`. Bila dipanggil dengan `List<Towel>`, javac berhenti:

```
WrongTotal.java:12: error: incompatible types: List<Towel> cannot be converted to
List<Weighed>
        System.out.println(totalKg(towels));
                                   ^
```

Mengapa? `List<Weighed>` mengizinkan `add(new Shirt(...))`; bila `List<Towel>` dapat diteruskan sebagai `List<Weighed>`, metode itu dapat memasukkan kemeja ke *list* handuk milik pemanggil. Karena itu generik Java bersifat *invarian*, dan `? extends Weighed` memulihkan keluwesan dengan melepaskan hak menulis. mypy menerapkan aturan yang sama pada `list` di Python dan menyarankan perbaikannya:

```
wrong_total.py:9: error: Argument 1 to "total_kg" has incompatible type "list[Towel]";
expected "list[Weighed]"  [arg-type]
wrong_total.py:9: note: "list" is invariant -- see
https://mypy.readthedocs.io/en/stable/common_issues.html#variance
wrong_total.py:9: note: Consider using "Sequence" instead, which is covariant
Found 1 error in 1 file (checked 1 source file)
```

`Sequence` bersifat hanya-baca, sehingga boleh menerima barisan berisi subtipe apa pun, penalaran yang sama di balik `? extends` [14]. Python sendiri menjalankan versi keliru itu dan mencetak total yang benar; hanya pemeriksa tipe yang melihat masalahnya.

> **UJI DIRI.** Tulislah signature metode Java `addTowels` yang menambahkan tiga handuk baru ke *list* yang diberikan pemanggil, dan yang menerima `List<Towel>`, `List<Weighed>`, maupun `List<Object>`.
>
> *Jawaban.* `static void addTowels(List<? super Towel> into)`. *List* berisi `Towel` *atau supertipe apa pun darinya* dapat menerima handuk dengan aman. Inilah separuh lain dari aturan yang dikenal sebagai "*producer `extends`, consumer `super`*" [7]: baca dari `? extends`, tulis ke `? super`.

12.8.3 Soal 12.13 — Rantai *comparator*

> **SOAL.** The pickup board of Profitina Laundry lists the day's orders. Express orders come first; among orders of the same kind, the one that is ready earlier comes first; among those, the customer's name decides alphabetically; and if all that is equal, the order number. Sort six sample orders this way in Java with a `Comparator` chain and in Python with a sort key. Also sort the orders by weight.
>
> *Dengan kata lain:* papan pengambilan Profitina Laundry mendaftar pesanan hari itu: pesanan ekspres lebih dulu; di antara pesanan sejenis, yang lebih awal siap lebih dulu; lalu nama pelanggan menurut abjad; lalu nomor pesanan. Urutkan enam contoh pesanan dengan rantai `Comparator` di Java dan dengan kunci pengurutan di Python, lalu urutkan juga menurut berat.

Dengan bahasa kita sendiri. Empat kunci, ditanyakan bergiliran: kunci yang lebih belakang baru berperan bila semua kunci sebelumnya sama. Hanya kunci pertama yang "terbalik" (ekspres, `true`, lebih dulu).

Rancangan. Java membangun urutan itu dari potongan-potongan [5]: `Comparator.comparing(Order::express, Comparator.reverseOrder())` menaruh `true` lebih dulu, lalu `.thenComparingInt(Order::readyAt)`, `.thenComparing(Order::customer)`, dan `.thenComparingInt(Order::id)`, masing-masing menyatakan arahnya sendiri. Python memakai kunci yang mengembalikan tuple, `(not o.express, o.ready_at, o.customer, o.id)`; tuple dibandingkan elemen demi elemen, dan `not o.express` bernilai `False`, yang terurut lebih dulu, untuk pesanan ekspres. **Tabel 12.10** mengerjakan contohnya dengan tangan.

Tabel 12.10 Keenam contoh pesanan dalam urutan papan, beserta kunci pengurutan masing-masing. Pesanan 105 dan 102 seri pada dua kunci pertama dan dipisahkan oleh nama pelanggan.

Id	Pelanggan	Ekspres	Siap	Kunci Python	Posisi
104	Citra	ya	11	`(False, 11, 'Citra', 104)`	1
105	Ani	ya	17	`(False, 17, 'Ani', 105)`	2
102	Budi	ya	17	`(False, 17, 'Budi', 102)`	3
106	Budi	tidak	9	`(True, 9, 'Budi', 106)`	4
103	Ani	tidak	15	`(True, 15, 'Ani', 103)`	5
101	Dewi	tidak	15	`(True, 15, 'Dewi', 101)`	6

Penyelesaian Java. **Kode 12.49** mendefinisikan pesanan sebagai `record` dan urutan papan sebagai `Comparator` konstan, bersama dua *comparator* keliru.

Kode 12.49 `OrderSorting.java`: papan pengambilan yang diurutkan dengan rantai `Comparator` empat kunci, beserta dua *comparator* keliru yang menggoda.

```java
// OrderSorting.java - Soal 12.13: rantai Comparator untuk papan pengambilan.
// Pesanan ekspres dahulu; lalu waktu siap yang lebih awal; lalu pelanggan; lalu id.
import java.util.ArrayList;
import java.util.Comparator;
import java.util.List;

public class OrderSorting {

    record Order(int id, String customer, boolean express, int readyAt, double kg) { }

    /** Benar: setiap kunci menyatakan arahnya sendiri; hanya kunci ekspres yang dibalik.
*/
    static final Comparator<Order> BOARD =
            Comparator.comparing(Order::express, Comparator.reverseOrder())
                    .thenComparingInt(Order::readyAt)
                    .thenComparing(Order::customer)
                    .thenComparingInt(Order::id);

    /** Menggoda tetapi salah: reversed() di ujung membalik SELURUH rantai. */
    static final Comparator<Order> WRONG =
            Comparator.comparing(Order::express)
                    .thenComparingInt(Order::readyAt)
                    .thenComparing(Order::customer)
                    .thenComparingInt(Order::id)
                    .reversed();

    /** Menggoda tetapi salah: mengurangkan berat lalu cast membuang pecahannya. */
    static final Comparator<Order> BY_KG_WRONG = (a, b) -> (int) (a.kg() - b.kg());

    static List<Integer> ids(List<Order> orders) {
        List<Integer> result = new ArrayList<>();
        for (Order o : orders) {
            result.add(o.id());
        }
        return result;
    }

    private static int passed = 0;

    private static void check(String name, Object expected, Object actual) {
        if (!java.util.Objects.equals(expected, actual)) {
            throw new AssertionError(name + ": expected " + expected + " but got " +
actual);
        }
        passed++;
        System.out.println("PASS " + name);
    }
```

Kode 12.49 (Lanjutan) *OrderSorting.java: papan pengambilan yang diurutkan dengan rantai Comparator empat kunci, beserta dua comparator keliru yang menggoda.*

```
    public static void main(String[] args) {
        List<Order> orders = List.of(
                new Order(101, "Dewi", false, 15, 4.5),
                new Order(102, "Budi", true, 17, 2.1),
                new Order(103, "Ani", false, 15, 6.0),
                new Order(104, "Citra", true, 11, 2.4),
                new Order(105, "Ani", true, 17, 3.0),
                new Order(106, "Budi", false, 9, 5.2));

        List<Order> board = new ArrayList<>(orders);
        board.sort(BOARD);
        for (Order o : board) {
            System.out.printf("%d %-5s %-7s ready %2d:00%n",
                    o.id(), o.customer(), o.express() ? "express" : "normal", o.readyAt());
        }
        check("board order", List.of(104, 105, 102, 106, 103, 101), ids(board));

        List<Order> wrong = new ArrayList<>(orders);
        wrong.sort(WRONG);
        System.out.println("reversed() at the end: " + ids(wrong));

        List<Order> byKg = new ArrayList<>(orders);
        byKg.sort(BY_KG_WRONG);
        System.out.println("(int) subtraction    : " + ids(byKg));
        byKg.sort(Comparator.comparingDouble(Order::kg));
        check("by kg, right", List.of(102, 104, 105, 101, 106, 103), ids(byKg));
        System.out.println("All " + passed + " checks passed.");
    }
}
```

```
104 Citra express ready 11:00
105 Ani   express ready 17:00
102 Budi  express ready 17:00
106 Budi  normal  ready  9:00
103 Ani   normal  ready 15:00
101 Dewi  normal  ready 15:00
PASS board order
reversed() at the end: [102, 105, 104, 101, 103, 106]
(int) subtraction    : [102, 104, 105, 101, 103, 106]
PASS by kg, right
All 2 checks passed.
```

Penyelesaian Python. **Kode 12.50** memakai kunci tuple, dan teknik kedua: pengurutan *stabil* dari kunci yang paling tidak penting ke yang paling penting. `sorted` di Python dijamin stabil [4], yaitu mempertahankan urutan semula elemen-elemen yang sama, sehingga urutan kunci minor bertahan di dalam setiap kelompok.

Kode 12.50 `order_sorting.py`: papan yang sama dengan kunci tuple dan dengan dua pengurutan stabil, serta dua urutan keliru yang sama.

```python
# order_sorting.py - Soal 12.13: kunci pengurutan untuk papan pengambilan.
# Pesanan ekspres dahulu; lalu waktu siap yang lebih awal; lalu pelanggan; lalu id.
import sys
import unittest
from dataclasses import dataclass
from functools import cmp_to_key


@dataclass(frozen=True)
class Order:
    id: int
    customer: str
    express: bool
    ready_at: int
    kg: float


ORDERS = [Order(101, "Dewi", False, 15, 4.5), Order(102, "Budi", True, 17, 2.1),
          Order(103, "Ani", False, 15, 6.0), Order(104, "Citra", True, 11, 2.4),
          Order(105, "Ani", True, 17, 3.0), Order(106, "Budi", False, 9, 5.2)]


def board_key(o):
    """Benar: tuple kunci, dibandingkan dari kiri ke kanan; 'not express' menaruh ekspres
di depan."""
    return (not o.express, o.ready_at, o.customer, o.id)


def board_two_passes(orders):
    """Juga benar: pengurutan stabil, dari kunci TERAKHIR ke yang pertama."""
    result = sorted(orders, key=lambda o: (o.ready_at, o.customer, o.id))
    return sorted(result, key=lambda o: o.express, reverse=True)


def wrong_board(orders):
    """Menggoda tetapi salah: reverse=True membalik SETIAP kunci dalam tuple."""
    return sorted(orders, key=lambda o: (o.express, o.ready_at, o.customer, o.id),
reverse=True)


def by_kg_wrong(a, b):
    """Menggoda tetapi salah: mengurangkan berat lalu memotong membuang pecahannya."""
    return int(a.kg - b.kg)


def ids(orders):
    return [o.id for o in orders]
```

Kode 12.50 (Lanjutan) *order_sorting.py: papan yang sama dengan kunci tuple dan dengan dua pengurutan stabil, serta dua urutan keliru yang sama.*

```
class TestBoard(unittest.TestCase):
    def test_board(self):
        self.assertEqual([104, 105, 102, 106, 103, 101], ids(sorted(ORDERS,
key=board_key)))

    def test_two_passes_agree(self):
        self.assertEqual(ids(sorted(ORDERS, key=board_key)), ids(board_two_passes(ORDERS)))

    def test_by_kg(self):
        self.assertEqual([102, 104, 105, 101, 106, 103], ids(sorted(ORDERS, key=lambda o:
o.kg)))


if __name__ == "__main__":
    for o in sorted(ORDERS, key=board_key):
        kind = "express" if o.express else "normal"
        print(f"{o.id} {o.customer:<5} {kind:<7} ready {o.ready_at:2d}:00")
    print("reverse=True         :", ids(wrong_board(ORDERS)))
    print("int() subtraction    :", ids(sorted(ORDERS, key=cmp_to_key(by_kg_wrong))))
    sys.stdout.flush()
    unittest.main(verbosity=2)
```

```
104 Citra express ready 11:00
105 Ani   express ready 17:00
102 Budi  express ready 17:00
106 Budi  normal  ready  9:00
103 Ani   normal  ready 15:00
101 Dewi  normal  ready 15:00
reverse=True         : [102, 105, 104, 101, 103, 106]
int() subtraction    : [102, 104, 105, 101, 103, 106]
test_board (__main__.TestBoard.test_board) ... ok
test_by_kg (__main__.TestBoard.test_by_kg) ... ok
test_two_passes_agree (__main__.TestBoard.test_two_passes_agree) ... ok

----------------------------------------------------------------------
Ran 3 tests in 0.000s

OK
```

Baris demi baris. *Record* `Order` membangkitkan pengakses yang dipakai oleh referensi metode. `board.sort(BOARD)` mengurutkan sebuah salinan, karena `List.of(...)` tidak dapat diubah; `%-5s` dan `%2d` meratakan kolom-kolomnya. `Comparator.comparingDouble(Order::kg)` adalah urutan berat yang benar. Pada **Kode 12.50**, `board_two_passes` mengurutkan menurut kunci-kunci minor lalu menurut `express` secara terbalik, dan `cmp_to_key` mengubah perbandingan gaya lama yang keliru menjadi sebuah kunci.

Penyelesaian keliru yang menggoda. Rantai yang menggoda menulis setiap kunci dengan arah maju lalu menambahkan `.reversed()` di ujung "agar ekspres lebih dulu". Hasilnya, `[102, 105, 104, 101, 103, 106]`, memang menaruh ekspres di depan, tetapi waktu siap yang *lebih lambat* didahulukan dan nama berjalan dari Z ke A: `.reversed()` membalik seluruh rantai. `reverse=True` berbuat sama pada kunci tuple. Baliklah hanya kunci yang memerlukannya.

Yang kedua, `(a, b) -> (int) (a.kg() - b.kg())`, memotong: 6,0 dikurangi 5,2 adalah 0,8, di-*cast* menjadi 0, "sama", sehingga 103 (6,0 kg) tetap berada sebelum 106 (5,2 kg). Pada bilangan bulat, pengurangan justru dapat meluap. `comparingDouble` dan `Integer.compare` tidak memiliki kedua cacat itu.

> **UJI DIRI.** Tulislah kunci Python yang mengurutkan pesanan menurut berat, yang terberat lebih dulu, dan untuk berat yang sama menurut id naik.
>
> *Jawaban.* `key=lambda o: (-o.kg, o.id)`. Tanda minus hanya membalik beratnya. `reverse=True` akan membalik id-nya juga. Di Java: `Comparator.comparingDouble(Order::kg).reversed().thenComparingInt(Order::id)`; di sini `.reversed()` benar karena diterapkan *sebelum* kunci kedua ditambahkan.

12.9 Soal Baru yang Dikerjakan: Peristiwa, Observer, dan MVC

Papan tombol pada Soal 12.10 menyembunyikan pelajarannya di antara banyak rincian. Tiga soal terakhir memisahkan tiga di antaranya: mengapa keadaan tinggal di dalam model, bagaimana observer bisa salah, dan mengapa tombol-tombol yang dibuat dalam gelung memerlukan kecermatan.

12.9.1 Soal 12.14 — Penghitung berbasis peristiwa dengan model yang teruji

> **SOAL.** The front desk of Profitina Laundry counts the laundry bags handed in today. Write a small window with three buttons, `+1 bag`, `-1 bag` and `reset`, and two labels: `Bags today: N` and `Machines needed: M`, where one washing machine takes 8 bags. The count may never go below zero. Organize the program as model, views and controller, test the model without any window, and give the program a `--selftest` mode that clicks the buttons from code.
>
> *Dengan kata lain:* meja depan Profitina Laundry menghitung kantong cucian yang diserahkan hari ini. Buatlah jendela kecil dengan tiga tombol (`+1 bag`, `-1 bag`, `reset`) dan dua label, jumlah kantong dan banyaknya mesin cuci yang diperlukan (satu mesin untuk 8 kantong). Jumlahnya tidak boleh di bawah nol. Susun sebagai model, view, dan controller, uji modelnya tanpa jendela, dan sediakan mode `--selftest` yang mengeklik tombol dari kode.

Dengan bahasa kita sendiri. Satu bilangan, tiga cara mengubahnya, dua cara menampilkannya: bilangan dan aturannya menjadi milik model, setiap label adalah view, dan tombol-tombolnya adalah controller.

Rancangan. Banyaknya mesin adalah banyaknya kantong dibagi 8, *dibulatkan ke atas*: 9 kantong memerlukan 2 mesin. Dengan pembagian bilangan bulat, yang membulatkan ke bawah,

$$\lceil \frac{b}{8} \rceil = \lfloor \frac{b+7}{8} \rfloor \tag{12.10}$$

untuk $b \geq 0$ kantong: $b = 8$ memberikan $\lfloor 15/8 \rfloor = 1$, $b = 9$ memberikan 2, $b = 0$ memberikan 0. Persamaan (12.10) adalah `(bags + BAGS_PER_MACHINE - 1) / BAGS_PER_MACHINE` di Java; `-(-bags // 8)` di Python membulatkan ke bawah nilai negatifnya, dengan hasil yang sama.

Model memberi tahu *listener*-nya hanya bila jumlahnya benar-benar berubah (`remove()` pada penghitung kosong tidak memberi tahu siapa pun), dan kedua view membaca model yang *sama*, sehingga keduanya tidak mungkin tidak sepakat.

Penyelesaian Java. **Kode 12.51** memuat model, kedua view, controller, dan sebuah panel keliru dalam satu berkas.

Kode 12.51 `BagCounterApp.java`: penghitung dalam bentuk MVC dengan dua view, uji-mandiri yang memeriksa model sendirian lalu mengeklik tombol-tombol yang sesungguhnya, dan panel tanpa model sebagai pembanding.

```
// BagCounterApp.java - Soal 12.14: penghitung berbasis peristiwa dalam bentuk MVC.
// Meja depan Profitina Laundry menghitung kantong hari ini; dua view menampilkan
// jumlahnya dan banyaknya mesin cuci yang diperlukan. Jalankan dengan --selftest untuk
menguji.
import java.awt.BorderLayout;
import java.awt.GridLayout;
import java.util.ArrayList;
import java.util.List;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.SwingUtilities;

public class BagCounterApp {

    /** Model: banyaknya kantong hari ini. Ia tidak tahu apa pun tentang Swing. */
    static final class BagCounter {
        static final int BAGS_PER_MACHINE = 8;
        private int bags = 0;
        private final List<Runnable> listeners = new ArrayList<>();

        void addListener(Runnable listener) {
            listeners.add(listener);
        }

        int bags() {
            return bags;
        }

        int machinesNeeded() {
            return (bags + BAGS_PER_MACHINE - 1) / BAGS_PER_MACHINE;   // dibulatkan ke
atas
        }

        void add() {
            bags++;
            changed();
        }

        void remove() {
            if (bags > 0) {                          // tidak pernah di bawah nol
                bags--;
                changed();
            }
        }
```

Kode 12.51 (Lanjutan) *BagCounterApp.java: penghitung dalam bentuk MVC dengan dua view, uji-mandiri yang memeriksa model sendirian lalu mengeklik tombol-tombol yang sesungguhnya, dan panel tanpa model sebagai pembanding.*

```
        void reset() {
            bags = 0;
            changed();
        }

        private void changed() {
            for (Runnable listener : listeners) {
                listener.run();
            }
        }
    }

    /** View 1: jumlahnya. */
    static final class CountView extends JLabel {
        private final BagCounter model;

        CountView(BagCounter model) {
            this.model = model;
            update();
        }

        void update() {
            setText("Bags today: " + model.bags());
        }
    }

    /** View 2: mesinnya. Ia berbagi model, bukan berbagi view yang lain. */
    static final class MachineView extends JLabel {
        private final BagCounter model;

        MachineView(BagCounter model) {
            this.model = model;
            update();
        }

        void update() {
            setText("Machines needed: " + model.machinesNeeded());
        }
    }

    /** Controller: tiga tombol yang hanya memanggil model. */
    static final class CounterButtons extends JPanel {
        final JButton plus = new JButton("+1 bag");
        final JButton minus = new JButton("-1 bag");
        final JButton reset = new JButton("reset");
```

Kode 12.51 (Lanjutan) *BagCounterApp.java: penghitung dalam bentuk MVC dengan dua view, uji-mandiri yang memeriksa model sendirian lalu mengeklik tombol-tombol yang sesungguhnya, dan panel tanpa model sebagai pembanding.*

```
        CounterButtons(BagCounter model) {
            super(new GridLayout(1, 3));
            plus.addActionListener(e -> model.add());
            minus.addActionListener(e -> model.remove());
            reset.addActionListener(e -> model.reset());
            add(plus);
            add(minus);
            add(reset);
        }
    }

    /** Menggoda tetapi salah: jumlahnya tinggal di teks label; tidak ada model. */
    static final class WrongCounterPanel extends JPanel {
        final JLabel count = new JLabel("Bags today: 0");
        final JLabel machines = new JLabel("Machines needed: 0");
        final JButton plus = new JButton("+1 bag");

        WrongCounterPanel() {
            plus.addActionListener(e -> {
                int n = Integer.parseInt(count.getText().substring("Bags today:
".length()));
                count.setText("Bags today: " + (n + 1));  // lalu label mesinnya? terlupa
            });
            add(count);
            add(machines);
            add(plus);
        }
    }

    private static int passed = 0;

    private static void check(String name, Object expected, Object actual) {
        if (!java.util.Objects.equals(expected, actual)) {
            throw new AssertionError(name + ": expected " + expected + " but got " +
actual);
        }
        passed++;
        System.out.println("PASS " + name);
    }
```

Kode 12.51 (Lanjutan) *BagCounterApp.java: penghitung dalam bentuk MVC dengan dua view, uji-mandiri yang memeriksa model sendirian lalu mengeklik tombol-tombol yang sesungguhnya, dan panel tanpa model sebagai pembanding.*

```
    static void selfTest() {
        BagCounter m = new BagCounter();              // model saja, sama sekali tanpa
Swing
        int[] calls = {0};
        m.addListener(() -> calls[0]++);
        m.remove();
        check("not below zero", 0, m.bags());
        check("no change, no update", 0, calls[0]);
        for (int i = 0; i < 9; i++) {
            m.add();
        }
        check("nine bags", 9, m.bags());
        check("two machines", 2, m.machinesNeeded());
        m.reset();
        check("reset", 0, m.machinesNeeded());

        BagCounter model = new BagCounter();          // sekarang model + view +
controller
        CountView countView = new CountView(model);
        MachineView machineView = new MachineView(model);
        model.addListener(countView::update);
        model.addListener(machineView::update);
        CounterButtons buttons = new CounterButtons(model);
        for (int i = 0; i < 9; i++) {
            buttons.plus.doClick();
        }
        buttons.minus.doClick();
        System.out.println("MVC  : " + countView.getText() + " | " +
machineView.getText());
        check("views agree", "Machines needed: 1", machineView.getText());

        WrongCounterPanel wrong = new WrongCounterPanel();
        for (int i = 0; i < 9; i++) {
            wrong.plus.doClick();
        }
        System.out.println("wrong: " + wrong.count.getText() + " | " +
wrong.machines.getText());
        System.out.println("All " + passed + " checks passed.");
    }
```

Kode 12.51 (Lanjutan) *BagCounterApp.java: penghitung dalam bentuk MVC dengan dua view, uji-mandiri yang memeriksa model sendirian lalu mengeklik tombol-tombol yang sesungguhnya, dan panel tanpa model sebagai pembanding.*

```
    public static void main(String[] args) {
        if (args.length > 0 && args[0].equals("--selftest")) {
            selfTest();
            return;
        }
        SwingUtilities.invokeLater(() -> {
            BagCounter model = new BagCounter();
            CountView countView = new CountView(model);
            MachineView machineView = new MachineView(model);
            model.addListener(countView::update);
            model.addListener(machineView::update);
            JFrame frame = new JFrame("Bag Counter");
            JPanel views = new JPanel(new GridLayout(2, 1));
            views.add(countView);
            views.add(machineView);
            frame.add(views, BorderLayout.CENTER);
            frame.add(new CounterButtons(model), BorderLayout.SOUTH);
            frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            frame.pack();
            frame.setVisible(true);
        });
    }
}
```

`java BagCounterApp --selftest` mencetak:

```
PASS not below zero
PASS no change, no update
PASS nine bags
PASS two machines
PASS reset
MVC  : Bags today: 8 | Machines needed: 1
PASS views agree
wrong: Bags today: 9 | Machines needed: 0
All 6 checks passed.
```

Penyelesaian Python. Pada **Kode 12.52**, sebuah view mengubah model menjadi teks dengan satu fungsi lalu menyerahkannya kepada fungsi lain, `show`, yang menuliskannya pada label Tkinter di jendela dan menyimpannya dalam sebuah *dictionary* di dalam uji; model dan view diuji tanpa Tkinter.

Kode 12.52 `bag_counter.py`: penghitung yang sama dalam Python; sebuah view menerima fungsi yang menampilkan teksnya, sehingga uji dapat mengganti labelnya.

```
# bag_counter.py - Soal 12.14: penghitung berbasis peristiwa dalam bentuk MVC.
# Meja depan Profitina Laundry menghitung kantong hari ini; dua view menampilkan
# jumlahnya dan banyaknya mesin cuci yang diperlukan. Jalankan dengan --selftest untuk
menguji.
import sys
import unittest


class BagCounter:
    """Model: banyaknya kantong hari ini. Ia tidak tahu apa pun tentang Tkinter."""
    BAGS_PER_MACHINE = 8

    def __init__(self):
        self._bags = 0
        self._listeners = []

    def add_listener(self, listener):
        self._listeners.append(listener)

    def bags(self):
        return self._bags

    def machines_needed(self):
        return -(-self._bags // self.BAGS_PER_MACHINE)     # dibulatkan ke atas

    def add(self):
        self._bags += 1
        self._changed()

    def remove(self):
        if self._bags > 0:                                 # tidak pernah di bawah nol
            self._bags -= 1
            self._changed()

    def reset(self):
        self._bags = 0
        self._changed()

    def _changed(self):
        for listener in self._listeners:
            listener()


class LabelView:
    """View: ia mengubah model menjadi teks dan menyerahkan teks itu kepada 'show'.
    'show' menaruhnya pada label Tkinter, atau menyimpannya untuk uji."""
```

Kode 12.52 (Lanjutan) *bag_counter.py: penghitung yang sama dalam Python; sebuah view menerima fungsi yang menampilkan teksnya, sehingga uji dapat mengganti labelnya.*

```
    def __init__(self, model, make_text, show):
        self._model, self._make_text, self._show = model, make_text, show
        self.update()

    def update(self):
        self._show(self._make_text(self._model))


def count_text(model):
    return f"Bags today: {model.bags()}"


def machine_text(model):
    return f"Machines needed: {model.machines_needed()}"


def wrong_plus(labels):
    """Menggoda tetapi salah: jumlahnya tinggal di teks label; tidak ada model."""
    n = int(labels["count"][len("Bags today: "):])
    labels["count"] = f"Bags today: {n + 1}"              # lalu label mesinnya? terlupa


class TestBagCounter(unittest.TestCase):
    def test_model_alone(self):
        m = BagCounter()
        calls = []
        m.add_listener(lambda: calls.append(1))
        m.remove()
        self.assertEqual(0, m.bags())
        self.assertEqual([], calls)                       # tidak ada perubahan, tidak ada
pembaruan
        for _ in range(9):
            m.add()
        self.assertEqual((9, 2), (m.bags(), m.machines_needed()))
        m.reset()
        self.assertEqual(0, m.machines_needed())
```

Kode 12.52 (Lanjutan) *bag_counter.py: penghitung yang sama dalam Python; sebuah view menerima fungsi yang menampilkan teksnya, sehingga uji dapat mengganti labelnya.*

```
    def test_views_agree(self):
        model, shown = BagCounter(), {}
        views = [LabelView(model, count_text, lambda t: shown.__setitem__("count", t)),
                 LabelView(model, machine_text, lambda t: shown.__setitem__("machines",
t))]
        for v in views:
            model.add_listener(v.update)
        for _ in range(9):
            model.add()                                  # yang dipanggil tombol "+1 bag"
        model.remove()
        self.assertEqual({"count": "Bags today: 8", "machines": "Machines needed: 1"},
shown)


def run_window():
    import tkinter as tk
    root = tk.Tk()
    root.title("Bag Counter")
    model = BagCounter()
    for make_text in (count_text, machine_text):
        label = tk.Label(root, width=24)
        label.pack()
        model.add_listener(LabelView(model, make_text, lambda t, lb=label:
lb.config(text=t)).update)
    buttons = tk.Frame(root)
    buttons.pack()
    for text, action in (("+1 bag", model.add), ("-1 bag", model.remove), ("reset",
model.reset)):
        tk.Button(buttons, text=text, command=action).pack(side="left")
    root.mainloop()


if __name__ == "__main__":
    if sys.argv[1:] == ["--selftest"]:
        labels = {"count": "Bags today: 0", "machines": "Machines needed: 0"}
        for _ in range(9):
            wrong_plus(labels)
        print("wrong:", labels["count"], "|", labels["machines"])
        sys.stdout.flush()
        unittest.main(argv=[sys.argv[0]], verbosity=2)
    else:
        run_window()
```

`python bag_counter.py --selftest` mencetak:

```
wrong: Bags today: 9 | Machines needed: 0
test_model_alone (__main__.TestBagCounter.test_model_alone) ... ok
test_views_agree (__main__.TestBagCounter.test_views_agree) ... ok

----------------------------------------------------------------------
Ran 2 tests in 0.000s

OK
```

Baris demi baris. Pada **Kode 12.51**, `BagCounter` tidak mengimpor apa pun dari Swing; sebuah halaman web dapat memakainya tanpa perubahan. `add`, `remove`, dan `reset` diakhiri `changed()`, sedangkan `remove` hanya di dalam `if`-nya. Setiap view mengisi labelnya dalam `update` dan memanggilnya sekali di konstruktornya, sehingga labelnya sudah benar sebelum klik pertama; controller hanya menghubungkan tombol dengan metode model. Uji-mandiri mula-mula memeriksa aturan model sendirian, termasuk “tidak ada perubahan, tidak ada pembaruan”, lalu

merakit objek-objek Swing yang sesungguhnya, mengeklik `+1 bag` sembilan kali dan `-1 bag` sekali, dan memeriksa bahwa kedua view menampilkan 8 kantong dan 1 mesin.

Pada **Kode 12.52**, `LabelView.update()` hanyalah `show(make_text(model))`. Dalam `run_window`, `lambda t, lb=label: lb.config(text=t)` mengikat label *saat itu* sebagai argumen bawaan (lihat Soal 12.16), dan `unittest.main(argv=[sys.argv[0]], ...)` mencegah unittest membaca `--selftest` sebagai opsinya sendiri.

Penyelesaian keliru yang menggoda. Program yang paling cepat ditulis tidak memiliki model: *listener* `+1 bag` mengurai bilangan dari teks label, menambah satu, lalu menuliskannya kembali. `WrongCounterPanel` dan `wrong_plus` melakukan hal ini, dan setelah sembilan klik menampilkan `Bags today: 9 | Machines needed: 0`: tidak ada yang mengingatkan *listener* itu tentang label kedua, sedangkan `changed()` milik model memperbarui setiap view. Jumlahnya juga tinggal di dalam teks yang diformat untuk manusia, sehingga mengganti label menjadi `Bags: 9` merusak penguraiannya, dan tidak ada yang dapat diuji tanpa mengeklik.

> **UJI DIRI.** Pemilik toko menginginkan view ketiga, label yang berubah merah bila lebih dari 40 kantong menunggu. Kelas mana yang berubah?
>
> *Jawaban.* Tidak satu pun dari yang sudah ada. Tambahkan kelas view baru (sebuah `JLabel` yang `update`-nya mengisi teks dan warna latar depan dari `model.bags()`), buat objeknya, lalu daftarkan `update`-nya dengan `model.addListener`. Model dan controller tidak tersentuh, dan itulah inti MVC.

12.9.2 Soal 12.15 — Dua *bug* observer

> **SOAL.** A detergent tank of Profitina Laundry notifies its listeners whenever detergent is used. Three listeners are registered: a one-shot alarm that rings once when the level falls below 10 liters and then unsubscribes itself, a display that shows the level, and a logger. The tank starts at 14 liters and three washes use 3 liters each. The first version of the tank has two bugs: the display ends up showing a level the tank does not have, and the notifications misbehave when the alarm unsubscribes. Find both bugs, explain them, and fix them.
>
> *Dengan kata lain:* tangki deterjen Profitina Laundry memberi tahu *listener*-nya setiap kali deterjen dipakai. Ada tiga *listener*: alarm sekali-bunyi yang berbunyi sekali ketika isi turun di bawah 10 liter lalu berhenti berlangganan, sebuah tampilan, dan pencatat. Tangki mulai dengan 14 liter, dan tiga kali cuci masing-masing memakai 3 liter. Versi pertama tangki memiliki dua *bug*: tampilan akhirnya menunjukkan isi yang tidak dimiliki tangki, dan pemberitahuan kacau ketika alarm berhenti berlangganan. Temukan, jelaskan, dan perbaiki keduanya.

Dengan bahasa kita sendiri. Setelah tiga kali cuci, tangki berisi $14 - 3 \cdot 3 = 5$ liter, tampilan menunjukkan 5, alarm sudah berbunyi sekali (pada 8), dan pencatat sudah melihat 11, 8, dan 5.

Rancangan. Pemberitahuan memiliki dua aturan yang mudah dilanggar [1]:

1. **Ubah keadaan dahulu, baru beri tahu.** *Listener* dipanggil untuk melihat keadaan yang *baru*; memberi tahu lebih dulu membuat setiap view tertinggal satu langkah.
2. **Jangan biarkan daftar *listener* berubah di bawah gelung yang menelusurinya.** Seorang *listener* boleh berhenti berlangganan (alarm sekali-bunyi) atau mendaftarkan *listener* lain selagi diberi tahu, sehingga gelung harus menelusuri *salinan* yang diambil sebelum pemanggilan pertama.

`Tank` memuat kedua versi, dipilih oleh bendera `careful`, sehingga satu eksekusi memperlihatkan *bug* dan perbaikannya berdampingan.

Penyelesaian Java. Kode 12.53.

Kode 12.53 `ObserverBugs.java`: subjek yang memberi tahu sebelum berubah dan menelusuri daftar *listener* yang hidup, di samping versi yang sudah diperbaiki.

```
// ObserverBugs.java - Soal 12.15: dua bug observer dan perbaikannya.
// Tangki deterjen melaporkan isinya; sebuah tampilan dan alarm sekali-bunyi mendengarkan.
import java.util.ArrayList;
import java.util.ConcurrentModificationException;
import java.util.List;
import java.util.function.IntConsumer;

public class ObserverBugs {

    /** Subjeknya. 'careful' memilih antara kode yang ber-bug dan kode yang sudah
diperbaiki. */
    static final class Tank {
        private int liters;
        private final boolean careful;
        private final List<IntConsumer> listeners = new ArrayList<>();

        Tank(int liters, boolean careful) {
            this.liters = liters;
            this.careful = careful;
        }

        void addListener(IntConsumer l) {
            listeners.add(l);
        }

        void removeListener(IntConsumer l) {
            listeners.remove(l);
        }

        int liters() {
            return liters;
        }

        void use(int amount) {
            if (careful) {
                liters -= amount;                       // PERBAIKAN 1: ubah keadaan
dahulu,
                for (IntConsumer l : List.copyOf(listeners)) {   // PERBAIKAN 2: lalu beri
tahu sebuah salinan
                    l.accept(liters);
                }
            } else {
                for (IntConsumer l : listeners) {       // BUG 2: list yang hidup
                    l.accept(liters);                   // BUG 1: sebelum perubahan
                }
                liters -= amount;
            }
        }
    }
```

Kode 12.53 (Lanjutan) *ObserverBugs.java: subjek yang memberi tahu sebelum berubah dan menelusuri daftar listener yang hidup, di samping versi yang sudah diperbaiki.*

```
    /** Alarm sekali-bunyi: di bawah 10 liter ia berbunyi sekali lalu berhenti
berlangganan. */
    static final class LowAlarm implements IntConsumer {
        private final Tank tank;
        private final List<String> log;

        LowAlarm(Tank tank, List<String> log) {
            this.tank = tank;
            this.log = log;
        }

        @Override
        public void accept(int level) {
            if (tank.liters() < 10) {
                log.add("ALARM at " + tank.liters() + " l");
                tank.removeListener(this);              // berhenti berlangganan SELAGI
pemberitahuan berlangsung
            }
        }
    }

    static List<String> run(boolean careful) {
        Tank tank = new Tank(14, careful);
        List<String> log = new ArrayList<>();
        int[] shown = {tank.liters()};                  // apa yang ditampilkan layar
        tank.addListener(new LowAlarm(tank, log));
        tank.addListener(level -> shown[0] = level);
        tank.addListener(level -> log.add("logger saw " + level));
        try {
            for (int i = 0; i < 3; i++) {
                tank.use(3);
            }
        } catch (ConcurrentModificationException e) {
            log.add("ConcurrentModificationException");
        }
        log.add("display shows " + shown[0] + " l, tank has " + tank.liters() + " l");
        return log;
    }

    private static int passed = 0;
```

Kode 12.53 (Lanjutan) *ObserverBugs.java: subjek yang memberi tahu sebelum berubah dan menelusuri daftar listener yang hidup, di samping versi yang sudah diperbaiki.*

```
    private static void check(String name, Object expected, Object actual) {
        if (!java.util.Objects.equals(expected, actual)) {
            throw new AssertionError(name + ": expected " + expected + " but got " +
actual);
        }
        passed++;
        System.out.println("PASS " + name);
    }

    public static void main(String[] args) {
        System.out.println("buggy tank:");
        List<String> buggy = run(false);
        buggy.forEach(line -> System.out.println("  " + line));
        System.out.println("careful tank:");
        List<String> careful = run(true);
        careful.forEach(line -> System.out.println("  " + line));
        check("buggy: stale display", "display shows 11 l, tank has 8 l",
buggy.get(buggy.size() - 1));
        check("careful: alarm once", 1L, careful.stream().filter(l ->
l.startsWith("ALARM")).count());
        check("careful: display agrees", "display shows 5 l, tank has 5 l",
careful.get(careful.size() - 1));
        System.out.println("All " + passed + " checks passed.");
    }
}
```

```
buggy tank:
  logger saw 14
  logger saw 11
  ALARM at 8 l
  ConcurrentModificationException
  display shows 11 l, tank has 8 l
careful tank:
  logger saw 11
  ALARM at 8 l
  logger saw 8
  logger saw 5
  display shows 5 l, tank has 5 l
PASS buggy: stale display
PASS careful: alarm once
PASS careful: display agrees
All 3 checks passed.
```

Penyelesaian Python. Kode 12.54.

Kode 12.54 `observer_bugs.py`: subjek yang sama dalam Python, tempat *bug* kedua gagal tanpa suara.

```python
# observer_bugs.py - Soal 12.15: dua bug observer dan perbaikannya.
# Tangki deterjen melaporkan isinya; sebuah tampilan dan alarm sekali-bunyi mendengarkan.
import sys
import unittest


class Tank:
    """Subjeknya. 'careful' memilih antara kode yang ber-bug dan kode yang sudah diperbaiki."""

    def __init__(self, liters, careful):
        self._liters = liters
        self._careful = careful
        self._listeners = []

    def add_listener(self, listener):
        self._listeners.append(listener)

    def remove_listener(self, listener):
        self._listeners.remove(listener)

    def liters(self):
        return self._liters

    def use(self, amount):
        if self._careful:
            self._liters -= amount                  # PERBAIKAN 1: ubah keadaan dahulu,
            for listener in list(self._listeners):  # PERBAIKAN 2: lalu beri tahu sebuah salinan
                listener(self._liters)
        else:
            for listener in self._listeners:        # BUG 2: list yang hidup
                listener(self._liters)              # BUG 1: sebelum perubahan
            self._liters -= amount


class LowAlarm:
    """Alarm sekali-bunyi: di bawah 10 liter ia berbunyi sekali lalu berhenti berlangganan."""

    def __init__(self, tank, log):
        self._tank, self._log = tank, log

    def __call__(self, level):
        if self._tank.liters() < 10:
            self._log.append(f"ALARM at {self._tank.liters()} l")
            self._tank.remove_listener(self)        # berhenti berlangganan SELAGI pemberitahuan berlangsung
```

Kode 12.54 (Lanjutan) *observer_bugs.py: subjek yang sama dalam Python, tempat bug kedua gagal tanpa suara.*

```
def run(careful):
    tank = Tank(14, careful)
    log = []
    shown = [tank.liters()]                          # apa yang ditampilkan layar

    def display(level):
        shown[0] = level

    tank.add_listener(LowAlarm(tank, log))
    tank.add_listener(display)
    tank.add_listener(lambda level: log.append(f"logger saw {level}"))
    for _ in range(3):
        tank.use(3)
    log.append(f"display shows {shown[0]} l, tank has {tank.liters()} l")
    return log


class TestTank(unittest.TestCase):
    def test_buggy_display_is_stale(self):
        self.assertEqual("display shows 11 l, tank has 5 l", run(False)[-1])

    def test_careful(self):
        log = run(True)
        self.assertEqual(1, sum(1 for line in log if line.startswith("ALARM")))
        self.assertEqual("display shows 5 l, tank has 5 l", log[-1])


if __name__ == "__main__":
    print("buggy tank:")
    for line in run(False):
        print("  " + line)
    print("careful tank:")
    for line in run(True):
        print("  " + line)
    sys.stdout.flush()
    unittest.main(verbosity=2)
```

```
buggy tank:
  logger saw 14
  logger saw 11
  ALARM at 8 l
  logger saw 8
  display shows 11 l, tank has 5 l
careful tank:
  logger saw 11
  ALARM at 8 l
  logger saw 8
  logger saw 5
  display shows 5 l, tank has 5 l
test_buggy_display_is_stale (__main__.TestTank.test_buggy_display_is_stale) ... ok
test_careful (__main__.TestTank.test_careful) ... ok

----------------------------------------------------------------------
Ran 2 tests in 0.000s

OK
```

Baris demi baris. Cabang yang ber-*bug* memberi tahu lebih dulu, sehingga pencatat mencatat 14 dan 11, isi sebelum pencucian. Pada cucian ketiga isi lamanya 8, dan alarm berbunyi lalu mengeluarkan dirinya selagi gelung menelusuri daftarnya. `ArrayList` di Java mendeteksi perubahan itu dan langkah gelung berikutnya melempar `ConcurrentModificationException`: tampilan terlewat pada cucian itu, dan karena eksepsinya meninggalkan `use` sebelum

pengurangan, isinya bahkan tidak berkurang: `display shows 11 l, tank has 8 l`. `for` di Python cukup melanjutkan ke indeks berikutnya; alarm berada di indeks 0, tampilan bergeser ke sana, dan gelung, yang kini di indeks 1, *melewatinya*. Tidak ada galat, hanya tampilan yang diam-diam basi: `display shows 11 l, tank has 5 l`. Versi *bug* yang diam justru yang lebih berbahaya. Cabang yang cermat mengurangi lebih dulu lalu menelusuri `List.copyOf(listeners)` atau `list(self._listeners)`, sehingga setiap *listener* melihat isi yang baru dan alarm dapat keluar dari daftar yang asli dengan aman. `LowAlarm` membaca `tank.liters()`, bukan argumennya, seperti yang sering dilakukan *listener* sungguhan, dan justru karena itulah "ubah dahulu" penting.

Penyelesaian keliru yang menggoda. Perbaikan yang menggoda untuk eksepsi itu adalah menangkapnya, atau menggelung dengan indeks, `for (int i = 0; i < listeners.size(); i++)`. Eksepsinya hilang dan program berperilaku seperti versi Python, melewatkan seorang *listener* tanpa sepatah kata: pesan galat itu tidak pernah menjadi masalahnya, melainkan laporan jujur tentang masalahnya. `CopyOnWriteArrayList` di Java, yang menyalin lariknya pada setiap perubahan sehingga penelusuran selalu berjalan atas potret yang stabil, memang dibuat untuk daftar *listener*, yang jauh lebih sering dibaca daripada diubah [5].

> **UJI DIRI.** Pada tangki yang cermat, seorang *listener menambahkan listener* lain selagi diberi tahu. Apakah *listener* baru itu dipanggil dalam pemberitahuan yang sama?
>
> *Jawaban.* Tidak. Gelung menelusuri salinan yang diambil sebelum pemanggilan pertama, dan *listener* baru tidak ada di dalamnya. Ia akan dipanggil mulai pemberitahuan berikutnya. Biasanya memang itulah yang diinginkan: *listener* yang didaftarkan selama sebuah peristiwa sebaiknya melihat peristiwa *berikutnya*, bukan peristiwa saat ini yang setengah jadi.

12.9.3 Soal 12.16 — Tombol-tombol yang dibuat dalam gelung

> **SOAL.** A keypad has twelve buttons labeled by the characters of `"123456789*0#"`, and each button must record its own character when it is clicked. Create the buttons and their listeners in one loop, in Java and in Python. Show three correct ways to give each listener its own key, and explain the version in which clicking 4, 6, 6, 3 records `####`.
>
> *Dengan kata lain:* papan tombol memiliki dua belas tombol berlabel karakter-karakter `"123456789*0#"`, dan setiap tombol harus mencatat karakternya sendiri bila diklik. Buat tombol dan *listener*-nya dalam satu gelung, di Java dan di Python; tunjukkan tiga cara benar memberi setiap *listener* tombolnya sendiri, dan jelaskan versi yang mencatat `####` ketika 4, 6, 6, 3 diklik.

Dengan bahasa kita sendiri. *Listener* berjalan *belakangan*, ketika tombolnya diklik, dan pada saat itu harus mengetahui tombolnya; saat itu variabel gelung sudah berisi hal lain. Setiap *listener* harus *membawa* tombolnya sendiri.

Rancangan. Lambda atau kelas dalam yang memakai variabel dari metode di sekelilingnya dikatakan *menangkap* (*capture*) variabel itu; kode beserta variabel tangkapannya membentuk *closure*. Kedua bahasa menangkap secara berbeda:

- **Java menangkap nilai.** Lambda atau kelas anonim hanya boleh memakai variabel lokal yang *effectively final* [15], yaitu tidak pernah ditugasi ulang, dan menyimpan salinan nilainya; kompilator menolak *bug* klasiknya.
- **Python menangkap variabel.** Sebuah `lambda` mencari nilai variabelnya ketika ia *dijalankan*; sesudah gelung, setiap lambda yang dibuat di dalamnya melihat nilai terakhir, jebakan yang terkenal [16].

Pola-pola yang benar: di Java, variabel baru pada setiap putaran (dideklarasikan di dalam gelung, atau variabel *for-each*), ditangkap oleh lambda, kelas anonim, atau kelas bernama dengan sebuah *field*; di Python, apa saja yang mengevaluasi tombolnya *ketika fungsi dibuat*: argumen bawaan, fungsi pabrik, atau `functools.partial`.

Penyelesaian Java. **Kode 12.55** membangun empat papan tombol: tiga yang benar dan satu yang ber-*bug*.

Kode 12.55 `KeypadClosures.java`: *listener* yang dibuat dalam gelung dengan lambda, kelas dalam anonim, dan kelas bernama, serta satu versi yang *listener*-nya berbagi satu sel yang dapat berubah.

```
// KeypadClosures.java - Soal 12.16: dua belas tombol dibuat dalam gelung. Setiap
// listener harus mengingat tombolnya SENDIRI. Tiga listener yang benar, satu yang tidak.
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.ArrayList;
import java.util.List;

import javax.swing.JButton;

public class KeypadClosures {

    static final String KEYS = "123456789*0#";

    /** Mengeklik tombol-tombol yang diberikan dan mengembalikan apa yang dicatat listener.
*/
    static String click(List<JButton> buttons, String keys, StringBuilder pressed) {
        pressed.setLength(0);
        for (char k : keys.toCharArray()) {
            buttons.get(KEYS.indexOf(k)).doClick();
        }
        return pressed.toString();
    }

    /** Benar 1: lambda yang menangkap variabel baru yang effectively final pada setiap
putaran. */
    static List<JButton> withLambdas(StringBuilder pressed) {
        List<JButton> buttons = new ArrayList<>();
        for (int i = 0; i < KEYS.length(); i++) {
            char key = KEYS.charAt(i);                  // 'key' yang baru pada setiap
putaran
            JButton b = new JButton(String.valueOf(key));
            b.addActionListener(e -> pressed.append(key));
            buttons.add(b);
        }
        return buttons;
    }

    /** Benar 2: kelas dalam anonim; ia pun menyalin nilai yang ditangkap. */
    static List<JButton> withInnerClasses(StringBuilder pressed) {
        List<JButton> buttons = new ArrayList<>();
        for (char key : KEYS.toCharArray()) {           // for-each: 'key' yang baru juga
            JButton b = new JButton(String.valueOf(key));
            b.addActionListener(new ActionListener() {
                @Override
                public void actionPerformed(ActionEvent e) {
                    pressed.append(key);
                }
            });
            buttons.add(b);
```

Kode 12.55 (Lanjutan) *KeypadClosures.java: listener yang dibuat dalam gelung dengan lambda, kelas dalam anonim, dan kelas bernama, serta satu versi yang listener-nya berbagi satu sel yang dapat berubah.*

```
        }
        return buttons;
    }

    /** Benar 3: kelas listener bernama yang menyimpan tombolnya dalam field. */
    static final class KeyListener implements ActionListener {
        private final char key;
        private final StringBuilder pressed;

        KeyListener(char key, StringBuilder pressed) {
            this.key = key;
            this.pressed = pressed;
        }

        @Override
        public void actionPerformed(ActionEvent e) {
            pressed.append(key);
        }
    }

    static List<JButton> withNamedClass(StringBuilder pressed) {
        List<JButton> buttons = new ArrayList<>();
        for (char key : KEYS.toCharArray()) {
            JButton b = new JButton(String.valueOf(key));
            b.addActionListener(new KeyListener(key, pressed));
            buttons.add(b);
        }
        return buttons;
    }

    /** Menggoda tetapi salah: semua listener berbagi SATU sel yang dapat berubah, dibaca
saat diklik. */
    static List<JButton> withSharedCell(StringBuilder pressed) {
        List<JButton> buttons = new ArrayList<>();
        char[] current = new char[1];                    // lariknya effectively final...
        for (char key : KEYS.toCharArray()) {
            current[0] = key;                            // ...tetapi isinya tidak
            JButton b = new JButton(String.valueOf(key));
            b.addActionListener(e -> pressed.append(current[0]));
            buttons.add(b);
        }
        return buttons;
    }

    private static int passed = 0;
```

Kode 12.55 (Lanjutan) *KeypadClosures.java: listener yang dibuat dalam gelung dengan lambda, kelas dalam anonim, dan kelas bernama, serta satu versi yang listener-nya berbagi satu sel yang dapat berubah.*

```
    private static void check(String name, Object expected, Object actual) {
        if (!java.util.Objects.equals(expected, actual)) {
            throw new AssertionError(name + ": expected " + expected + " but got " +
actual);
        }
        passed++;
        System.out.println("PASS " + name);
    }

    public static void main(String[] args) {
        StringBuilder pressed = new StringBuilder();
        check("lambdas", "4663", click(withLambdas(pressed), "4663", pressed));
        check("inner classes", "4663", click(withInnerClasses(pressed), "4663", pressed));
        check("named class", "4663", click(withNamedClass(pressed), "4663", pressed));
        System.out.println("shared cell: " + click(withSharedCell(pressed), "4663",
pressed));
        System.out.println("All " + passed + " checks passed.");
    }
}
```

```
PASS lambdas
PASS inner classes
PASS named class
shared cell: ####
All 3 checks passed.
```

Gelung berindeks sama sekali tidak dapat menangkap `i`. Menulis `b.addActionListener(e -> pressed.append(keys.charAt(i)));` di dalam `for (int i = 0; ...; i++)` ditolak kompilator:

```
LoopIndex.java:8: error: local variables referenced from a lambda expression must be final
or effectively final
            b.addActionListener(e -> pressed.append(keys.charAt(i)));
                                                                ^
1 error
```

karena `i++` menugasi ulang `i`: Java mengubah kejutan Python saat program berjalan menjadi galat saat kompilasi.

Penyelesaian Python. Kode 12.56.

Kode 12.56 `keypad_closures.py`: perintah-perintah yang dibuat dalam gelung, satu versi dengan *bug late binding* dan tiga versi yang benar.

```
# keypad_closures.py - Soal 12.16: dua belas perintah tombol dibuat dalam gelung.
# Setiap perintah harus mengingat tombolnya SENDIRI. Satu versi menggoda yang tidak.
import sys
import unittest
from functools import partial

KEYS = "123456789*0#"
pressed = []                                    # apa yang telah dicatat tombol-tombol


def click(commands, keys):
    """Panggil perintah setiap tombol, seperti bila tombolnya diklik; kembalikan apa yang
ditekan."""
    pressed.clear()
    for k in keys:
        commands[KEYS.index(k)]()
    return "".join(pressed)


def wrong_commands():
    """Menggoda tetapi salah: setiap lambda mencari 'key' saat DIJALANKAN, bukan saat
dibuat."""
    return [lambda: pressed.append(key) for key in KEYS]


def with_default_argument():
    """Benar 1: nilai bawaan dievaluasi saat lambda DIBUAT."""
    return [lambda key=key: pressed.append(key) for key in KEYS]


def make_command(key):
    return lambda: pressed.append(key)          # setiap panggilan memiliki 'key' sendiri


def with_factory():
    """Benar 2: fungsi pabrik (make_command dari Soal 12.10)."""
    return [make_command(key) for key in KEYS]


def with_partial():
    """Benar 3: functools.partial menyimpan argumen sekarang dan memanggil nanti."""
    return [partial(pressed.append, key) for key in KEYS]
```

Kode 12.56 (Lanjutan) *keypad_closures.py: perintah-perintah yang dibuat dalam gelung, satu versi dengan bug late binding dan tiga versi yang benar.*

```
class TestClosures(unittest.TestCase):
    def test_right_versions(self):
        for make in (with_default_argument, with_factory, with_partial):
            self.assertEqual("4663", click(make(), "4663"))

    def test_wrong_version(self):
        self.assertEqual("####", click(wrong_commands(), "4663"))


if __name__ == "__main__":
    print("wrong commands :", click(wrong_commands(), "4663"))
    print("default arg    :", click(with_default_argument(), "4663"))
    sys.stdout.flush()
    unittest.main(verbosity=2)
```

```
wrong commands : ####
default arg    : 4663
test_right_versions (__main__.TestClosures.test_right_versions) ... ok
test_wrong_version (__main__.TestClosures.test_wrong_version) ... ok

----------------------------------------------------------------------
Ran 2 tests in 0.000s

OK
```

Baris demi baris. Pada **Kode 12.55**, `withLambdas` mendeklarasikan `char key` di dalam badan gelung, dua belas variabel yang masing-masing ditugasi sekali; `withInnerClasses` memakai variabel *for-each*, yang juga baru pada setiap putaran, ditangkap oleh `ActionListener` anonim dengan aturan yang sama (satu-satunya cara menulis *listener* sebelum Java 8); `withNamedClass` membuat penangkapan itu eksplisit dalam sebuah *field* `private final`, yang juga dilakukan kompilator di balik layar untuk dua cara lainnya. Pada **Kode 12.56**, lambda-lambda `wrong_commands` semuanya merujuk variabel `key` milik *comprehension*, yang sudah bernilai `"#"` ketika lambda-lambda itu dijalankan. `lambda key=key:` mengevaluasi nilai bawaannya ketika lambda dibuat; `make_command(key)` berhasil karena setiap *pemanggilan* membuat `key` lokal yang baru (metode bantu dari Soal 12.10); `partial(pressed.append, key)` menyimpan argumennya untuk nanti.

Penyelesaian keliru yang menggoda. Pemrogram Java terkadang "memperbaiki" galat *effectively final* dengan larik satu elemen: referensinya final, isinya tidak. `withSharedCell` menulis setiap tombol ke `current[0]` dan setiap *listener* membaca `current[0]` *ketika diklik*, dan saat itu isinya `#`: mengeklik 4, 6, 6, 3 mencatat `####`, *bug* Python yang dimasukkan kembali dengan tangan. Aturan kompilator melindungi program dari keadaan bersama yang dapat berubah; berilah setiap *listener* salinannya *sendiri*.

> **UJI DIRI.** Di Python, `buttons = [tk.Button(root, text=k, command=lambda: press(k)) for k in "0123"]`. Tombol apa yang ditekan setiap tombol, dan perubahan terkecil apa yang memperbaikinya?
>
> *Jawaban.* Keempat tombol menekan `"3"`, nilai terakhir `k`. Perbaikan terkecilnya adalah argumen bawaan: `command=lambda k=k: press(k)`.

12.10 Perbandingan Soal-Soal dalam Bab Ini

Tabel 12.11 menjajarkan keenam belas soal. Seperti di Bab 8, bacalah kolom terakhir dari atas ke bawah: setiap penyelesaian keliru yang menggoda lolos dari uji yang jelas-jelas terpikir, lalu gagal pada satu masukan tertentu atau satu urutan peristiwa tertentu.

Tabel 12.11 Keenam belas soal dalam bab ini: apa yang dilatih masing-masing dan kasus yang keliru ditangani penyelesaian keliru yang menggoda.

Soal	Topik	Gagasan utama	Yang terlewat oleh penyelesaian keliru
12.1	string	satu tombol per huruf; `StringBuffer` tumbuh di tempat	tombol empat huruf (s, v, y, z)
12.2	berkas, himpunan	himpunan membuang duplikat; berkas selalu ditutup	huruf kapital dan spasi dalam berkas
12.3	baris perintah	`args` lawan `sys.argv`; keluaran terurut	larik tercetak sebagai `[L...@hash`
12.4	*list* terurut	urutan numerik tanpa konversi; pencarian biner ditambah penelusuran	beberapa kata per signature; signature 17 angka
12.5	antarmuka	kontrak dari kebutuhan pemakai	membocorkan *list* milik implementasi pertama
12.6	pengukuran waktu	bangun sekali, lalu cari berkali-kali	mengukur waktu mulai, bukan pencarian
12.7	*hash map*	temukan-atau-buat laci; hasil hanya-baca	`put` menimpa; pemanggil mengosongkan laci
12.8	*trie*	simpan setiap kata di sepanjang lintasannya; potong hasilnya	kata hanya disimpan di ujung: tanpa prefiks
12.9	pemakaian antarmuka	satu baris memilih implementasi	variabel bertipe kelas konkret
12.10	MVC	keadaan di model, dihitung ulang bukan digandakan	area teks sebagai data; signature disimpan dua kali
12.11	koleksi	pilih menurut operasi; ukur	`List.contains` dalam gelung: $O(n^2)$
12.12	generik	parameter tipe berbatas; `? extends`	`List<Towel>` bukan `List<Weighed>`
12.13	*comparator*	setiap kunci menyatakan arahnya sendiri	`.reversed()` di ujung; pengurangan dengan `(int)`
12.14	MVC	dua view dari satu model; diuji tanpa jendela	view kedua terlupa
12.15	observer	ubah dahulu, beri tahu sebuah salinan	view basi; *listener* terlewat
12.16	*closure*	setiap *listener* membawa tombolnya sendiri	*late binding*: setiap tombol menekan `#`

12.11 Profitina dalam Praktik

Profitina, platform visibilitas AI (GEO) yang beroperasi dalam produksi di app.profitina.com, mengukur seberapa siap situs web sebuah merek untuk dikutip oleh mesin jawaban AI dan membantu memperbaiki apa yang ditemukannya. Fitur-fitur publiknya memperlihatkan gagasan bab ini bekerja dalam produk nyata.

Koleksi yang tepat untuk pertanyaan yang sering. Perayap (*crawler*) menjumpai halaman yang sama dengan banyak ejaan alamat (garis miring di ujung, parameter pelacak, huruf besar-kecil). “Apakah halaman ini sudah pernah kulihat?” adalah pertanyaan keanggotaan dari Soal 12.11, yang diajukan untuk setiap tautan di setiap halaman, sehingga tempatnya di *hash set* berisi alamat kanonis, tidak pernah di *list*. Tren skor mingguan per merek adalah kebalikannya: deret yang dibaca menurut urutan waktu dan diekspor ke CSV, tugas sebuah struktur terurut.

Satu antarmuka, banyak implementasi. Pemindai memeriksa sebuah situs melalui *sinyal*, seperti seberapa mudah isinya diekstraksi dan apakah situs itu menerbitkan data terstruktur,

masing-masing diberi skor dan dijelaskan secara mandiri di balik satu antarmuka bersama: bentuk yang sama dengan `Dictionary` dan ketiga implementasinya. Sinyal baru berarti kelas baru, bukan suntingan pada gelungnya.

Model, view, dan observer. Dasbor mengumpulkan notifikasi, kanal yang diikuti, peringkat, dan aktivitas, dan setiap kanal menampilkan penghitung "N baru" untuk unggahan tingkat teratas: view dari sebuah model, seperti label-label pada Soal 12.14. Umpan komunitas waktu-nyata mendorong perubahan ke setiap halaman yang terbuka dengan *Server-Sent Events*, yaitu pola observer pada Soal 12.15 yang melintasi jaringan, dan *optimistic posting* (pengunggahan optimistis) mengikuti aturan pertamanya: kotak tulis langsung kosong dan unggahan langsung muncul karena model lokal diubah lebih dulu; bila pengiriman gagal, model dipulihkan dan teksnya kembali ke kotak tulis, sehingga tidak ada view yang menampilkan sesuatu yang tidak terjadi.

Sistem menyarankan; manusia memilih. Teks prediktif menawarkan kandidat dan pengguna memastikan salah satunya. Fitur-fitur AI Profitina mengikuti aturan yang sama: draf dan skor adalah saran beserta penjelasannya, tidak pernah jaminan. Versi yang dapat dijalankan dari setiap gagasan di sini adalah kode Profitina Laundry dalam bab ini.

12.12 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya

Memilih koleksi karena kebiasaan. *List* adalah koleksi pertama yang dipelajari semua orang, dan ia pilihan yang salah untuk "apakah ini ada?" yang ditanyakan berkali-kali. Tuliskan dulu operasi-operasi yang sering (Subbab 12.2) dan cocokkan masing-masing dengan sebuah struktur (Soal 12.4, 12.7, 12.11).

Membandingkan bilangan sebagai string, atau string sebagai bilangan. `"22" < "3"` sebagai string; `Double.parseDouble` kehilangan angka lewat dari enam belas. Signature dibandingkan menurut panjang lalu sebagai string, Persamaan (12.4) (Soal 12.4).

Hanya menyimpan temuan pertama pencarian biner. Pencarian berhenti pada *salah satu* kecocokan; kumpulkan tetangganya (Soal 12.4).

Menyerahkan koleksi milik sebuah objek. Pemanggil yang menerima himpunan milik *map* dapat mengubah *map* itu. Kembalikan *view* hanya-baca (`Collections.unmodifiableSet`, `List.copyOf`) atau salinan (`frozenset`, `tuple`) (Soal 12.7, 12.10).

Mengganti alih-alih menambah pada *map* bernilai jamak. `put(key, newSet)` untuk setiap nilai hanya menyisakan nilai terakhir. Pakailah `computeIfAbsent(...).add(...)` di Java dan `setdefault(...).add(...)` di Python (Soal 12.7).

Mencetak himpunan dan mengharapkan urutan. *Hash set* tidak memiliki urutan yang bermakna, dan Python bahkan mengubah urutan himpunan string di antara eksekusi. Urutkan sebelum mencetak, atau pakai koleksi yang terurut (Soal 12.3, 12.11).

Memakai tipe mentah atau konkret di tempat antarmuka sudah cukup. `DictionaryListImpl d` mengikat kode pada satu implementasi; `Dictionary d` tidak. `List<Weighed>` menolak `List<Towel>`; `List<? extends Weighed>` menerimanya (Soal 12.9, 12.12).

Membalik seluruh rantai *comparator*, atau mengurangkan untuk membandingkan. Pasang `reverseOrder()` pada satu kunci yang memerlukannya; pakailah `Integer.compare` dan `comparingDouble` alih-alih pengurangan (Soal 12.13).

Menyimpan keadaan di view. Label atau area teks adalah untuk menampilkan, bukan menyimpan. Taruh keadaan di dalam model dengan metode-metode biasa dan ujilah tanpa jendela (Soal 12.10, 12.14).

Memberi tahu sebelum mengubah, atau menelusuri daftar *listener* yang hidup. Ubah keadaan dahulu, lalu beri tahu sebuah salinan daftar *listener* (Soal 12.15).

Menangkap variabel gelung yang terus berubah. Di Python, ikat nilainya dengan argumen bawaan, fungsi pabrik, atau `partial`; di Java, beri setiap *listener* variabel final atau *field*-nya sendiri, dan jangan pernah berbagi sel yang dapat berubah (Soal 12.16).

Mengukur hal yang salah. Satu pengukuran kecil hanya mengukur jamnya, dan `time` pada masukan kecil hanya mengukur waktu mulai program. Ukurlah fase yang membedakan, dengan data besar, beberapa kali (Soal 12.6).

12.13 Ringkasan Bab

- Soal tentang koleksi adalah soal tentang operasi: daftarkan operasi beserta frekuensinya, cocokkan setiap operasi yang sering dengan sebuah struktur, sembunyikan pilihan itu di balik antarmuka, taruh keadaan di dalam model, uji model tanpa jendela, dan pastikan dengan pengukuran (Subbab 12.2, **Tabel 12.1**).
- Signature sebuah kata, Persamaan (12.1), dihitung dengan tabel atau *switch* dan dibangun dalam `StringBuffer` atau *list* yang digabung sekali, karena penyambungan string berulang menyalin $n(n+1)/2$ karakter, Persamaan (12.2) (Soal 12.1).
- Prototipe membaca berkas untuk setiap pertanyaan, Persamaan (12.3); ia menjadi tolok ukur kebenaran dan kecepatan (Soal 12.2, 12.3).
- *List* terurut berisi pasangan `WordSig`, diurutkan menurut Persamaan (12.4), menjawab dengan pencarian biner paling banyak Persamaan (12.5) tengokan ditambah penelusuran ke tetangga, Persamaan (12.6); antarmuka `Dictionary` hanya berisi apa yang diperlukan pemakainya (Soal 12.4, 12.5).
- Pengukuran waktu harus memisahkan pembangunan dari pencarian; untuk banyak pencarian pada kamus besar, prototipe puluhan ribu kali lebih lambat daripada *list* (Soal 12.6, **Tabel 12.5**).
- `HashMap` dari signature ke himpunan kata mencari dan menyisipkan dalam waktu konstan harapan, Persamaan (12.8); *trie* menyimpan setiap kata di sepanjang lintasannya, Persamaan (12.9), dan menjadi satu-satunya dari keempatnya yang menjawab prefiks (Soal 12.7, 12.8, **Tabel 12.7**, **Gambar 12.4**).
- GUI dalam MVC menyimpan keadaannya di model yang menyimpan setiap fakta sekali; view hanya menampilkan, controller hanya meneruskan; mode `--selftest` dan kamus tiruan memungkinkan segala sesuatu kecuali pikselnya diuji (Soal 12.10).
- Koleksi dipilih menurut operasi dan dipastikan dengan pengukuran (12.11); kelas generik memakai parameter tipe berbatas dan *wildcard*, karena tipe generik bersifat invarian (12.12); rantai *comparator* menyatakan arah setiap kunci (12.13); dua view dari satu model tidak mungkin tidak sepakat, Persamaan (12.10) (12.14); observer memerlukan "ubah dahulu, beri tahu sebuah salinan" (12.15); *listener* yang dibuat dalam gelung harus menangkap nilainya sendiri (12.16).

12.14 Soal Latihan

Latihan 12.1 sampai 12.5 meminta kode; penyelesaiannya berupa program di Subbab 12.15. Latihan lainnya meminta penalaran dan dijawab dengan kata-kata.

1. **(Kesamaan dalam *hash set*.)** Masukkan dua objek `new WordSig("4663", "home")` ke dalam sebuah `HashSet`. Berapa elemen yang dimiliki himpunan itu, dan mengapa? Tulislah kelas `Pair` dengan `equals` dan `hashCode` (Python: `__eq__` dan `__hash__`) sehingga himpunannya berisi satu elemen.
2. **(Tabel alih-alih *switch*.)** Tulislah `tableSignature(String word)` yang memakai `Map<Character, Character>` yang dibangun sekali, dalam sebuah *static initializer*, alih-alih `keyOf`. Ujilah bahwa hasilnya sama dengan `wordToSignature` untuk ke-26 huruf, untuk `"Don't"`, dan untuk `"3rd"`.
3. **(Prefiks dari *map* terurut.)** Simpanlah kamus dalam `TreeMap<String, Set<String>>` berkunci signature dengan urutan string biasa, lalu tulis `signatureToWords` yang mengembalikan setiap kata *utuh* yang signature-nya *diawali* signature yang diberikan, dengan memakai `subMap`. Apa hasilnya untuk 9675? Mengapa batas atas rentangnya diakhiri karakter `:`?
4. **(Membalik *map*.)** Tulislah metode generik `<K, V> Map<V, Set<K>> invert(Map<K, V> m)` yang mengubah "kunci ke nilai" menjadi "nilai ke himpunan kunci". Terapkan pada *map* dari lima kata ke signature-nya, lalu carilah 5477.
5. **(Yang terpanjang dahulu.)** Tulislah `Comparator<String>` (Python: fungsi kunci) yang mengurutkan kata menurut panjang, yang terpanjang lebih dulu, dan kata-kata yang sama panjang menurut abjad. Urutkan `go, home, a, wash, laundry, is`.
6. **(Menghitung tengokan.)** Untuk kamus 400.000 kata, paling banyak berapa kartu yang diperiksa pencarian biner pada Soal 12.4? Berapa simpul yang dikunjungi pohon pada Soal 12.8 untuk menemukan simpul 96753? Berapa laci yang dibuka *map* pada Soal 12.7?
7. **(Memori pohon.)** Berapa referensi kata yang disimpan `DictionaryTreeImpl` untuk `words.txt`, dan untuk kamus rekaan 200.000 kata? Pakailah Persamaan (12.9); angka-angka yang Anda perlukan ada di Subbab 12.6.2.
8. **(Lebih sedikit pembaruan.)** `DictionaryModel.press` memberi tahu *listener* setelah setiap tekanan, bahkan bila tidak ada yang berubah. Jelaskan cara memberi tahu hanya bila teks yang ditampilkan berubah. Pada **Kode 12.39**, model menerima 16 tekanan; berapa pemberitahuan yang akan dikirim model yang diperbaiki?
9. **(Tombol tanda baca.)** Pemilik Profitina Laundry ingin tombol 1 mengetik titik sesudah kata saat ini. Kelas GUI mana dari keempatnya yang berubah, dan bagaimana? Apakah aturan "tidak ada spasi atau signature di dalam model" mengizinkannya?
10. **(Adu cepat Anda sendiri.)** Bangkitkan kamus 400.000 kata dengan `MakeDictionary 400000 big400.txt` dan jalankan `TimingComparison` padanya. Sebelum menjalankan, ramalkan untuk setiap baris dari keempatnya apakah waktu per pencarian kira-kira tetap, sedikit bertambah, atau berlipat dua dibandingkan eksekusi 200.000 kata. Lalu bandingkan dengan pengukuran Anda dan dengan **Gambar 12.4**.

12.15 Penyelesaian Soal Latihan

Latihan 1 sampai 5. Kode 12.57 dan **Kode 12.58** memuat semua kode yang diminta, diperiksa terhadap jawaban-jawaban di bawah. Berkas Java-nya termasuk paket `predictive`, karena

memakai `WordSig` dan `PredictivePrototype`; jalankan dengan `java predictive.ExerciseSolutions` dari folder `java`.

Kode 12.57 `ExerciseSolutions.java`: pasangan dengan `equals` dan `hashCode`, signature berbasis tabel, kamus `TreeMap` dengan rentang prefiks, `invert` generik, dan sebuah *comparator*, beserta pemeriksaannya.

```
// ExerciseSolutions.java - penyelesaian soal latihan pemrograman Bab 12.
package predictive;

import java.nio.file.Files;
import java.nio.file.Path;
import java.util.ArrayList;
import java.util.Comparator;
import java.util.HashMap;
import java.util.HashSet;
import java.util.List;
import java.util.Map;
import java.util.Objects;
import java.util.Set;
import java.util.TreeMap;
import java.util.TreeSet;

public class ExerciseSolutions {

    /** Latihan 12.1: pasangan dengan equals dan hashCode, sehingga HashSet mengenali
duplikat. */
    static final class Pair {
        private final String signature;
        private final String word;

        Pair(String signature, String word) {
            this.signature = signature;
            this.word = word;
        }

        @Override
        public boolean equals(Object o) {
            if (this == o) {
                return true;
            }
            if (!(o instanceof Pair other)) {
                return false;
            }
            return signature.equals(other.signature) && word.equals(other.word);
        }

        @Override
        public int hashCode() {
            return Objects.hash(signature, word);    // pasangan sama, kode hash sama
        }
    }
```

Kode 12.57 (Lanjutan) *ExerciseSolutions.java: pasangan dengan equals dan hashCode, signature berbasis tabel, kamus TreeMap dengan rentang prefiks, invert generik, dan sebuah comparator, beserta pemeriksaannya.*

```
    /** Latihan 12.2: tabel dibangun sekali, dipakai untuk setiap huruf. */
    private static final Map<Character, Character> KEY_OF = new HashMap<>();

    static {
        String[] letters = {"abc", "def", "ghi", "jkl", "mno", "pqrs", "tuv", "wxyz"};
        for (int k = 0; k < letters.length; k++) {
            for (char c : letters[k].toCharArray()) {
                KEY_OF.put(c, (char) ('2' + k));
            }
        }
    }

    static String tableSignature(String word) {
        StringBuilder sb = new StringBuilder();
        for (char c : word.toLowerCase().toCharArray()) {
            sb.append(KEY_OF.getOrDefault(c, ' '));
        }
        return sb.toString();
    }

    /** Latihan 12.3: map terurut menjawab "setiap signature yang diawali s". */
    static final class SortedMapDictionary implements Dictionary {
        private final TreeMap<String, Set<String>> map = new TreeMap<>();

        SortedMapDictionary(List<String> words) {
            for (String w : words) {
                map.computeIfAbsent(PredictivePrototype.wordToSignature(w), k -> new
HashSet<>()).add(w);
            }
        }

        /** Kata utuh yang signature-nya diawali signature yang diberikan; ':' sesudah '9'.
*/
        @Override
        public Set<String> signatureToWords(String signature) {
            Set<String> result = new HashSet<>();
            for (Set<String> words : map.subMap(signature, signature + ":").values()) {
                result.addAll(words);
            }
            return result;
        }
    }
```

Kode 12.57 (Lanjutan) *ExerciseSolutions.java: pasangan dengan equals dan hashCode, signature berbasis tabel, kamus TreeMap dengan rentang prefiks, invert generik, dan sebuah comparator, beserta pemeriksaannya.*

```
    /** Latihan 12.4: ubah kunci -> nilai menjadi nilai -> himpunan kunci. */
    static <K, V> Map<V, Set<K>> invert(Map<K, V> m) {
        Map<V, Set<K>> result = new HashMap<>();
        for (Map.Entry<K, V> e : m.entrySet()) {
            result.computeIfAbsent(e.getValue(), v -> new HashSet<>()).add(e.getKey());
        }
        return result;
    }

    /** Latihan 12.5: kata yang lebih panjang dahulu, panjang sama menurut abjad. */
    static final Comparator<String> LONGEST_FIRST =

Comparator.comparingInt(String::length).reversed().thenComparing(Comparator.naturalOrder())
;

    private static int passed = 0;

    private static void check(String name, Object expected, Object actual) {
        if (!Objects.equals(expected, actual)) {
            throw new AssertionError(name + ": expected " + expected + " but got " +
actual);
        }
        passed++;
        System.out.println("PASS " + name);
    }

    public static void main(String[] args) throws Exception {
        Set<WordSig> plain = new HashSet<>(List.of(new WordSig("4663", "home"), new
WordSig("4663", "home")));
        Set<Pair> pairs = new HashSet<>(List.of(new Pair("4663", "home"), new Pair("4663",
"home")));
        check("12.1 WordSig duplicates kept", 2, plain.size());
        check("12.1 Pair duplicates removed", 1, pairs.size());

        boolean same = true;
        for (char c = 'a'; c <= 'z'; c++) {
            String s = String.valueOf(c);
            same &= tableSignature(s).equals(PredictivePrototype.wordToSignature(s));
        }
        check("12.2 all letters agree", true, same);
        check("12.2 other characters", "366 8", tableSignature("Don't"));
        check("12.2 digits", " 73", tableSignature("3rd"));
```

Kode 12.57 (Lanjutan) *ExerciseSolutions.java: pasangan dengan equals dan hashCode, signature berbasis tabel, kamus TreeMap dengan rentang prefiks, invert generik, dan sebuah comparator, beserta pemeriksaannya.*

```
        List<String> words = new ArrayList<>();
        for (String line : Files.readAllLines(Path.of("words.txt"))) {
            String w = line.trim().toLowerCase();
            if (PredictivePrototype.isValidWord(w)) {
                words.add(w);
            }
        }
        Dictionary sorted = new SortedMapDictionary(words);
        check("12.3 prefix 9675", "[work, worked, worker, works, world, worlds]",
                new TreeSet<>(sorted.signatureToWords("9675")).toString());
        check("12.3 whole words: goods", true,
sorted.signatureToWords("4663").contains("goods"));

        Map<String, String> wordToSig = new HashMap<>();
        for (String w : List.of("gone", "home", "kiss", "lips", "cat")) {
            wordToSig.put(w, PredictivePrototype.wordToSignature(w));
        }
        Map<String, Set<String>> sigToWords = invert(wordToSig);
        check("12.4 invert", "[kiss, lips]", new
TreeSet<>(sigToWords.get("5477")).toString());

        List<String> list = new ArrayList<>(List.of("go", "home", "a", "wash", "laundry",
"is"));
        list.sort(LONGEST_FIRST);
        check("12.5 longest first", "[laundry, home, wash, go, is, a]", list.toString());
        System.out.println("All " + passed + " checks passed.");
    }
}
```

```
PASS 12.1 WordSig duplicates kept
PASS 12.1 Pair duplicates removed
PASS 12.2 all letters agree
PASS 12.2 other characters
PASS 12.2 digits
PASS 12.3 prefix 9675
PASS 12.3 whole words: goods
PASS 12.4 invert
PASS 12.5 longest first
All 9 checks passed.
```

Kode 12.58 `exercise_solutions.py`: penyelesaian yang sama dalam Python; pencarian prefiksnya memakai dua panggilan `bisect_left` pada signature yang terurut.

```
# exercise_solutions.py - penyelesaian soal latihan pemrograman Bab 12.
import unittest
from bisect import bisect_left

from predictive.dictionary import Dictionary
from predictive.prototype import is_valid_word, word_to_signature
from predictive.word_sig import WordSig


class Pair:
    """Latihan 12.1: pasangan dengan __eq__ dan __hash__, sehingga set mengenali
duplikat."""

    def __init__(self, signature, word):
        self.signature, self.word = signature, word

    def __eq__(self, other):
        if not isinstance(other, Pair):
            return NotImplemented
        return (self.signature, self.word) == (other.signature, other.word)

    def __hash__(self):
        return hash((self.signature, self.word))     # pasangan sama, hash sama


# Latihan 12.2: tabelnya, dibangun sekali dengan dict comprehension.
_KEY = {letter: str(k + 2)
        for k, group in enumerate(["abc", "def", "ghi", "jkl", "mno", "pqrs", "tuv",
"wxyz"])
        for letter in group}


def table_signature(word):
    return "".join(_KEY.get(c, " ") for c in word.lower())


class SortedKeysDictionary(Dictionary):
    """Latihan 12.3: signature terurut menjawab "setiap signature yang diawali s"."""

    def __init__(self, words):
        self._map = {}
        for w in words:
            self._map.setdefault(word_to_signature(w), set()).add(w)
        self._keys = sorted(self._map)               # urutan string biasa
```

Kode 12.58 (Lanjutan) *exercise_solutions.py: penyelesaian yang sama dalam Python; pencarian prefiksnya memakai dua panggilan bisect_left pada signature yang terurut.*

```
    def signature_to_words(self, signature):
        """Kata utuh yang signature-nya diawali signature yang diberikan; ':' sesudah
'9'."""
        lo = bisect_left(self._keys, signature)
        hi = bisect_left(self._keys, signature + ":")
        return set().union(*(self._map[k] for k in self._keys[lo:hi]))


def invert(m):
    """Latihan 12.4: ubah kunci -> nilai menjadi nilai -> himpunan kunci."""
    result = {}
    for k, v in m.items():
        result.setdefault(v, set()).add(k)
    return result


def longest_first(words):
    """Latihan 12.5: kata yang lebih panjang dahulu, panjang sama menurut abjad."""
    return sorted(words, key=lambda w: (-len(w), w))


class TestExercises(unittest.TestCase):
    def test_12_1_equality(self):
        self.assertEqual(2, len({WordSig("4663", "home"), WordSig("4663", "home")}))
        self.assertEqual(1, len({Pair("4663", "home"), Pair("4663", "home")}))

    def test_12_2_table(self):
        for c in "abcdefghijklmnopqrstuvwxyz":
            self.assertEqual(word_to_signature(c), table_signature(c))
        self.assertEqual("366 8", table_signature("Don't"))
        self.assertEqual(" 73", table_signature("3rd"))

    def test_12_3_prefixes(self):
        with open("words.txt", encoding="utf-8") as f:
            words = [w for w in (line.strip().lower() for line in f) if is_valid_word(w)]
        d = SortedKeysDictionary(words)
        self.assertEqual(["work", "worked", "worker", "works", "world", "worlds"],
                         sorted(d.signature_to_words("9675")))
        self.assertIn("goods", d.signature_to_words("4663"))

    def test_12_4_invert(self):
        m = {w: word_to_signature(w) for w in ("gone", "home", "kiss", "lips", "cat")}
        self.assertEqual({"kiss", "lips"}, invert(m)["5477"])
```

Kode 12.58 (Lanjutan) *exercise_solutions.py: penyelesaian yang sama dalam Python; pencarian prefiksnya memakai dua panggilan bisect_left pada signature yang terurut.*

```
    def test_12_5_order(self):
        self.assertEqual(["laundry", "home", "wash", "go", "is", "a"],
                         longest_first(["go", "home", "a", "wash", "laundry", "is"]))


if __name__ == "__main__":
    unittest.main(verbosity=2)
```

```
test_12_1_equality (__main__.TestExercises.test_12_1_equality) ... ok
test_12_2_table (__main__.TestExercises.test_12_2_table) ... ok
test_12_3_prefixes (__main__.TestExercises.test_12_3_prefixes) ... ok
test_12_4_invert (__main__.TestExercises.test_12_4_invert) ... ok
test_12_5_order (__main__.TestExercises.test_12_5_order) ... ok

----------------------------------------------------------------------
Ran 5 tests in 0.000s

OK
```

1. Dua elemen. `WordSig` mewarisi `equals` dan `hashCode` dari `Object`, tempat dua objek sama hanya bila keduanya objek yang *sama*. `Pair` meng-*override* keduanya, dan `Objects.hash(signature, word)` memberi pasangan yang sama kode *hash* yang sama, kontrak yang menjadi sandaran `HashSet` [7]; meng-*override* hanya `equals` tetap menghasilkan dua elemen, karena kedua objek akan jatuh ke ember yang berbeda. `__eq__` dan `__hash__` di Python memainkan peran yang sama.
2. *Static initializer*, `static { ... }`, berjalan sekali ketika kelas dimuat dan mengisi `KEY_OF` dari delapan kelompok huruf; `(char) ('2' + k)` adalah angka kelompoknya, dan `getOrDefault(c, ' ')` mengubah `"3rd"` menjadi `" 73"`. Python membangun tabelnya dengan *dict comprehension* berklausa ganda.
3. `[work, worked, worker, works, world, worlds]`, tanpa dipotong. Dalam urutan string, semua string yang diawali `9675` berdampingan, dan `:` adalah karakter sesudah `9`, sehingga `"9675:"` lebih besar daripada semuanya dan lebih kecil daripada signature berikutnya, misalnya `9676`. `subMap(s, s + ":")` mengembalikan tepat rentang itu, dengan awalnya ditemukan dalam $O(\log n)$: prefiks dari *map* terurut, tanpa pohon buatan tangan.
4. `computeIfAbsent(value, ...)` menemukan atau membuat himpunan setiap nilai lalu menambahkan kuncinya: *map* bernilai jamak dari Soal 12.7 dalam bentuk generik. Gone dan home dipetakan ke 4663, kiss dan lips ke 5477, cat ke 228, sehingga pencarian 5477 memberikan `{kiss, lips}`.
5. `[laundry, home, wash, go, is, a]`. `comparingInt(String::length).reversed()` menaruh kata yang lebih panjang lebih dulu, dan `thenComparing(naturalOrder())`, yang ditambahkan *sesudah* `reversed()`, mengurutkan panjang yang sama menurut abjad; kunci Python `(-len(w), w)` berbuat sama.
6. Menurut Persamaan (12.5), $\lceil \log_2 400001 \rceil = 19$ kartu. Pohon memerlukan 5 langkah di bawah akar untuk 96753, berapa pun ukuran kamusnya. *Map* membuka tepat satu laci.
7. Menurut Persamaan (12.9), satu referensi untuk setiap huruf setiap kata yang berbeda: 354 referensi di 120 simpul bukan-akar untuk `words.txt`, dan 1.183.435 untuk 165.929 kata rekaan yang berbeda. Kata-katanya sendiri disimpan sekali; hanya referensinya yang berulang.
8. Simpan teks sebelum perubahan (`String before = getText();`) dan jalankan *listener* hanya bila `!getText().equals(before)`. Tiga tekanan dalam uji itu tidak mengubah apa pun (angka `7` yang ditolak, `0` yang kedua, dan `#` terakhir pada kata kosong), sehingga model yang diperbaiki mengirim 13 pemberitahuan.
9. Controller mengaktifkan tombol 1 dan memberinya label; `press('1')` pada model menyelesaikan kata saat ini, bila ada, lalu menambahkan `"."` ke kata-kata yang selesai, yang digabung `getText` tanpa spasi di depannya. Model tetap tidak menyimpan spasi maupun signature, karena spasi baru ditambahkan ketika teks ditampilkan; view tidak berubah.
10. Menggandakan kamus seharusnya kira-kira menggandakan waktu prototipe (Persamaan (12.3)), tidak mengubah waktu *map* (Persamaan (12.8)), menambah satu perbandingan pada 18 perbandingan *list* (Persamaan (12.6)), dan sedikit menaikkan waktu

pohon, karena lebih banyak kata berbagi setiap simpul (Persamaan (12.9)). Eksekusi di balik **Gambar 12.4** memberikan, dari 200.000 ke 400.000 kata, pertumbuhan prototipe 1,5 kali di Java dan 2,5 kali di Python, *map* di bawah satu mikrodetik, serta *list* dan pohon yang bergerak dalam batas deraunya. Bila *map* Anda berlipat dua atau prototipe Anda tetap datar, curigailah percobaannya lebih dulu.

Daftar Pustaka

[1] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Reading, MA, USA: Addison-Wesley, 1994.

[2] Python Software Foundation, "unittest — Unit testing framework," *The Python Standard Library*, Python 3.14. [Online]. Available: https://docs.python.org/3/library/unittest.html

[3] D. L. Grover, M. T. King, and C. A. Kushler, "Reduced keyboard disambiguating computer," U.S. Patent 5 818 437, Oct. 6, 1998.

[4] Python Software Foundation, "Built-in Types," "bisect — Array bisection algorithm" and "Sorting Techniques," *The Python Standard Library* and *Python HOWTOs*, Python 3.14. [Online]. Available: https://docs.python.org/3/

[5] Oracle, "Java SE 21 API Specification: java.util (Collections, Comparator, HashMap, TreeMap, CopyOnWriteArrayList, Observable)." [Online]. Available: https://docs.oracle.com/en/java/javase/21/docs/api/

[6] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 4th ed. Cambridge, MA, USA: MIT Press, 2022.

[7] J. Bloch, *Effective Java*, 3rd ed. Boston, MA, USA: Addison-Wesley, 2018.

[8] D. E. Knuth, *The Art of Computer Programming, Vol. 2: Seminumerical Algorithms*, 3rd ed. Reading, MA, USA: Addison-Wesley, 1997.

[9] D. E. Knuth, *The Art of Computer Programming, Vol. 3: Sorting and Searching*, 2nd ed. Reading, MA, USA: Addison-Wesley, 1998.

[10] G. E. Krasner and S. T. Pope, "A cookbook for using the model-view controller user interface paradigm in Smalltalk-80," *Journal of Object-Oriented Programming*, vol. 1, no. 3, pp. 26–49, 1988.

[11] Python Software Foundation, "tkinter — Python interface to Tcl/Tk," *The Python Standard Library*, Python 3.14. [Online]. Available: https://docs.python.org/3/library/tkinter.html

[12] Oracle, "The Java Tutorials: Creating a GUI With Swing" and "Concurrency in Swing." [Online]. Available: https://docs.oracle.com/javase/tutorial/uiswing/

[13] M. Naftalin and P. Wadler, *Java Generics and Collections*. Sebastopol, CA, USA: O'Reilly Media, 2006.

[14] G. van Rossum, J. Lehtosalo, and Ł. Langa, "PEP 484 — Type Hints," 2014; and The mypy project, "mypy documentation: Generics and Common issues (variance)." [Online]. Available: https://peps.python.org/pep-0484/ and https://mypy.readthedocs.io/

[15] J. Gosling, B. Joy, G. Steele, G. Bracha, A. Buckley, D. Smith, and G. Bierman, *The Java Language Specification, Java SE 21 Edition*. Oracle America, 2023. [Online]. Available: https://docs.oracle.com/javase/specs/

[16] Python Software Foundation, "Programming FAQ: Why do lambdas defined in a loop with different values all return the same result?" [Online]. Available: https://docs.python.org/3/faq/programming.html

Koleksi, biayanya, serta saran tentang `equals`, `hashCode`, *view* hanya-baca, dan *wildcard* mengikuti dokumentasi API Java [5], Bloch [7], serta Naftalin dan Wadler [13], dengan aturan bahasa untuk generik dan variabel *effectively final* dari spesifikasinya [15]. Pencarian biner dan *hashing* dianalisis dalam Cormen dkk. [6], *trie* dalam Knuth [9], serta pembangkit kongruensial linear dan kelemahan bit rendahnya dalam Knuth [8]. Materi Python bersandar pada dokumentasi pustaka

bakunya [4], [2], [11], FAQ pemrogramannya [16], serta spesifikasi petunjuk tipe dan mypy [14]. Pola observer berasal dari Gamma dkk. [1], Model–View–Controller dari Krasner dan Pope [10], aturan *thread* Swing dari tutorial Java [12], dan teks prediktif dari patennya [3].

Bab 13

Iterator, Imutabilitas, dan Koleksi Lanjutan

Pada akhir bab ini Anda akan sanggup menelusuri koleksi apa pun dengan sebuah *iterator* di Java maupun di Python, dan menjelaskan dengan tepat apa yang dikerjakan `hasNext()`, `next()`, `__iter__`, dan `__next__` pada setiap langkah; menulis koleksi buatan sendiri yang dapat dipakai dalam perulangan `for`, mula-mula dengan tangan, lalu, di Python, dengan sebuah *generator*; menjelaskan mengapa mengubah list selagi menelusurinya melemparkan `ConcurrentModificationException` di Java tetapi diam-diam melompati elemen di Python, dan sebagai gantinya mengubah koleksi dengan aman; mengurutkan objek dalam urutan apa pun yang Anda kehendaki dengan `Comparable`, `Comparator`, dan fungsi kunci Python, serta mengenali perbandingan yang hanya tampak benar; membedakan *alias* dari salinan dangkal, dan salinan dangkal dari salinan dalam; merancang objek *imutabel* (tidak dapat berubah) dari nol — field `final`, koleksi yang tak dapat diubah, *record*, dan salinan defensif di Java; *dataclass* beku, *tuple*, dan `frozenset` di Python — serta menjelaskan mengapa objek semacam itu membuat program lebih sederhana; menerangkan apa itu *objek nilai* (*value object*) dan mengapa kunci yang berubah di dalam `HashMap` atau `dict` menjadi hilang; dan mengganti banyak perulangan dengan *stream* di Java serta dengan *comprehension* dan `itertools` di Python. Profitina Laundry mendapat tipe `Money` yang imutabel, riwayat pesanan yang dapat ditelusuri mulai dari yang terbaru, dan laporan penjualan yang dibangun sebagai sebuah *pipeline*.

13.1 Di Mana Kita Berhenti

Bab 9 memperkenalkan tiga keluarga besar koleksi — list, set, dan map — beserta generik yang membuat sebuah `List<String>` hanya dapat menampung string. Bab 10 menampilkan jendela di layar, Bab 11 memisahkan program menjadi model, *view*, dan *controller*, dan Bab 12 membahas soal-soal yang memadukan semuanya. Sepanjang bab-bab itu kita memakai koleksi dengan cara yang sederhana: menambahkan sesuatu ke dalamnya, mencari sesuatu di dalamnya, dan menelusurinya dengan perulangan `for` tanpa pernah bertanya bagaimana perulangan itu sebenarnya mengunjungi setiap elemen.

Bab ini membuka tutupnya. Paruh pertama membahas **penelusuran**: objek kecil bernama *iterator* yang diam-diam diciptakan oleh setiap perulangan `for` atas sebuah koleksi, apa yang dapat dan tidak dapat dilakukannya, cara menulisnya untuk koleksi buatan kita sendiri, dan apa yang terjadi bila sebuah koleksi berubah di bawah kaki iteratornya. Bagian tengah bab membahas **urutan dan salinan**: cara mengurutkan objek yang tidak memiliki satu urutan yang jelas, dan apa sesungguhnya arti menyalin sebuah list berisi objek. Paruh kedua membahas **imutabilitas**: objek yang tidak dapat berubah setelah dibuat. Sekilas, objek yang tidak dapat berubah terdengar seperti cacat. Pada akhir bab ia akan tampak sebagai salah satu alat paling berguna yang kita miliki, karena objek yang tidak dapat berubah boleh dibagi pakai, disimpan sebagai kunci, diserahkan kepada metode mana pun, dan diberikan kepada *thread* lain tanpa rasa waswas sedikit pun. Kita menutup bab dengan **stream** dan *comprehension*, yang mengungkapkan "ambil pesanan-pesanan besar lalu jumlahkan totalnya" hampir persis seperti kita mengucapkannya.

Binatu kita tumbuh lagi. Pemiliknya menginginkan riwayat setiap pesanan, tampilan pesanan satu pelanggan, dan laporan mingguan, sementara harga tidak boleh berubah karena kekeliruan. Itulah persis jenis masalah yang menjadi alasan diciptakannya alat-alat dalam bab ini.

> **WAWASAN KUNCI:** Sebuah *iterator* memisahkan kegiatan **menelusuri** koleksi dari kegiatan **menyimpannya**; *imutabilitas* memisahkan kegiatan **menciptakan** objek dari kegiatan **memakainya**. Kedua gagasan itu menyederhanakan kode karena alasan yang sama: setiap bagian program perlu mengetahui lebih sedikit hal dan memiliki lebih sedikit cara untuk keliru.

13.2 Menelusuri Koleksi: Iterator

Misalkan seseorang menyerahkan kepada Anda sebuah `Collection<String>`. Anda tahu isinya string, dan Anda tahu berapa banyak (`size()`), tetapi Anda tidak tahu apakah ia sebuah list, sebuah set, atau sesuatu yang sama sekali lain. Bagaimana cara mencetak setiap string di dalamnya?

Seandainya ia sebuah `List`, Anda dapat memakai indeks: `get(0)`, `get(1)`, dan seterusnya sampai `size() - 1`, karena list menyimpan elemennya dalam urutan bernomor. Namun `HashSet` tidak memiliki indeks; tidak ada "elemen nomor 2" dari sebuah set. Set pohon, antrean, atau koleksi yang ditulis orang tahun lalu mungkin menyimpan elemennya dengan cara yang tidak dapat Anda lihat. Kita memerlukan cara untuk berkata "berikan elemen-elemenmu satu per satu, dalam urutan apa pun yang cocok bagimu, sampai habis" yang berlaku untuk *setiap* koleksi. Cara itu adalah iterator.

> **DEFINISI. Iterator** adalah objek yang memungkinkan kita memeriksa setiap elemen sebuah koleksi secara bergiliran, satu per satu, tanpa mengetahui bagaimana koleksi itu menyimpan elemennya. Ia mengingat posisinya sendiri, dapat memberi tahu apakah masih ada elemen, dan dapat menyerahkan elemen berikutnya.

Gagasan ini sudah lama dan diberi nama dalam katalog pola desain yang termasyhur: *pola Iterator* "menyediakan cara untuk mengakses elemen-elemen sebuah objek agregat secara berurutan tanpa menyingkapkan representasi dasarnya" [1]. Akarnya bahkan lebih tua lagi, yaitu bahasa CLU dari tahun 1970-an yang dirancang oleh kelompok Barbara Liskov; buku Java yang ditulisnya kemudian masih mengajarkan cara menspesifikasikan iterator dengan cermat [2]. Setiap koleksi di Java dan di Python menyediakan iterator, dan kedua bahasa membangun perulangan `for` mereka di atasnya.

13.2.1 Protokol Iterator di Java

Di Java, iterator adalah objek apa pun yang mengimplementasikan antarmuka `java.util.Iterator<E>`. Dua metode pokoknya adalah:

- `boolean hasNext()` — masih adakah elemen yang harus dikunjungi?
- `E next()` — kembalikan elemen berikutnya dan bergeserlah melewatinya. Bila tidak ada lagi, lemparkan `NoSuchElementException`.

Metode ketiga, `void remove()`, bersifat opsional; kita akan menjumpainya di Subbab 13.5. Setiap koleksi memiliki metode `iterator()` yang menciptakan iterator baru yang posisinya berada sebelum elemen pertama [3]. **Kode 13.1** memakai metode-metode ini secara langsung.

Kode 13.1 `IteratorBasics.java`: menelusuri koleksi dengan iterator eksplisit, mengubah salinan lokal sebuah elemen, perulangan `for` yang disempurnakan, dan meminta satu elemen terlalu banyak.

```
import java.util.ArrayList;
import java.util.Collection;
import java.util.Iterator;
import java.util.NoSuchElementException;

public class IteratorBasics {
    public static void main(String[] args) {
        // 1. Cetak setiap string dalam koleksi dengan iterator eksplisit.
        Collection<String> crew = new ArrayList<>();
        crew.add("Luffy, Monkey D.");
        crew.add("Zoro, Roronoa");
        crew.add("Robin, Nico");
        Iterator<String> iter = crew.iterator();
        while (iter.hasNext()) {
            System.out.println(iter.next());
        }

        // 2. Mengubah variabel lokal tidak mengubah koleksi.
        System.out.println();
        Collection<String> captain = new ArrayList<>();
        captain.add("Luffy, Monkey D.");
        Iterator<String> it2 = captain.iterator();
        while (it2.hasNext()) {
            String element = it2.next();
            element += " and Zoro, Roronoa";
            System.out.println(element);
        }
        System.out.println("Print captain's content:");
        for (String s : captain) {
            System.out.println(s);
        }

        // 3. Perulangan for yang disempurnakan adalah iterator yang menyamar.
        System.out.println();
        Collection<Integer> c = new ArrayList<>();
        c.add(3);
        c.add(2);
        c.add(4);
        int sum = 0;
        for (Integer i : c) {
            sum += i;
        }
        System.out.println("sum = " + sum);
```

Kode 13.1 (Lanjutan) *IteratorBasics.java: menelusuri koleksi dengan iterator eksplisit, mengubah salinan lokal sebuah elemen, perulangan for yang disempurnakan, dan meminta satu elemen terlalu banyak.*

```
        // 4. Meminta satu elemen terlalu banyak.
        Iterator<Integer> it3 = c.iterator();
        it3.next();
        it3.next();
        it3.next();
        System.out.println("hasNext() after three calls to next(): " + it3.hasNext());
        try {
            it3.next();
        } catch (NoSuchElementException e) {
            System.out.println("fourth next(): " + e.getClass().getSimpleName());
        }
    }
}
```

Kompilasi dan jalankan program ini di folder `java` bab ini dengan `javac IteratorBasics.java` dan `java IteratorBasics`. Program mencetak:

```
Luffy, Monkey D.
Zoro, Roronoa
Robin, Nico

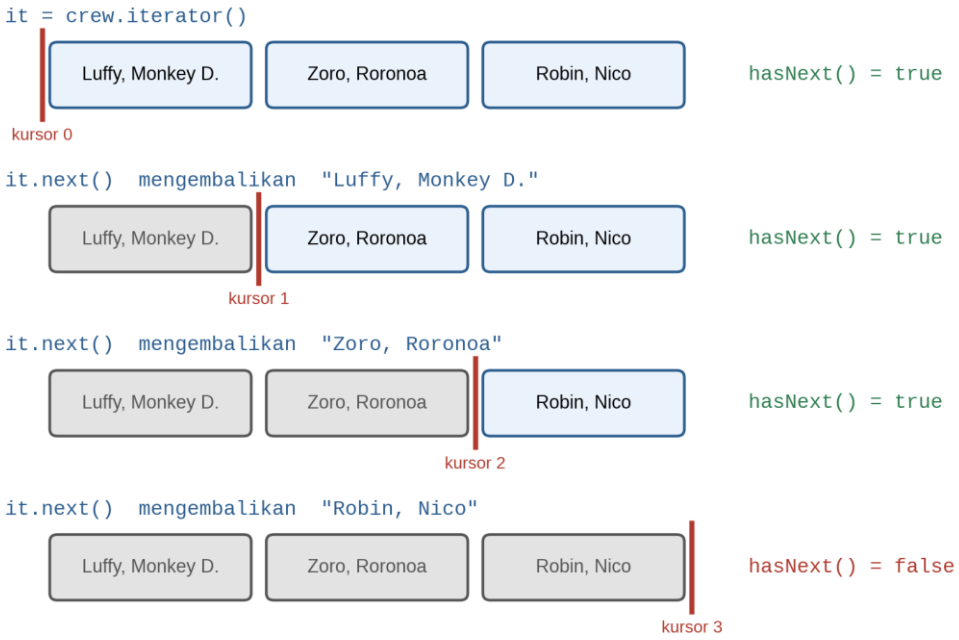
Luffy, Monkey D. and Zoro, Roronoa
Print captain's content:
Luffy, Monkey D.

sum = 9
hasNext() after three calls to next(): false
fourth next(): NoSuchElementException
```

Bacalah blok pertama pelan-pelan, karena setiap iterasi dalam bab ini merupakan variasi darinya.

- `Collection<String> crew = new ArrayList<>();` mendeklarasikan variabel dengan tipe paling umum yang memadai, yakni `Collection`, persis seperti anjuran Bab 9. Mulai titik ini kode hanya tahu bahwa `crew` adalah *suatu* koleksi.
- `Iterator<String> iter = crew.iterator();` meminta sebuah iterator kepada koleksi. Iterator adalah objek tersendiri; koleksinya tidak berubah karena iterator itu dibuat, dan Anda boleh membuat sebanyak yang Anda mau.
- `while (iter.hasNext())` bertanya, sebelum setiap langkah, apakah masih ada elemen lain. Dengan tiga string, jawabannya `true` tiga kali lalu `false`.
- `iter.next()` mengembalikan string berikutnya *dan* menggeser iterator maju. Efek kedua ini mudah terlewat: memanggil `next()` dua kali dalam satu badan perulangan berarti menghabiskan dua elemen.

Posisi iterator paling mudah dibayangkan sebagai sebuah **kursor** yang berada *di antara* elemen-elemen, bukan di atasnya, seperti diperlihatkan **Gambar 13.1**. Kursor iterator yang baru berada sebelum elemen pertama. Setiap panggilan `next()` melompatkan kursor melewati satu elemen dan mengembalikan elemen yang dilompatinya. Setelah kursor melewati elemen terakhir, `hasNext()` menjawab `false`.



Gambar 13.1 Kursor iterator berada di antara elemen. Setiap `next()` melompati satu elemen dan mengembalikannya (kotak abu-abu sudah dikembalikan); setelah lompatan terakhir, `hasNext()` bernilai `false`.

Jika celah-celah itu kita beri nomor $0, 1, \ldots, n$ untuk koleksi berisi n elemen, kursor c mulai di 0, setiap `next()` menambahkan satu padanya, dan seluruh perilaku `hasNext()` hanyalah satu perbandingan:

$$\text{hasNext}() \Leftrightarrow c \neq n \tag{13.1}$$

Rumus mungil ini, Persamaan (13.1), tampak terlalu jelas untuk dituliskan. Tetap ingatlah ia: di Subbab 13.5 rumus ini akan menjelaskan sebuah *bug* yang diam-diam melompati satu elemen.

Blok kedua **Kode 13.1** memperlihatkan apa yang *tidak* dapat dilakukan iterator. Di dalam perulangan, `String element = it2.next();` menyalin sebuah *referensi* ke string itu ke dalam variabel lokal `element`. Pernyataan `element += " and Zoro, Roronoa";` membangun string yang sama sekali baru dan membuat `element` merujuk kepadanya. Koleksi masih memegang referensi lama, sehingga mencetak `captain` sesudahnya memperlihatkan `Luffy, Monkey D.` yang tidak berubah. String itu imutabel, jadi sejak awal tidak ada cara untuk mengubah string itu sendiri; dan menugaskan ulang variabel lokal tidak pernah menjangkau kembali ke dalam koleksi. Seandainya elemen-elemennya objek *mutabel*, `element.setName(...)` *memang* akan mengubah objek di dalam koleksi — sebuah perbedaan yang diperiksa dengan cermat di Subbab 13.7.

Blok ketiga menghitung `sum = 9` dengan perulangan `for` yang disempurnakan, dan blok keempat memperlihatkan ujung jalan. Setelah tiga panggilan `next()` pada koleksi tiga elemen, `hasNext()` bernilai `false`, dan `next()` keempat melemparkan `NoSuchElementException`. Program yang benar tidak pernah menjumpai eksepsi ini: ia bertanya kepada `hasNext()` lebih dahulu.

13.2.2 Protokol yang sama di Python

Python memiliki gagasan yang persis sama dengan nama yang berbeda. Objek apa pun yang dapat ditelusuri disebut **iterable**; memanggil fungsi bawaan `iter(x)` padanya mengembalikan sebuah

iterator; memanggil `next(it)` pada iterator mengembalikan elemen berikutnya, dan bila elemen sudah habis, iterator membangkitkan eksepsi `StopIteration` [4]. Tidak ada `hasNext()` tersendiri: di Python Anda cukup mencoba mengambil elemen berikutnya, dan eksepsi itulah jawaban "tidak ada lagi". Rancangan ini ditetapkan pada tahun 2001 oleh proposal yang memberi Python iteratornya, PEP 234 [5]. **Kode 13.2** adalah kembaran Python dari **Kode 13.1**.

Kode 13.2 `iterator_basics.py`: `iter()`, `next()`, dan `StopIteration`, perulangan `for`, serta meminta satu elemen terlalu banyak.

```
"""iterator_basics.py: protokol iterator Python, langkah demi langkah."""


def main():
    # 1. Cetak setiap string dalam koleksi dengan iterator eksplisit.
    crew = ["Luffy, Monkey D.", "Zoro, Roronoa", "Robin, Nico"]
    it = iter(crew)               # minta sebuah iterator kepada iterable
    while True:
        try:
            element = next(it)    # elemen berikutnya, atau StopIteration
        except StopIteration:
            break
        print(element)

    # 2. Mengubah variabel lokal tidak mengubah koleksi.
    print()
    captain = ["Luffy, Monkey D."]
    for element in captain:
        element += " and Zoro, Roronoa"
        print(element)
    print("Print captain's content:")
    for s in captain:
        print(s)

    # 3. Perulangan for mengerjakan langkah 1 dan 2 untuk kita.
    print()
    c = [3, 2, 4]
    total = 0
    for i in c:
        total += i
    print("sum =", total)

    # 4. Meminta satu elemen terlalu banyak.
    it3 = iter(c)
    next(it3)
    next(it3)
    next(it3)
    print("next(it3, 'no more') after three calls:", next(it3, "no more"))
    try:
        next(it3)
    except StopIteration as e:
        print("fourth next():", type(e).__name__)
    print("an iterator is its own iterator:", iter(it3) is it3)


```

Kode 13.2 (Lanjutan) *iterator_basics.py: iter(), next(), dan StopIteration, perulangan for, serta meminta satu elemen terlalu banyak.*

```
if __name__ == "__main__":
    main()
```

Jalankan dengan `python3 iterator_basics.py` (di Windows, `python iterator_basics.py` atau `py iterator_basics.py`):

```
Luffy, Monkey D.
Zoro, Roronoa
Robin, Nico

Luffy, Monkey D. and Zoro, Roronoa
Print captain's content:
Luffy, Monkey D.

sum = 9
next(it3, 'no more') after three calls: no more
fourth next(): StopIteration
an iterator is its own iterator: True
```

Blok pertama adalah apa yang dikerjakan perulangan `for` Python di balik layar. `iter(crew)` meminta sebuah iterator kepada list; perulangan `while True` memanggil `next(it)` berulang-ulang; ketika `next` membangkitkan `StopIteration`, klausa `except` meninggalkan perulangan dengan `break`. Tidak ada yang menulis perulangan seperti ini dalam kode sungguhan, tetapi ada gunanya menuliskannya sekali, untuk melihat bahwa `for element in crew:` bukanlah sihir.

Blok kedua dan ketiga sama dengan versi Java: menugaskan ulang variabel perulangan `element` tidak mengubah list, dan `for i in c` menjumlahkan sampai 9. Blok keempat memperlihatkan dua kemudahan kecil. `next(it3, "no more")` menerima nilai *bawaan* yang dikembalikan sebagai ganti membangkitkan `StopIteration`; inilah padanan Python yang paling dekat dengan memeriksa `hasNext()`. Baris terakhir mencetak `True` untuk `iter(it3) is it3`: memanggil `iter` pada sebuah iterator mengembalikan iterator itu sendiri. Aturan itulah yang membuat perulangan `for` Python dapat menerima baik iterable maupun iterator, dan di Subbab 13.4 kita akan melihat mengapa hal itu penting.

Tabel 13.1 menyandingkan kedua protokol.

Tabel 13.1 Protokol iterasi di Java dan Python.

Gagasan	Java	Python
Sesuatu yang dapat ditelusuri	mengimplementasikan `Iterable<E>`	mendefinisikan `__iter__` (sebuah *iterable*)
Mendapat iterator baru	`c.iterator()`	`iter(c)`, yang memanggil `c.__iter__()`
Masih ada elemen lain?	`it.hasNext()`	tidak ada metode: coba `next` dan tangkap `StopIteration`
Ambil elemen berikutnya	`it.next()`	`next(it)`, yang memanggil `it.__next__()`
Elemen sudah habis	`next()` melemparkan `NoSuchElementException`	`__next__` membangkitkan `StopIteration`
Hapus elemen yang terakhir dikembalikan	`it.remove()` (opsional)	bukan bagian dari protokol
Perulangan yang memakai semua ini	`for (E e : c)`	`for e in c:`
Iterator yang ditulis untuk Anda	—	*generator*: fungsi yang memuat `yield`

13.2.3 Apa yang tidak dapat dilakukan iterator

Iterator sengaja dibuat bersahaja. Melalui `Iterator` biasa Anda hanya dapat bergerak *maju*, satu elemen setiap kali, dan hanya dapat *membaca* elemen yang diberikan kepada Anda. Anda tidak dapat mundur, tidak dapat melompat ke depan, dan tidak dapat meletakkan objek lain ke dalam

koleksi di tempat objek yang baru saja Anda terima. Jika Anda menginginkan koleksi yang sebagian elemennya diganti, jawaban iterator biasa adalah: buatlah koleksi baru, lalu tambahkan elemen ke dalamnya sambil menelusuri koleksi yang lama.

Kebersahajaan itu justru sebuah keunggulan: karena iterator menjanjikan begitu sedikit, *setiap* koleksi sanggup menepati janjinya — list, set, pohon, berkas yang dibaca baris demi baris, bahkan deretan yang dihitung saat diminta dan tidak pernah disimpan, seperti di Subbab 13.4.

13.3 Iterable, Perulangan for yang Disempurnakan, dan ListIterator

Perulangan eksplisit `while (iter.hasNext())` itu tepat tetapi bertele-tele. Karena itu Java menawarkan **perulangan `for` yang disempurnakan** (*enhanced for loop*, juga disebut *for-each*), dan perulangan `for` biasa di Python adalah gagasan yang persis sama. Kelas apa pun yang mengimplementasikan antarmuka `java.lang.Iterable<T>` — yang hanya memiliki satu metode, `Iterator<T> iterator()` — boleh muncul sesudah titik dua, dan setiap kelas yang mengimplementasikan `Collection` juga mengimplementasikan `Iterable`. Perulangan yang disempurnakan mengunjungi elemen dengan arah maju dan satu per satu, sama seperti iterator biasa.

Spesifikasi Bahasa Java mendefinisikan perulangan yang disempurnakan melalui penerjemahan [6]. Ketika Anda menulis

```
for (Integer i : c) {
    sum += i;
}
```

kompilator mengubahnya menjadi

```
for (Iterator<Integer> $it = c.iterator(); $it.hasNext(); ) {
    Integer i = $it.next();
    sum += i;
}
```

dengan `$it` sebagai variabel tersembunyi yang tidak dapat Anda sebut namanya. Jadi blok ketiga **Kode 13.1** *adalah* blok pertama, ditulis lebih pendek. `for i in c:` di Python diterjemahkan dengan cara yang sama menjadi perulangan yang ditulis dengan tangan pada blok pertama **Kode 13.2**. (Untuk larik, perulangan yang disempurnakan di Java memakai indeks alih-alih iterator, karena larik bukan objek yang mengimplementasikan `Iterable`.)

Perulangan yang disempurnakan menyembunyikan iteratornya, dan hal itu membawa satu akibat yang harus Anda ingat: **di dalam perulangan for-each Anda tidak memegang iteratornya**, sehingga Anda tidak dapat memanggil metode `remove()` miliknya. Subbab 13.5 memperlihatkan mengapa hal ini penting.

13.3.1 Menghapus duplikat yang bersebelahan

Berikut sebuah tugas kecil yang memperlihatkan keterbatasan iterator dan cara hidup bersamanya. Sebuah list sudah terurut, sehingga elemen-elemen yang sama duduk bersebelahan, dan kita menginginkan list baru yang memuat setiap nilai hanya sekali: `[Ace, Luffy, Luffy, Sanji, Sanji, Zoro, Zoro, Zoro]` harus menjadi `[Ace, Luffy, Sanji, Zoro]`. Dengan iterator biasa kita tidak dapat menoleh ke belakang, dan kita tidak dapat menghapus elemen yang sedang kita lihat dari dalam perulangan for-each. Penyelesaian klasiknya adalah memakai sebuah variabel untuk *mengingat elemen sebelumnya*, dan sebuah list *baru* untuk mengumpulkan elemen-

elemen yang unik. **Kode 13.3** memperlihatkan versi klasik itu, sebuah versi yang lebih aman, dan versi di tempat (*in place*) yang memakai iterator yang lebih cakap.

Kode 13.3 `RemoveDuplicates.java`: menghapus duplikat bersebelahan dari list terurut dengan mengingat elemen sebelumnya, versi yang menangani list kosong dan `null`, serta versi di tempat dengan `ListIterator`.

```
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import java.util.ListIterator;
import java.util.Objects;

public class RemoveDuplicates {
    // Versi klasik: mengingat elemen sebelumnya dalam sebuah variabel.
    public static <T> List<T> removeDuplicatesFromSortedList(List<T> input) {
        // Perhatikan bahwa metode ini tidak sanggup menghapus duplikat bernilai null
        T previous = null;
        List<T> result = new ArrayList<T>();
        result.add(input.get(0));
        Iterator<T> iter = input.iterator();
        while (iter.hasNext()) {
            T current = iter.next();
            if (previous != null && !previous.equals(current)) {
                result.add(current);
            }
            previous = current;
        }
        return result;
    }

    // Versi yang lebih aman: berjalan untuk list kosong dan untuk elemen null.
    public static <T> List<T> dedupSorted(List<T> input) {
        List<T> result = new ArrayList<>();
        boolean first = true;
        T previous = null;
        for (T current : input) {
            if (first || !Objects.equals(previous, current)) {
                result.add(current);
            }
            previous = current;
            first = false;
        }
        return result;
    }
```

Kode 13.3 (Lanjutan) *RemoveDuplicates.java: menghapus duplikat bersebelahan dari list terurut dengan mengingat elemen sebelumnya, versi yang menangani list kosong dan null, serta versi di tempat dengan ListIterator.*

```
    // Di tempat, dengan ListIterator: tidak perlu list kedua.
    public static <T> void dedupSortedInPlace(List<T> list) {
        ListIterator<T> it = list.listIterator();
        boolean first = true;
        T previous = null;
        while (it.hasNext()) {
            T current = it.next();
            if (!first && Objects.equals(previous, current)) {
                it.remove();  // menghapus elemen yang baru saja dikembalikan next()
            }
            previous = current;
            first = false;
        }
    }

    public static void main(String[] args) {
        List<String> myList = new ArrayList<>();
        myList.add("Ace, Portgas D.");
        myList.add("Luffy, Monkey D.");
        myList.add("Luffy, Monkey D.");
        myList.add("Sanji, Vinsmoke");
        myList.add("Sanji, Vinsmoke");
        myList.add("Zoro, Roronoa");
        myList.add("Zoro, Roronoa");
        myList.add("Zoro, Roronoa");
        System.out.println("Print myList's content:");
        System.out.println(myList);
        System.out.println("After removeDuplicatesFromSortedList():");
        System.out.println(removeDuplicatesFromSortedList(myList));

        List<String> empty = new ArrayList<>();
        try {
            removeDuplicatesFromSortedList(empty);
        } catch (IndexOutOfBoundsException e) {
            System.out.println("empty list: " + e.getClass().getSimpleName());
        }
        System.out.println("dedupSorted(empty) = " + dedupSorted(empty));
        List<String> withNulls = new ArrayList<>();
        withNulls.add(null);
        withNulls.add(null);
        withNulls.add("Nami");
        System.out.println("classic with nulls: " +
removeDuplicatesFromSortedList(withNulls));
        System.out.println("dedupSorted with nulls: " + dedupSorted(withNulls));
```

Kode 13.3 (Lanjutan) *RemoveDuplicates.java: menghapus duplikat bersebelahan dari list terurut dengan mengingat elemen sebelumnya, versi yang menangani list kosong dan null, serta versi di tempat dengan ListIterator.*

```
        dedupSortedInPlace(myList);
        System.out.println("in place: " + myList);

        // ListIterator juga dapat berjalan mundur dan mengganti elemen.
        ListIterator<String> back = myList.listIterator(myList.size());
        StringBuilder sb = new StringBuilder("backwards:");
        while (back.hasPrevious()) {
            int index = back.previousIndex();
            String name = back.previous();
            back.set(name.toUpperCase());
            sb.append(" ").append(index).append("=").append(name.substring(0,
name.indexOf(',')));
        }
        System.out.println(sb);
        System.out.println("after set(): " + myList);
    }
}
```

```
Print myList's content:
[Ace, Portgas D., Luffy, Monkey D., Luffy, Monkey D., Sanji, Vinsmoke, Sanji, Vinsmoke,
Zoro, Roronoa, Zoro, Roronoa, Zoro, Roronoa]
After removeDuplicatesFromSortedList():
[Ace, Portgas D., Luffy, Monkey D., Sanji, Vinsmoke, Zoro, Roronoa]
empty list: IndexOutOfBoundsException
dedupSorted(empty) = []
classic with nulls: [null]
dedupSorted with nulls: [null, Nami]
in place: [Ace, Portgas D., Luffy, Monkey D., Sanji, Vinsmoke, Zoro, Roronoa]
backwards: 3=Zoro 2=Sanji 1=Luffy 0=Ace
after set(): [ACE, PORTGAS D., LUFFY, MONKEY D., SANJI, VINSMOKE, ZORO, RORONOA]
```

Metode pertama, `removeDuplicatesFromSortedList`, adalah versi yang dapat Anda temukan di banyak buku, dan ia menyimpan dua cacat tersembunyi yang layak dipelajari, karena keduanya lahir dari anggapan ceroboh yang sama.

- Ia dimulai dengan `result.add(input.get(0))`, sehingga ia menganggap list itu memiliki elemen pertama. Pada list kosong, `get(0)` melemparkan `IndexOutOfBoundsException`, seperti tampak pada keluaran.
- Ia memakai `null` sebagai penanda yang berarti "belum ada elemen sebelumnya". Uji `previous != null && !previous.equals(current)` lalu melewati elemen pertama perulangan (yang sudah ditambahkan) dan menambahkan setiap elemen yang berbeda dari pendahulunya. Namun jika list itu sendiri memuat `null`, penandanya menjadi ambigu. Dengan list `[null, null, "Nami"]`, `null` pertama ditambahkan oleh `get(0)`; lalu `previous` bernilai `null` pada setiap langkah, sehingga tidak ada lagi yang pernah ditambahkan — dan `"Nami"` hilang. Komentar di dalam kode bahkan memperingatkan bahwa metode ini "tidak sanggup menghapus duplikat bernilai null"; kenyataannya lebih buruk, karena nilai yang *bukan* null pun ikut terbuang.

Metode kedua, `dedupSorted`, memperbaiki kedua cacat itu dengan satu gagasan: alih-alih membebani `null` dengan makna khusus, ia menyimpan `boolean first` tersendiri yang bernilai `true` hanya pada langkah pertama, dan ia membandingkan dengan `Objects.equals(previous, current)`, yang menganggap dua `null` sama dan tidak pernah melemparkan eksepsi. Metode ini berjalan pada list kosong dan mempertahankan `"Nami"`. Setiap kali Anda mendapati diri memakai nilai sungguhan seperti `null`, `0`, atau `-1` untuk berarti "belum ada apa-apa", tanyakan apakah nilai itu mungkin juga muncul sebagai data; bila mungkin, pakailah penanda tersendiri.

13.3.2 ListIterator: mundur, mengganti, menyisipkan, menghapus

Ada satu pengecualian terhadap aturan bahwa iterator hanya bergerak maju dan hanya membaca, dan pengecualian itu berlaku untuk list. Sebuah `List` juga dapat memberikan `java.util.ListIterator`, iterator yang lebih kaya yang dapat bergerak **mundur** (`hasPrevious()`, `previous()`), melaporkan **indeks** (`nextIndex()`, `previousIndex()`), **mengganti** elemen yang terakhir dikembalikan (`set(e)`), **menyisipkan** di posisi kursor (`add(e)`), dan **menghapus** elemen yang terakhir dikembalikan (`remove()`) [3].

Metode ketiga **Kode 13.3**, `dedupSortedInPlace`, memakai `it.remove()` untuk menghapus setiap duplikat dari list itu sendiri, sehingga tidak diperlukan list kedua; `remove()` menghapus elemen yang dikembalikan oleh `next()` terakhir, dan iterator tetap konsisten karena ia sendiri yang melakukan perubahan. Bagian akhir `main` meminta `myList.listIterator(myList.size())` — iterator yang kursornya mulai *sesudah* elemen terakhir — lalu berjalan mundur dengan `hasPrevious()` dan `previous()`, mencetak setiap indeks dan mengganti setiap nama dengan versi huruf kapitalnya melalui `set()`. Keluaran memperlihatkan indeks dalam urutan `3 2 1 0` dan list dalam huruf kapital sesudahnya.

Python tidak memiliki `ListIterator`, karena tidak memerlukannya: list Python berindeks, dan fungsi-fungsi bawaan menangani sisanya. **Kode 13.4** mengerjakan keempat tugas yang sama.

Kode 13.4 `remove_duplicates.py`: tugas yang sama dalam Python, dengan penanda, dengan `itertools.groupby`, di tempat dengan menelusuri indeks dari ujung, dan berjalan mundur dengan `reversed`.

```
"""remove_duplicates.py: menghapus duplikat yang bersebelahan, dengan tiga cara."""
from itertools import groupby


def remove_duplicates_from_sorted_list(items):
    """Versi klasik: ingat elemen sebelumnya dalam sebuah variabel."""
    result = []
    first = True
    previous = None
    for current in items:
        if first or previous != current:
            result.append(current)
        previous = current
        first = False
    return result


def dedup_sorted_in_place(items):
    """Di tempat: telusuri indeks dari ujung agar penghapusan tidak menggeser yang
tersisa."""
    for i in range(len(items) - 1, 0, -1):
        if items[i] == items[i - 1]:
            del items[i]


def main():
    my_list = ["Ace, Portgas D.", "Luffy, Monkey D.", "Luffy, Monkey D.",
               "Sanji, Vinsmoke", "Sanji, Vinsmoke",
               "Zoro, Roronoa", "Zoro, Roronoa", "Zoro, Roronoa"]
    print("Print my_list's content:")
    print(my_list)
    print("After remove_duplicates_from_sorted_list():")
    print(remove_duplicates_from_sorted_list(my_list))
    print("empty list:", remove_duplicates_from_sorted_list([]))
    print("with None:", remove_duplicates_from_sorted_list([None, None, "Nami"]))

    # groupby() menghasilkan satu pasangan (key, group) untuk setiap deretan tetangga yang
sama.
    print("groupby:", [key for key, _ in groupby(my_list)])

    dedup_sorted_in_place(my_list)
    print("in place:", my_list)
```

Kode 13.4 (Lanjutan) *remove_duplicates.py: tugas yang sama dalam Python, dengan penanda, dengan itertools.groupby, di tempat dengan menelusuri indeks dari ujung, dan berjalan mundur dengan reversed.*

```
    # Berjalan mundur dan mengganti elemen: reversed() dan penugasan lewat indeks.
    parts = []
    for index in reversed(range(len(my_list))):
        name = my_list[index]
        my_list[index] = name.upper()   # penugasan lewat indeks selalu diizinkan
        parts.append(f"{index}={name[:name.index(',')]}")
    print("backwards:", " ".join(parts))
    print("after assignment:", my_list)


if __name__ == "__main__":
    main()
```

```
Print my_list's content:
['Ace, Portgas D.', 'Luffy, Monkey D.', 'Luffy, Monkey D.', 'Sanji, Vinsmoke', 'Sanji,
Vinsmoke', 'Zoro, Roronoa', 'Zoro, Roronoa', 'Zoro, Roronoa']
After remove_duplicates_from_sorted_list():
['Ace, Portgas D.', 'Luffy, Monkey D.', 'Sanji, Vinsmoke', 'Zoro, Roronoa']
empty list: []
with None: [None, 'Nami']
groupby: ['Ace, Portgas D.', 'Luffy, Monkey D.', 'Sanji, Vinsmoke', 'Zoro, Roronoa']
in place: ['Ace, Portgas D.', 'Luffy, Monkey D.', 'Sanji, Vinsmoke', 'Zoro, Roronoa']
backwards: 3=Zoro 2=Sanji 1=Luffy 0=Ace
after assignment: ['ACE, PORTGAS D.', 'LUFFY, MONKEY D.', 'SANJI, VINSMOKE', 'ZORO,
RORONOA']
```

`remove_duplicates_from_sorted_list` sudah berupa versi yang aman sejak awal: ia memakai penanda `first`, dan `!=` di Python membandingkan `None` dengan `None` tanpa keberatan. Baris yang memuat `groupby` memperlihatkan sebuah alat dari modul `itertools` yang akan kita jumpai lagi di Subbab 13.10: `groupby(my_list)` menghasilkan satu pasangan `(key, group)` untuk setiap *deretan* tetangga yang sama, sehingga daftar kuncinya tepat berupa list tanpa duplikat bersebelahan. Untuk menghapus di tempat, `dedup_sorted_in_place` menelusuri indeks **dari ujung menuju awal**: menghapus `items[i]` hanya menggeser elemen *sesudah* `i`, yang sudah dikunjungi perulangan, sehingga tidak ada yang terlewat. (Berjalan maju sambil menghapus adalah kesalahan klasik; Subbab 13.5 memperlihatkannya.) Akhirnya, `reversed(range(len(my_list)))` menghasilkan indeks `3, 2, 1, 0`, dan penugasan biasa `my_list[index] = ...` mengganti sebuah elemen — pekerjaan yang di Java dilakukan `ListIterator.set()`.

13.4 Menulis Koleksi Iterable Sendiri

Memakai iterator itu berguna; *menulisnya* adalah tempat gagasan ini membuahkan hasil. Begitu sebuah kelas buatan kita bersifat iterable, ia dapat dipakai dalam perulangan for-each, dengan setiap metode pustaka yang menerima `Iterable`, dan — di Python — dengan `list()`, `sum()`, `max()`, `sorted()`, `zip()`, serta setiap fungsi lain yang mengonsumsi iterable.

Contoh kita berasal dari lantai binatu. Seorang pelanggan membawa 12.5 kg cucian, dan mesin-mesinnya menampung paling banyak 5 kg per muatan. Petugas memerlukan *rencananya*: 5 kg, 5 kg, lalu 2.5 kg. Kita memodelkan rencana itu sebagai koleksi berat muatan. Bagian menariknya, rencana itu tidak pernah perlu *menyimpan* muatannya. Bila total dan kapasitasnya diketahui, muatan berikutnya selalu dapat dihitung dari sisa yang belum dimuat, sehingga koleksi ini dapat menghasilkan elemennya saat diminta.

13.4.1 Java: Iterable ditambah kelas iterator dalam

Kode 13.5 mengimplementasikan `Iterable<Double>` dan memberikan iterator dari sebuah kelas dalam (*inner class*) yang privat.

Kode 13.5 `LoadPlanDemo.java`: `LoadPlan` adalah `Iterable` yang menghitung muatannya saat diminta; setiap panggilan `iterator()` mengembalikan `LoadIterator` yang mandiri.

```
import java.util.Iterator;
import java.util.NoSuchElementException;

public class LoadPlanDemo {
    public static void main(String[] args) {
        LoadPlan plan = new LoadPlan(12.5, 5.0);
        System.out.println(plan);
        for (double load : plan) {           // berjalan karena LoadPlan adalah Iterable
            System.out.println("  load of " + load + " kg");
        }

        // Setiap panggilan iterator() memulai penelusuran baru yang mandiri.
        Iterator<Double> a = plan.iterator();
        Iterator<Double> b = plan.iterator();
        a.next();
        a.next();
        System.out.println("a is at its third load: " + a.next() + " kg");
        System.out.println("b is still at its first: " + b.next() + " kg");

        // Tak ada yang dihitung sebelum diminta: rencana raksasa tidak berbiaya untuk
dibuat.
        LoadPlan huge = new LoadPlan(1_000_000.0, 7.5);
        Iterator<Double> h = huge.iterator();
        System.out.println("first load of the huge plan: " + h.next() + " kg");
        System.out.println("loads in the huge plan: " + huge.count());
    }
}

final class LoadPlan implements Iterable<Double> {
    private final double totalKg;
    private final double capacityKg;

    LoadPlan(double totalKg, double capacityKg) {
        if (totalKg <= 0 || capacityKg <= 0) {
            throw new IllegalArgumentException("weights must be positive");
        }
        this.totalKg = totalKg;
        this.capacityKg = capacityKg;
    }

    @Override
    public Iterator<Double> iterator() {
        return new LoadIterator();
    }
```

Kode 13.5 (Lanjutan) *LoadPlanDemo.java: LoadPlan adalah Iterable yang menghitung muatannya saat diminta; setiap panggilan iterator() mengembalikan LoadIterator yang mandiri.*

```
    int count() {
        int n = 0;
        for (double ignored : this) {
            n++;
        }
        return n;
    }

    @Override
    public String toString() {
        return "LoadPlan[" + totalKg + " kg in machines of " + capacityKg + " kg]";
    }

    // Kelas dalam: setiap LoadIterator dapat membaca field LoadPlan-nya.
    private class LoadIterator implements Iterator<Double> {
        private double remaining = totalKg;  // posisi milik iterator itu sendiri

        @Override
        public boolean hasNext() {
            return remaining > 0;
        }

        @Override
        public Double next() {
            if (!hasNext()) {
                throw new NoSuchElementException("no loads left");
            }
            double load = Math.min(remaining, capacityKg);
            remaining -= load;
            return load;
        }
    }
}
```

```
LoadPlan[12.5 kg in machines of 5.0 kg]
  load of 5.0 kg
  load of 5.0 kg
  load of 2.5 kg
a is at its third load: 2.5 kg
b is still at its first: 5.0 kg
first load of the huge plan: 7.5 kg
loads in the huge plan: 133334
```

Strukturnya terdiri atas dua bagian, dan memisahkan keduanya adalah seluruh rahasianya.

- **Koleksinya, `LoadPlan`.** Ia hanya memegang apa yang mendefinisikan rencana: `totalKg` dan `capacityKg`, keduanya `final`. `iterator()` miliknya hanya mengerjakan satu hal: `return new LoadIterator();`. `count()` miliknya adalah perulangan for-each atas `this`.
- **Iteratornya, `LoadIterator`.** Ia hanya memegang apa yang mendefinisikan sebuah *penelusuran*: `remaining`, berat yang belum dimasukkan ke muatan mana pun, yang mulai dari `totalKg`. `hasNext()` adalah `remaining > 0`. `next()` pertama-tama memeriksa `hasNext()` dan melemparkan `NoSuchElementException` bila penelusuran sudah selesai — kontrak `Iterator` mewajibkannya — lalu mengambil `Math.min(remaining, capacityKg)` sebagai muatan, menguranginya, dan mengembalikannya.

`LoadIterator` dideklarasikan `private class` *di dalam* `LoadPlan` tanpa kata `static`. Hal itu menjadikannya **kelas dalam**: setiap objek `LoadIterator` terikat pada `LoadPlan` yang menciptakannya dan dapat membaca field rencana itu secara langsung, itulah sebabnya `remaining = totalKg` dan `Math.min(remaining, capacityKg)` lolos kompilasi tanpa awalan

`plan.` apa pun. Bab 10 memakai kelas dalam untuk *listener* peristiwa; iterator adalah pemakaian klasik yang lain. Karena kelas itu `private`, tidak seorang pun di luar `LoadPlan` dapat menyebut namanya — bagian program lainnya hanya melihat antarmuka `Iterator<Double>`.

Keluaran menegaskan tiga sifat. Pertama, perulangan for-each berjalan: `for (double load : plan)` mencetak 5.0, 5.0, dan 2.5 (objek `Double` dibuka otomatis menjadi `double`). Kedua, iterator-iterator itu **mandiri**: setelah `a` mengambil dua muatan, `a.next()` bernilai 2.5 sementara iterator baru `b` tetap mulai dari 5.0; setiap penelusuran memiliki `remaining` sendiri. Ketiga, rencana itu **malas** (*lazy*): membuat rencana untuk sejuta kilogram tidak berbiaya apa pun, muatan pertama langsung tersedia, dan 133,334 muatan yang ditelusuri `count()` dihitung satu per satu tanpa pernah disimpan. (Satu juta dibagi 7.5 adalah 133,333.$\overline{3}$, jadi ada 133,333 muatan penuh dan satu muatan terakhir seberat 2.5 kg.)

13.4.2 Python: iter, next, dan generator

Di Python rancangan yang sama memakai dua metode khusus. Sebuah **iterable** mendefinisikan `__iter__`, yang mengembalikan iterator. Sebuah **iterator** mendefinisikan `__next__`, yang mengembalikan elemen berikutnya atau membangkitkan `StopIteration`, dan juga `__iter__`, yang mengembalikan `self` (aturan yang kita jumpai di Subbab 13.2.2). **Kode 13.6** menulis kedua kelas itu dengan tangan, lalu menulis iterable yang sama dengan cara kedua yang jauh lebih pendek.

Kode 13.6 `load_plan.py`: iterable dengan kelas iterator yang ditulis dengan tangan, dan iterable yang sama yang ditulis dengan fungsi generator.

```python
"""load_plan.py: iterable yang ditulis dengan tangan, lalu yang sama sebagai generator."""
from itertools import islice


class LoadIterator:
    """Iteratornya: mengingat berapa banyak yang masih harus dimuat."""

    def __init__(self, total_kg, capacity_kg):
        self._remaining = total_kg
        self._capacity = capacity_kg

    def __iter__(self):
        return self                     # sebuah iterator adalah iterator bagi dirinya sendiri

    def __next__(self):
        if self._remaining <= 0:
            raise StopIteration         # sinyal "tidak ada elemen lagi"
        load = min(self._remaining, self._capacity)
        self._remaining -= load
        return load


class LoadPlan:
    """Iterable-nya: setiap panggilan __iter__ memulai penelusuran baru."""

    def __init__(self, total_kg, capacity_kg):
        if total_kg <= 0 or capacity_kg <= 0:
            raise ValueError("weights must be positive")
        self.total_kg = total_kg
        self.capacity_kg = capacity_kg

    def __iter__(self):
        return LoadIterator(self.total_kg, self.capacity_kg)

    def __repr__(self):
        return f"LoadPlan[{self.total_kg} kg in machines of {self.capacity_kg} kg]"


class GeneratorLoadPlan(LoadPlan):
    """Iterable yang sama; fungsi generator menuliskan iteratornya untuk kita."""
```

Kode 13.6 (Lanjutan) *load_plan.py: iterable dengan kelas iterator yang ditulis dengan tangan, dan iterable yang sama yang ditulis dengan fungsi generator.*

```
    def __iter__(self):
        remaining = self.total_kg
        while remaining > 0:
            load = min(remaining, self.capacity_kg)
            remaining -= load
            print(f"    (generator computes {load})")
            yield load                  # berhenti sejenak di sini sampai next()
berikutnya


def main():
    plan = LoadPlan(12.5, 5.0)
    print(plan)
    for load in plan:
        print(f"  load of {load} kg")

    a, b = iter(plan), iter(plan)
    next(a)
    next(a)
    print("a is at its third load:", next(a), "kg")
    print("b is still at its first:", next(b), "kg")

    # Iterator habis setelah satu kali penelusuran; iterable tidak.
    it = iter(plan)
    print("list(it) =", list(it), " list(it) again =", list(it))
    print("list(plan) twice:", list(plan), list(plan))

    lazy = GeneratorLoadPlan(12.5, 5.0)
    g = iter(lazy)
    print("generator created, nothing computed yet")
    print("first:", next(g))
    print("rest:", list(g))

    # Tak ada yang dihitung sebelum diminta: rencana raksasa tidak berbiaya untuk dibuat.
    huge = LoadPlan(1_000_000.0, 7.5)
    print("first two loads of the huge plan:", list(islice(huge, 2)))
    print("loads in the huge plan:", sum(1 for _ in huge))


if __name__ == "__main__":
    main()
```

```
LoadPlan[12.5 kg in machines of 5.0 kg]
  load of 5.0 kg
  load of 5.0 kg
  load of 2.5 kg
a is at its third load: 2.5 kg
b is still at its first: 5.0 kg
list(it) = [5.0, 5.0, 2.5]  list(it) again = []
list(plan) twice: [5.0, 5.0, 2.5] [5.0, 5.0, 2.5]
generator created, nothing computed yet
    (generator computes 5.0)
first: 5.0
    (generator computes 5.0)
    (generator computes 2.5)
rest: [5.0, 2.5]
first two loads of the huge plan: [7.5, 7.5]
loads in the huge plan: 133334
```

`LoadIterator` dan `LoadPlan` mencerminkan kelas-kelas Java baris demi baris; satu-satunya perbedaan struktural adalah bahwa Python tidak memiliki kelas dalam yang otomatis dapat mengakses objek luarnya, sehingga `LoadPlan.__iter__` meneruskan kedua bilangan itu ke konstruktor iterator.

Kelas `GeneratorLoadPlan` adalah tempat Python meninggalkan Java di belakang. `__iter__` miliknya memuat kata `yield`, dan fungsi yang memuat `yield` adalah **fungsi generator**: memanggilnya tidak menjalankan badannya; ia mengembalikan sebuah *generator*, yaitu iterator yang `__next__`-nya menjalankan badan fungsi *sampai `yield` berikutnya*, menyerahkan nilainya, lalu **berhenti sejenak** di sana dengan seluruh variabel lokalnya utuh. Panggilan `next()` berikutnya melanjutkan tepat sesudah `yield` itu. Ketika badannya selesai, generator membangkitkan `StopIteration` dengan sendirinya. Jadi kelas `LoadIterator` delapan baris, dengan konstruktornya, `__iter__`-nya, `__next__`-nya, dan `StopIteration` eksplisitnya, telah menjadi perulangan empat baris. Generator ditambahkan ke Python untuk tujuan persis ini oleh PEP 255 [7], dan kini menjadi salah satu alat yang paling khas bahasa itu [8].

Baris-baris cetakan `(generator computes ...)` memungkinkan kita menyaksikan jeda itu. Setelah `g = iter(lazy)`, belum ada yang dihitung. `next(g)` menjalankan badan fungsi sampai `yield` pertama — kita melihat satu baris "computes", lalu `first: 5.0`. `list(g)` kemudian menggerakkan generator sampai habis, dan kita melihat dua perhitungan sisanya sebelum list dicetak.

> **DEFINISI. Fungsi generator** adalah fungsi Python yang badannya memuat `yield`. Memanggilnya mengembalikan sebuah **generator**, yaitu iterator yang menjalankan badan fungsi langkah demi langkah: setiap `next()` melanjutkan badan fungsi, menjalankannya sampai `yield` berikutnya, mengembalikan nilai yang di-*yield*, lalu berhenti lagi; ketika badan fungsi berakhir, generator membangkitkan `StopIteration`.

Java tidak memiliki `yield` untuk iterator. Yang paling mendekati di Java adalah *stream* yang dibangun dengan `Stream.iterate` atau `Stream.generate`, yang kita jumpai di Subbab 13.10; untuk kelas yang harus bersifat `Iterable`, iterator dalam yang eksplisit seperti di **Kode 13.5** tetap menjadi cara bakunya.

13.4.3 Iterable bukanlah iterator

Bagian tengah **Kode 13.6** memperagakan kebingungan paling umum tentang iterasi di Python. `it = iter(plan)` adalah sebuah *iterator*; `list(it)` menelusurinya sampai habis dan mengembalikan `[5.0, 5.0, 2.5]`; `list(it)` yang kedua mengembalikan `[]`, karena iterator itu sudah **habis terpakai** — kursornya berada melewati ujung, dan tidak ada cara untuk memundurkannya. Sebaliknya, `list(plan)` berhasil setiap kali, karena `plan` adalah sebuah *iterable*, dan setiap `list(plan)` memanggil `plan.__iter__()` lagi dan memperoleh iterator baru.

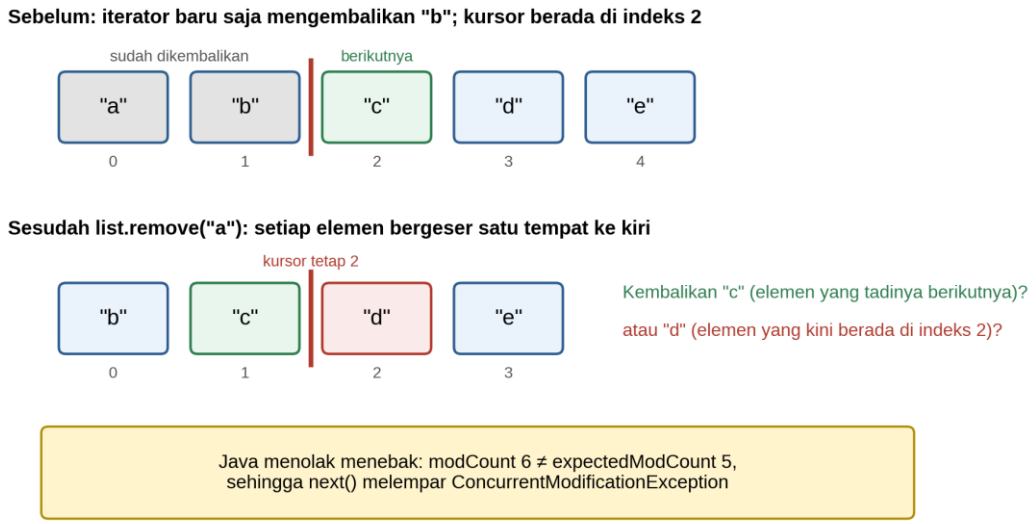
Java membuat pembedaan yang sama, dan sistem tipenya menegakkannya: perulangan for-each menerima `Iterable` dan menolak `Iterator`. Perulangan `for` Python menerima keduanya, karena `__iter__` sebuah iterator mengembalikan dirinya sendiri. Kemudahan itu ada harganya: fungsi yang menelusuri argumennya dua kali akan berjalan baik bila diberi list, tetapi diam-diam tidak mengerjakan apa-apa pada kali kedua bila diberi generator. Kaidah praktisnya sederhana. Jika Anda perlu menelusuri sesuatu lebih dari sekali, pastikan Anda memegang **iterable** (list, tuple, objek seperti `LoadPlan`), bukan iterator sekali pakai; bila ragu, panggil `list(...)` sekali dan simpan list-nya.

Baris-baris terakhir memakai `itertools.islice(huge, 2)`, yang secara malas mengambil dua elemen pertama, dan `sum(1 for _ in huge)`, yang menghitung banyaknya elemen tanpa membangun list.

13.5 Mengubah Koleksi Selagi Menelusurinya

Kita sudah dua kali diperingatkan bahwa iterator tidak dibuat untuk mengubah koleksinya. Apa yang sebenarnya terjadi bila kita mencobanya? Ambil list `["a", "b", "c", "d", "e"]` dan misalkan sebuah iterator baru saja mengembalikan `"b"`, sehingga kursornya berada di celah 2, di depan `"c"`. Sekarang ada kode yang menghapus `"a"` dari list itu — bukan melalui iterator, melainkan dengan memanggil `remove` milik list sendiri. Setiap elemen bergeser satu tempat ke kiri. Kursor iterator tetap bernilai 2. Apa yang harus dikembalikan `next()`: `"c"`, elemen yang tadinya berikutnya dan kini berada di indeks 1, atau `"d"`, elemen yang kini berada di indeks 2?

Gambar 13.2 menggambarkan dilema itu. Tidak ada jawaban yang baik. Mengembalikan `"d"` berarti melompati `"c"` tanpa jejak; mengembalikan `"c"` menuntut iterator mengetahui setiap perubahan yang dilakukan oleh siapa pun. Karena itu koleksi-koleksi Java menolak menebak. Iterator menyadari bahwa list itu telah **diubah secara struktural** — ukurannya berubah — oleh pihak lain selain dirinya, dan panggilan `next()` berikutnya melemparkan eksepsi waktu jalan bertipe `ConcurrentModificationException`.



Gambar 13.2 Mengapa Java melemparkan `ConcurrentModificationException`. Setelah `remove("a")` elemen-elemen bergeser ke kiri, tetapi kursor iterator tidak; tidak ada jawaban yang benar untuk "apa berikutnya?", maka iterator menolak menjawab.

Bagaimana iterator mengetahuinya? Setiap `ArrayList` menyimpan sebuah pencacah, `modCount`, yang bertambah satu setiap kali ukuran list berubah (setiap `add` dan `remove`, tetapi tidak `set`, yang mengganti sebuah elemen tanpa mengubah ukuran). Ketika sebuah iterator diciptakan, ia menyalin pencacah itu ke field miliknya, `expectedModCount`. `remove()` milik iterator sendiri memperbarui keduanya. Dan pada awal setiap `next()` iterator memeriksa

$$\text{modCount} = \text{expectedModCount} \tag{13.2}$$

Jika Persamaan (13.2) salah, pihak lain telah mengubah list, dan iterator melemparkan eksepsi [3]. Namanya sedikit menyesatkan: "concurrent" di sini tidak berarti dua *thread* (Bab 14 membahasnya); artinya perubahan terjadi *selama* iterasi, bahkan dalam satu thread saja.

Kata *struktural* menjelaskan satu rincian aturannya: selagi beriterasi atas sebuah `List`, memanggil `add` atau `remove` pada list itu terlarang, tetapi memanggil `set` diizinkan, karena `set` tidak memindahkan elemen apa pun dan tidak mengubah `modCount`. **Kode 13.7** mencoba ketiganya, lalu memperlihatkan cara-cara yang benar.

Kode 13.7 `ConcurrentModification.java`: `set` selama iterasi diizinkan; `add` dan `remove` melalui list melemparkan `ConcurrentModificationException`; satu kasus lolos diam-diam; tiga alternatif yang benar.

```
import java.util.ArrayList;
import java.util.ConcurrentModificationException;
import java.util.Iterator;
import java.util.List;

public class ConcurrentModification {
    static List<String> crew() {
        return new ArrayList<>(List.of("Ace", "Luffy", "Sanji", "Zoro", "Nami"));
    }

    public static void main(String[] args) {
        // 1. set() di tengah iterasi: diizinkan, ukurannya tidak berubah.
        List<String> myList = crew();
        for (String name : myList) {
            if (name.equals("Luffy")) {
                myList.set(2, "Sabo");        // ganti Sanji, yang belum dikunjungi
            }
            System.out.print(name + " ");
        }
        System.out.println(" <- set() is fine");

        // 2. add() di tengah iterasi: iterator mengetahuinya.
        myList = crew();
        try {
            for (String name : myList) {
                if (name.equals("Luffy")) {
                    myList.add("Buggy");
                }
            }
        } catch (ConcurrentModificationException e) {
            System.out.println("add() while iterating: " + e.getClass().getSimpleName());
        }

        // 3. remove() melalui list, bukan melalui iterator: vonisnya sama.
        myList = crew();
        try {
            for (String name : myList) {
                if (name.equals("Ace")) {
                    myList.remove(name);
                }
            }
        } catch (ConcurrentModificationException e) {
            System.out.println("remove() while iterating: " +
e.getClass().getSimpleName());
        }
```

Kode 13.7 (Lanjutan) *ConcurrentModification.java: set selama iterasi diizinkan; add dan remove melalui list melemparkan ConcurrentModificationException; satu kasus lolos diam-diam; tiga alternatif yang benar.*

```
        // 4. Pemeriksaan ini sebatas upaya terbaik: menghapus elemen kedua dari akhir
        //    mengakhiri perulangan lebih awal, diam-diam, dan Nami tidak pernah
dikunjungi.
        myList = crew();
        StringBuilder seen = new StringBuilder();
        for (String name : myList) {
            seen.append(name).append(' ');
            if (name.equals("Zoro")) {
                myList.remove(name);
            }
        }
        System.out.println("no exception, but visited only: " + seen.toString().trim());

        // 5. Cara-cara yang benar.
        myList = crew();
        Iterator<String> it = myList.iterator();
        while (it.hasNext()) {
            if (it.next().length() <= 3) {
                it.remove();                // penghapusan melalui iterator itu sendiri
            }
        }
        System.out.println("Iterator.remove(): " + myList);

        myList = crew();
        myList.removeIf(name -> name.length() <= 3);
        System.out.println("removeIf():        " + myList);

        myList = crew();
        for (String name : new ArrayList<>(myList)) {  // iterasi atas sebuah salinan
            if (name.startsWith("S")) {
                myList.add(name + " (again)");
            }
        }
        System.out.println("copy, then add:    " + myList);
    }
}
```

```
Ace Luffy Sabo Zoro Nami  <- set() is fine
add() while iterating: ConcurrentModificationException
remove() while iterating: ConcurrentModificationException
no exception, but visited only: Ace Luffy Sanji Zoro
Iterator.remove(): [Luffy, Sanji, Zoro, Nami]
removeIf():        [Luffy, Sanji, Zoro, Nami]
copy, then add:    [Ace, Luffy, Sanji, Zoro, Nami, Sanji (again)]
```

Tiga blok pertama berperilaku sesuai ramalan aturannya. Blok 1 mengganti `Sanji` dengan `Sabo` ketika perulangan berada di `Luffy`, dan perulangan kemudian mengunjungi `Sabo` seperti biasa. Blok 2 dan 3 mengubah ukuran list di dalam perulangan for-each, dan panggilan `next()` *berikutnya* — yang tersembunyi di kepala perulangan — melemparkan eksepsi.

Blok 4 adalah alasan dokumentasi Java menegaskan bahwa perilaku ini hanyalah pemeriksaan "upaya terbaik" (*best-effort*) yang "tidak boleh diandalkan" [3]. Perulangan itu menghapus `Zoro`, elemen kedua dari akhir, ketika sampai padanya. Sekarang terapkan Persamaan (13.1). Setelah `next()` mengembalikan `Zoro`, kursor c bernilai 4. Penghapusan menyusutkan list dari 5 menjadi 4 elemen, jadi $n = 4$. `hasNext()` yang tersembunyi bertanya apakah $c \neq n$, yaitu $4 \neq 4$: salah. Perulangan itu *berakhir* begitu saja — sebelum `next()` sempat memeriksa Persamaan (13.2) — dan `Nami` tidak pernah dikunjungi. Tanpa eksepsi, tanpa peringatan, hanya jawaban yang salah. Itulah sebabnya aturannya bukan "tangkaplah `ConcurrentModificationException`", melainkan "**jangan pernah** mengubah ukuran koleksi selagi perulangan for-each menelusurinya".

Blok 5 memperlihatkan tiga cara aman untuk melakukan apa yang dicoba blok 2–4:

- **Hapus melalui iterator.** Tulislah perulangan eksplisit `while (it.hasNext())` dan panggil `it.remove()`. Iterator menghapus elemen yang terakhir dikembalikannya dan memperbarui `expectedModCount` miliknya, sehingga semuanya tetap konsisten.
- **Pakai `removeIf`.** Sejak Java 8, setiap koleksi memiliki `removeIf(predicate)`, yang menghapus semua elemen yang memenuhi syaratnya. `myList.removeIf(name -> name.length() <= 3)` menyatakan maksud kita dalam satu baris, dan pustaka mengerjakan urusan iterator dengan benar.
- **Iterasi atas sebuah salinan.** `for (String name : new ArrayList<>(myList))` menelusuri sebuah potret, sehingga list yang asli boleh diubah dengan bebas di dalam perulangan. Cara ini memerlukan sebuah salinan, tetapi inilah pilihan paling sederhana bila Anda harus *menambahkan* elemen selama penelusuran.

List Python bersikap lain: ia **tidak** memeriksa. **Kode 13.8** memperlihatkan artinya.

Kode 13.8 `concurrent_modification.py`: list Python yang diubah selama perulangan `for` diam-diam melompati elemen; dictionary dan set membangkitkan `RuntimeError`; tiga alternatif yang benar.

```
"""concurrent_modification.py: mengubah koleksi selagi menelusurinya."""


def crew():
    return ["Ace", "Luffy", "Sanji", "Zoro", "Nami"]


def main():
    # 1. Mengganti elemen lewat indeks selagi beriterasi: diizinkan.
    my_list = crew()
    for name in my_list:
        if name == "Luffy":
            my_list[2] = "Sabo"
        print(name, end=" ")
    print(" <- assignment is fine")

    # 2. Menghapus dari list selagi beriterasi: tanpa galat, hanya jawaban yang salah.
    numbers = [2, 4, 6, 7]
    for n in numbers:
        if n % 2 == 0:
            numbers.remove(n)
    print("remove evens while iterating:", numbers)

    # 3. Kesalahan yang sama dengan para awak: Nami tidak pernah dikunjungi.
    my_list = crew()
    seen = []
    for name in my_list:
        seen.append(name)
        if name == "Zoro":
            my_list.remove(name)
    print("visited only:", seen)

    # 4. Dictionary dan set memang memeriksa, dan membangkitkan RuntimeError.
    stock = {"soap": 3, "softener": 0, "bleach": 1}
    try:
        for item in stock:
            if stock[item] == 0:
                del stock[item]
    except RuntimeError as e:
        print("dict:", e)
    tags = {"wash", "iron"}
    try:
        for t in tags:
            tags.add(t.upper())
    except RuntimeError as e:
        print("set:", e)
```

Kode 13.8 (Lanjutan) *concurrent_modification.py: list Python yang diubah selama perulangan for diam-diam melompati elemen; dictionary dan set membangkitkan RuntimeError; tiga alternatif yang benar.*

```
    # 5. Cara-cara yang benar.
    numbers = [2, 4, 6, 7]
    numbers = [n for n in numbers if n % 2 != 0]       # bangun list baru
    print("comprehension:", numbers)

    my_list = crew()
    my_list[:] = [name for name in my_list if len(name) > 3]  # ganti isinya di tempat
    print("slice assignment:", my_list)

    stock = {"soap": 3, "softener": 0, "bleach": 1}
    for item in list(stock):                           # iterasi atas salinan kunci-
kuncinya
        if stock[item] == 0:
            del stock[item]
    print("copy of the keys:", stock)


if __name__ == "__main__":
    main()
```

```
Ace Luffy Sabo Zoro Nami  <- assignment is fine
remove evens while iterating: [4, 7]
visited only: ['Ace', 'Luffy', 'Sanji', 'Zoro']
dict: dictionary changed size during iteration
set: Set changed size during iteration
comprehension: [7]
slice assignment: ['Luffy', 'Sanji', 'Zoro', 'Nami']
copy of the keys: {'soap': 3, 'bleach': 1}
```

Blok 2 adalah jebakan yang termasyhur. Kita ingin menghapus bilangan genap dari `[2, 4, 6, 7]`, dan jawabannya semestinya `[7]`. Iterator list menyimpan sebuah indeks. Pada indeks 0 ia melihat 2 dan menghapusnya; list menjadi `[4, 6, 7]`; indeks bergeser ke 1, yang kini berisi **6** — jadi **4 terlewat**. 6 dihapus, list menjadi `[4, 7]`, indeks bergeser ke 2, yang sudah melewati ujung, dan perulangan berhenti. Hasil `[4, 7]` salah, dan Python tidak berkata apa-apa. Blok 3 adalah kesalahan yang sama dengan nama-nama: menghapus `Zoro` membuat `Nami` berpindah ke posisi yang sudah dilewati perulangan.

Dictionary dan set lebih ketat. Mengubah ukurannya selagi beriterasi membangkitkan `RuntimeError` dengan pesan `dictionary changed size during iteration` atau `Set changed size during iteration`, karena tabel hash yang membesar atau menyusut dapat menata ulang elemennya sepenuhnya (Subbab 13.9 menjelaskan sebabnya), dan melanjutkan penelusuran tidak akan bermakna lagi.

Obat-obat Python di blok 5 mencerminkan obat-obat Java: bangun **list baru** dengan *comprehension* (Subbab 13.10 menjelaskan sintaksnya); ganti isi list **di tempat** dengan penugasan irisan `my_list[:] = [...]`, yang mempertahankan objek list yang sama bagi siapa pun yang memegang referensi kepadanya; atau iterasi atas sebuah **salinan** seperti `list(stock)`, yaitu list kunci dictionary yang diambil sebelum perulangan dimulai. **Tabel 13.2** merangkum pola-pola yang aman.

Tabel 13.2 Cara-cara aman untuk mengubah koleksi yang sedang ditelusuri.

Tujuan selagi menelusuri	Java	Python
Mengganti sebuah elemen	`list.set(i, e)` atau `ListIterator.set(e)`	`lst[i] = e`
Menghapus elemen yang cocok	`it.remove()` dalam perulangan eksplisit, atau `c.removeIf(p)`	`lst[:] = [x for x in lst if keep(x)]`
Menambahkan elemen	perulangan atas `new ArrayList<>(list)`	perulangan atas `list(lst)` atau `lst[:]`
Menghapus entri dictionary	`map.entrySet().removeIf(p)`	perulangan atas `list(d)` lalu `del d[k]`
Akibatnya bila aturan dilanggar	biasanya `ConcurrentModificationException`; kadang lompatan diam-diam	list: lompatan diam-diam; dict dan set: `RuntimeError`

13.6 Mengurutkan Objek: Comparable dan Comparator

List itu berurutan, dan sering kali berguna untuk mengurutkan elemennya. `Collections.sort(list)` (atau `list.sort(...)`) di Java serta `list.sort()` dan `sorted(...)` di Python mengerjakannya, tetapi mereka perlu tahu *cara membandingkan dua elemen*. Tidak semua list dapat diurutkan: list string dapat, karena string memiliki urutan abjad, tetapi apa urutan sebuah list mahasiswa?

Java memberi sebuah kelas dua cara untuk menjawab. Jika kelas itu memiliki satu **urutan alami** yang disepakati semua orang — bilangan menurut besarnya, string menurut abjad, tanggal menurut waktu — ia mengimplementasikan antarmuka `Comparable<T>`, yang satu-satunya metodenya, `int compareTo(T other)`, membandingkan `this` dengan `other`. Bab 9 memakai antarmuka ini sebagai batas pada tanda tangan generik pengurutan:

```
public static <E extends Comparable<E>> void sort(List<E> list)
```

yang dibaca "`sort` menerima list bertipe apa pun `E`, asalkan `E` tahu cara membandingkan dirinya dengan `E` lainnya". (Batas sebenarnya di pustaka, `<T extends Comparable<? super T>>`, sedikit lebih umum, dengan alasan-alasan wildcard di Bab 9.)

Namun banyak kelas tidak memiliki satu urutan alami. Setiap mahasiswa memiliki nama depan, nama belakang, usia, program studi, nilai, dan sebagainya. Bagaimana sebuah list mahasiswa semestinya diurutkan? Bergantung pada informasi apa yang Anda perlukan dari list itu: menurut nama untuk daftar hadir, menurut usia untuk tim olahraga, menurut nilai untuk penghargaan. Menanamkan salah satunya ke dalam kelas dengan `Comparable` lebih banyak membingungkan daripada menolong. Itulah sebabnya antarmuka kedua ada: sebuah **`Comparator<T>`** adalah objek *terpisah* yang metodenya, `int compare(T a, T b)`, membandingkan dua objek dari luar [3]. Sebuah kelas boleh memiliki satu urutan alami dan comparator sebanyak apa pun.

13.6.1 Kontrak yang harus dipegang setiap perbandingan

Baik `compareTo` maupun `compare` mengembalikan sebuah `int` yang **tandanya** merupakan jawaban: negatif berarti "yang pertama berada sebelum yang kedua", nol berarti "keduanya setara dalam urutan ini", positif berarti "yang pertama berada sesudahnya". Hanya tandanya yang penting; `-1` dan `-37` bermakna sama. Algoritme pengurutan bergantung pada tiga aturan, yang oleh dokumentasi disebut kontrak umum. Dengan sgn(x) sebagai tanda x (-1, 0, atau 1) dan cmp sebagai perbandingannya, aturan-aturan itu adalah:

$$\operatorname{sgn}(\operatorname{cmp}(a,b)) = -\operatorname{sgn}(\operatorname{cmp}(b,a)) \quad \text{untuk semua } a, b \tag{13.3}$$

$$\operatorname{cmp}(a,b) > 0 \text{ dan } \operatorname{cmp}(b,c) > 0 \;\Rightarrow\; \operatorname{cmp}(a,c) > 0 \tag{13.4}$$

dan, yang ketiga, jika $\operatorname{cmp}(a,b) = 0$ maka a dan b dibandingkan dengan cara yang sama terhadap setiap elemen lain. Persamaan (13.3) menyatakan bahwa urutan itu *antisimetris*: jika a berada sebelum b, maka b berada sesudah a. Persamaan (13.4) menyatakan bahwa urutan itu *transitif*: jika a sesudah b dan b sesudah c, maka a sesudah c. Inilah sifat-sifat yang Anda anggap sudah semestinya ketika menyusun barisan, dan perbandingan yang melanggarnya membuat hasil pengurutan tidak bermakna [9].

Sebuah jalan pintas yang lazim melanggarnya. Untuk membandingkan dua usia, orang tergoda mengembalikan `s1.getAge() - s2.getAge()`: selisihnya negatif, nol, atau positif tepat pada saat yang semestinya. Untuk usia hal itu tidak berbahaya. Tetapi `int` di Java hanya menampung nilai dari -2^{31} sampai $2^{31} - 1$, dan selisih dua nilai besar dapat *meluap* (*overflow*):

$$2^{31} - 1 \;-\; (-1) \;=\; 2^{31} \overset{\text{luapan}}{\rightarrow} -2^{31} < 0 \tag{13.5}$$

Persamaan (13.5) memperlihatkan `int` terbesar dibandingkan dengan -1: selisih sebenarnya positif, tetapi selisih yang dihitung membungkus menjadi `int` paling negatif, sehingga perbandingan itu mengklaim bilangan terbesar *lebih kecil* daripada -1, melanggar Persamaan (13.3). Obatnya tidak berbiaya apa pun: `Integer.compare(a, b)` (juga `Long.compare`, `Double.compare`) mengembalikan tanda yang benar untuk setiap pasangan nilai.

13.6.2 Comparator berdasarkan usia

Kode 13.9 mengurutkan tiga mahasiswa menurut usia dengan `AgeComparator` yang ditulis dengan tangan, lalu dengan comparator yang dirakit dari komponen pustaka.

Kode 13.9 `AgeSort.java`: `AgeComparator` yang ditulis dengan tangan dan juga menangani `null`, serta urutan-urutan yang sama yang dirakit dengan `Comparator.comparingInt`, `reversed`, dan `nullsFirst`.

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.Iterator;
import java.util.List;

public class AgeSort {
    static final class Student {
        private final String name;
        private final int age;

        Student(String name, int age) {
            this.name = name;
            this.age = age;
        }

        String getName() {
            return name;
        }

        int getAge() {
            return age;
        }

        @Override
        public String toString() {
            return name + " (" + age + ")";
        }
    }

    // Comparator adalah objek terpisah yang tahu cara mengurutkan dua mahasiswa.
    static final class AgeComparator implements Comparator<Student> {
        @Override
        public int compare(Student s1, Student s2) {
            if (s1 == s2) {
                return 0;            // objek yang sama, atau keduanya null
            } else if (s1 == null) {
                return -1;           // s1 harus berada sebelum s2
            } else if (s2 == null) {
                return 1;            // s1 harus berada sesudah s2
            } else {
                return Integer.compare(s1.getAge(), s2.getAge());
            }
        }
    }
```

Kode 13.9 (Lanjutan) *AgeSort.java: AgeComparator yang ditulis dengan tangan dan juga menangani null, serta urutan-urutan yang sama yang dirakit dengan Comparator.comparingInt, reversed, dan nullsFirst.*

```
    public static void main(String[] args) {
        List<Student> students = new ArrayList<>();
        Student s = new Student("Chopper", 15);
        students.add(s);
        Student s1 = new Student("Nami", 20);
        students.add(s1);
        Student s2 = new Student("Usopp", 12);
        students.add(s2);
        Iterator<Student> itStudents = students.iterator();
        System.out.println("Print the list of student:");
        while (itStudents.hasNext()) {
            System.out.println(itStudents.next().getAge());
        }
        AgeComparator age = new AgeComparator();
        System.out.println();
        System.out.printf("Age comparison between s1 (age %d) and s2 (age %d) = %d%n",
                s1.getAge(), s2.getAge(), age.compare(s1, s2));
        System.out.println();
        System.out.println("After sort is called, the list of student becomes:");
        Collections.sort(students, age);
        for (Student st : students) {
            System.out.println(st.getAge());
        }

        // Urutan yang sama, disusun dari potongan kecil alih-alih satu kelas utuh.
        students.sort(Comparator.comparingInt(Student::getAge).reversed());
        System.out.println("oldest first: " + students);
        students.add(null);
        students.sort(Comparator.nullsFirst(Comparator.comparing(Student::getName)));
        System.out.println("by name, nulls first: " + students);
    }
}
```

```
Print the list of student:
15
20
12

Age comparison between s1 (age 20) and s2 (age 12) = 1

After sort is called, the list of student becomes:
12
15
20
oldest first: [Nami (20), Chopper (15), Usopp (12)]
by name, nulls first: [null, Chopper (15), Nami (20), Usopp (12)]
```

`AgeComparator.compare` menangani kasus-kasus khusus lebih dulu. `s1 == s2` mencakup "objek yang persis sama" sekaligus "keduanya `null`", dan mengembalikan 0. Jika hanya `s1` yang `null`, ia mengembalikan -1 agar `null` berada di depan; jika hanya `s2` yang `null`, ia mengembalikan 1. Selain itu ia mengembalikan `Integer.compare(s1.getAge(), s2.getAge())`. Untuk Nami (20) dan Usopp (12), hasilnya 1: dalam urutan usia Nami berada sesudah Usopp. (Jalan pintas pengurangan akan mengembalikan $20 - 12 = 8$, dengan tanda yang sama dan, untuk usia, efek yang sama.) `Collections.sort(students, age)` lalu menyusun usia dalam urutan 12, 15, 20.

Baris-baris terakhir memperlihatkan gaya modern. `Comparator.comparingInt(Student::getAge)` membangun comparator yang sama dari sebuah *pengekstrak kunci* — rujukan metode (*method reference*) yang menyatakan nilai mana yang dibandingkan — dan `.reversed()` membaliknya untuk "yang tertua lebih dulu". `Comparator.nullsFirst(Comparator.comparing(Student::getName))` membandingkan

menurut nama dan menaruh `null` di depan, yakni penanganan `null` milik `AgeComparator` dalam satu panggilan. Kombinator-kombinator ini ada justru agar tidak ada orang yang perlu menulis bagian-bagian yang rawan galat itu dengan tangan.

Pengurutan di Python menempuh jalan lain. Alih-alih fungsi pembanding yang melihat *dua* elemen, `sort` dan `sorted` di Python menerima **fungsi kunci** (*key function*) yang melihat *satu* elemen dan mengembalikan nilai yang dijadikan dasar pengurutan. Pengurutan menghitung setiap kunci sekali, lalu membandingkan kunci-kunci itu dengan `<`. **Kode 13.10** memperlihatkan kedua gaya.

Kode 13.10 `age_sort.py`: mengurutkan dengan fungsi kunci (`attrgetter`, `lambda`), `reverse=True`, dan comparator gaya Java yang diadaptasi dengan `functools.cmp_to_key`.

```
"""age_sort.py: mengurutkan objek dengan fungsi kunci, dan dengan comparator."""
from functools import cmp_to_key
from operator import attrgetter


class Student:
    def __init__(self, name, age):
        self._name = name
        self._age = age

    @property
    def name(self):
        return self._name

    @property
    def age(self):
        return self._age

    def __repr__(self):
        return f"{self._name} ({self._age})"


def age_compare(s1, s2):
    """Comparator gaya Java: negatif, nol, atau positif."""
    if s1 is s2:
        return 0
    if s1 is None:
        return -1
    if s2 is None:
        return 1
    return (s1.age > s2.age) - (s1.age < s2.age)


def main():
    s = Student("Chopper", 15)
    s1 = Student("Nami", 20)
    s2 = Student("Usopp", 12)
    students = [s, s1, s2]
    print("Print the list of student:")
    for st in students:
        print(st.age)
    print()
    print(f"Age comparison between s1 (age {s1.age}) and s2 (age {s2.age}) = "
          f"{age_compare(s1, s2)}")
    print()
    print("After sort is called, the list of student becomes:")
```

Kode 13.10 (Lanjutan) *age_sort.py: mengurutkan dengan fungsi kunci (attrgetter, lambda), reverse=True, dan comparator gaya Java yang diadaptasi dengan functools.cmp_to_key.*

```
    students.sort(key=attrgetter("age"))       # cara lazim Python: fungsi kunci
    for st in students:
        print(st.age)

    students.sort(key=lambda st: st.age, reverse=True)
    print("oldest first:", students)
    students.append(None)
    students.sort(key=cmp_to_key(lambda a, b: -1 if a is None else 1 if b is None
                                 else (a.name > b.name) - (a.name < b.name)))
    print("by name, None first:", students)


if __name__ == "__main__":
    main()
```

```
Print the list of student:
15
20
12

Age comparison between s1 (age 20) and s2 (age 12) = 1

After sort is called, the list of student becomes:
12
15
20
oldest first: [Nami (20), Chopper (15), Usopp (12)]
by name, None first: [None, Chopper (15), Nami (20), Usopp (12)]
```

`students.sort(key=attrgetter("age"))` mengurutkan menurut atribut `age`; `operator.attrgetter("age")` adalah fungsi siap pakai yang setara dengan `lambda st: st.age`. `reverse=True` menggantikan `.reversed()` milik Java. Bila Anda benar-benar memiliki perbandingan dua argumen — barangkali hasil porting dari Java — `functools.cmp_to_key` membungkusnya menjadi kunci; pengurutan terakhir memakainya untuk menaruh `None` di depan. Ekspresi `(a > b) - (a < b)` adalah idiom Python untuk "tanda perbandingan", karena `True` dan `False` dihitung sebagai 1 dan 0.

Fungsi kunci memiliki dua keunggulan. Ia lebih sulit dibuat keliru: sebuah kunci tidak mungkin melanggar Persamaan (13.3) atau Persamaan (13.4), karena perbandingan antarkunci dikerjakan oleh `<` milik Python sendiri. Dan ia lebih cepat: setiap kunci dihitung sekali per elemen, bukan sekali per perbandingan. Pengurutan Python juga **stabil**: elemen-elemen dengan kunci yang sama mempertahankan urutan relatif aslinya. Pengurutan objek di Java pun stabil. Kestabilan memungkinkan Anda mengurutkan menurut beberapa kriteria dalam beberapa lintasan — urutkan menurut nama depan, lalu (secara stabil) menurut nama belakang, dan orang-orang yang bernama belakang sama tetap terurut menurut nama depannya [10].

13.6.3 Nama dalam tiga urutan, dan perbandingan yang hanya tampak benar

Contoh berikutnya mengurutkan lima anggota sebuah keluarga fiksi yang termasyhur menurut nama lengkap dalam beberapa urutan, masing-masing untuk dua urutan awal yang berbeda ("Case 1" dan "Case 2"). Pengurutan yang benar harus memberikan hasil yang *sama* untuk kedua kasus, karena hasilnya hanya bergantung pada nama-namanya, bukan pada posisi awalnya. **Kode 13.11** dengan sengaja menyertakan `compareTo` dari jenis yang ternyata cukup sering ditulis orang.

Kode 13.11 `NameOrder.java`: `compareTo` yang hanya menangani −1, 0, dan 1, urutan alami yang benar, comparator nama-belakang-lalu-nama-depan, dan mengapa membandingkan string hasil rekatan itu rapuh.

```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Comparator;
import java.util.List;

public class NameOrder {
    // Urutan alami yang tampak benar padahal tidak: compareTo boleh mengembalikan int
berapa pun.
    static final class BrokenName implements Comparable<BrokenName> {
        final String firstName, lastName;

        BrokenName(String firstName, String lastName) {
            this.firstName = firstName;
            this.lastName = lastName;
        }

        @Override
        public int compareTo(BrokenName other) {
            int result = Integer.MIN_VALUE;
            switch (firstName.compareTo(other.firstName)) {
                case -1:
                    result = -1;
                    break;
                case 0:
                    result = lastName.compareTo(other.lastName);
                    break;
                case 1:
                    result = 1;
                    break;
            }
            return result;
        }

        @Override
        public String toString() {
            return firstName + " " + lastName.charAt(0) + ".";  // mis. "Luffy M."
        }
    }

    // Gagasan yang sama, dikerjakan dengan benar: teruskan tandanya, berapa pun besarnya.
    record Name(String firstName, String lastName) implements Comparable<Name> {
        @Override
        public int compareTo(Name other) {
            int byFirst = firstName.compareTo(other.firstName);
            return byFirst != 0 ? byFirst : lastName.compareTo(other.lastName);
        }
```

Kode 13.11 (Lanjutan) *NameOrder.java: compareTo yang hanya menangani -1, 0, dan 1, urutan alami yang benar, comparator nama-belakang-lalu-nama-depan, dan mengapa membandingkan string hasil rekatan itu rapuh.*

```
        @Override
        public String toString() {
            return firstName + " " + lastName.charAt(0) + ".";
        }
    }

    static final Comparator<Name> LAST_THEN_FIRST =
            Comparator.comparing(Name::lastName).thenComparing(Name::firstName);

    // Merekatkan kedua nama lalu membandingkan string hasil rekatan.
    static final Comparator<Name> CONCATENATED =
            (n1, n2) -> (n1.lastName() + n1.firstName()).compareTo(n2.lastName() +
n2.firstName());

    static String show(List<?> items) {
        List<String> parts = new ArrayList<>();
        for (Object item : items) {
            parts.add(item.toString());
        }
        return String.join(", ", parts);
    }

    public static void main(String[] args) {
        String[][] case1 = {{"Luffy", "Monkey D."}, {"Garp", "Monkey D."},
            {"Dragon", "Monkey D."}, {"Rouge", "Portgas D."}, {"Ace", "Portgas D."}};
        String[][] case2 = {{"Luffy", "Monkey D."}, {"Rouge", "Portgas D."},
            {"Garp", "Monkey D."}, {"Ace", "Portgas D."}, {"Dragon", "Monkey D."}};
        System.out.println("\"Luffy\".compareTo(\"Garp\") = " + "Luffy".compareTo("Garp"));

        int caseNo = 1;
        for (String[][] data : List.of(case1, case2)) {
            BrokenName[] broken = new BrokenName[data.length];
            List<Name> names = new ArrayList<>();
            for (int i = 0; i < data.length; i++) {
                broken[i] = new BrokenName(data[i][0], data[i][1]);
                names.add(new Name(data[i][0], data[i][1]));
            }
            System.out.println("Case " + caseNo++);
            System.out.println("  original order:   " + show(names));
            Arrays.sort(broken);
            System.out.println("  broken compareTo: " + show(Arrays.asList(broken)));
            names.sort(null);                       // null berarti: pakai compareTo
            System.out.println("  first, last:      " + show(names));
            names.sort(LAST_THEN_FIRST);
            System.out.println("  last, first:      " + show(names));
        }
```

Kode 13.11 (Lanjutan) *NameOrder.java: compareTo yang hanya menangani -1, 0, dan 1, urutan alami yang benar, comparator nama-belakang-lalu-nama-depan, dan mengapa membandingkan string hasil rekatan itu rapuh.*

```
        // Mengapa merekatkan string itu rapuh: spasi diurutkan sebelum huruf.
        List<Name> tricky = new ArrayList<>(List.of(
                new Name("Luffy", "Monkey D."), new Name("Zed", "Monkey")));
        tricky.sort(CONCATENATED);
        System.out.println("concatenated:     " + tricky.get(0).lastName() + " first");
        tricky.sort(LAST_THEN_FIRST);
        System.out.println("last, then first: " + tricky.get(0).lastName() + " first");
    }
}
```

```
"Luffy".compareTo("Garp") = 5
Case 1
  original order:   Luffy M., Garp M., Dragon M., Rouge P., Ace P.
  broken compareTo: Ace P., Rouge P., Dragon M., Garp M., Luffy M.
  first, last:      Ace P., Dragon M., Garp M., Luffy M., Rouge P.
  last, first:      Dragon M., Garp M., Luffy M., Ace P., Rouge P.
Case 2
  original order:   Luffy M., Rouge P., Garp M., Ace P., Dragon M.
  broken compareTo: Dragon M., Ace P., Garp M., Rouge P., Luffy M.
  first, last:      Ace P., Dragon M., Garp M., Luffy M., Rouge P.
  last, first:      Dragon M., Garp M., Luffy M., Ace P., Rouge P.
concatenated:     Monkey D. first
last, then first: Monkey first
```

Perhatikan `BrokenName.compareTo`. Metode ini membandingkan nama depan dengan `String.compareTo` lalu memakai `switch` dengan kasus `-1`, `0`, dan `1`, serta mengembalikan `Integer.MIN_VALUE` untuk nilai lainnya. Siapa pun yang menulis kode seperti itu percaya bahwa `compareTo` hanya mengembalikan -1, 0, atau 1. Kenyataannya tidak. Baris pertama keluaran memperlihatkan `"Luffy".compareTo("Garp") = 5`: `String.compareTo` mengembalikan selisih antara dua karakter pertama yang berbeda, di sini $'L' - 'G' = 76 - 71 = 5$. Jadi untuk hampir setiap pasangan nama, `switch` jatuh ke `Integer.MIN_VALUE`, bilangan negatif yang berarti "lebih kecil" — apa pun urutan argumennya. Perbandingan itu menyatakan $a < b$ *dan* $b < a$, yang melanggar Persamaan (13.3).

Apa yang dilakukan pengurutan dengan perbandingan semacam itu? Pengurutan objek di Java adalah sebuah *Timsort*, yang dimulai dengan mencari sebuah **deretan** (*run*) — bentangan larik yang sudah terurut. Ia bertanya apakah elemen kedua lebih kecil daripada elemen pertama; jika ya, ia terus memperpanjang deretan yang *menurun tegas* selama setiap elemen "lebih kecil" daripada elemen sebelumnya, lalu membalik deretan itu di tempat. Karena perbandingan yang rusak menjawab "lebih kecil" untuk setiap pertanyaan, seluruh larik tampak sebagai satu deretan menurun yang panjang, dan larik "terurut" itu tidak lain adalah **masukan yang dibalik**. Bandingkan dengan keluarannya: Case 1 dimulai sebagai Luffy, Garp, Dragon, Rouge, Ace, dan "broken compareTo" mencetak Ace, Rouge, Dragon, Garp, Luffy — persis terbalik. Case 2 dimulai dengan urutan lain dan keluar dengan urutan yang lain pula (lagi-lagi terbalik). Itulah tanda khas perbandingan yang rusak: **hasilnya bergantung pada urutan masukan**. Pada larik yang lebih besar, pengurutan Java bahkan dapat mendeteksi ketidakkonsistenan itu dan melemparkan `IllegalArgumentException: Comparison method violates its general contract!`.

Record `Name` yang benar memperbaikinya dengan cara paling sederhana: ia meneruskan hasil perbandingan pertama, berapa pun besarnya, dan hanya bila hasil itu nol ia membandingkan nama belakang. Kedua kasus kini memberikan Ace, Dragon, Garp, Luffy, Rouge. `LAST_THEN_FIRST` mengurutkan menurut nama belakang lalu nama depan dengan `Comparator.comparing(...).thenComparing(...)`, menghasilkan keluarga Monkey D. sebelum keluarga Portgas D., masing-masing terurut menurut nama depan, lagi-lagi sama untuk kedua kasus.

Baris-baris terakhir membahas sebuah jalan pintas yang menggoda: rekatkan nama belakang dan nama depan menjadi satu string lalu bandingkan string rekatan itu. Biasanya cara ini berhasil, tetapi tidak selalu. `"Monkey D." + "Luffy"` adalah `"Monkey D.Luffy"` dan `"Monkey" + "Zed"` adalah `"MonkeyZed"`. Keduanya sama pada enam huruf pertama; lalu sebuah spasi (karakter 32) bertemu `Z` (karakter 90), dan spasi lebih kecil, sehingga perbandingan rekatan menaruh Luffy dari keluarga "Monkey D." di depan — padahal nama belakang `"Monkey"` adalah awalan dari

`"Monkey D."` dan karena itu berada lebih dulu dalam urutan nama-belakang-lalu-nama-depan yang benar. Membandingkan field demi field tidak pernah mengalami masalah ini.

Kode 13.12 mengerjakan hal yang sama dalam Python.

Kode 13.12 `name_order.py`: `@dataclass(order=True)` untuk urutan alami field demi field, kunci tuple untuk urutan lain, dan perbandingan rusak yang direproduksi dengan `cmp_to_key`.

```
"""name_order.py: mengurutkan nama dengan beberapa kunci, dan comparator yang rusak."""
from dataclasses import dataclass
from functools import cmp_to_key


@dataclass(frozen=True, order=True)       # order=True membandingkan field demi field
class Name:
    first_name: str
    last_name: str

    def __str__(self):
        return f"{self.first_name} {self.last_name[0]}."


def java_compare(a, b):
    """String.compareTo milik Java: selisih karakter pertama yang berbeda."""
    for x, y in zip(a, b):
        if x != y:
            return ord(x) - ord(y)
    return len(a) - len(b)


def broken_compare(n1, n2):
    """Kesalahan yang sama dengan BrokenName di Java: hanya -1, 0, dan 1 yang ditangani."""
    result = -(2 ** 31)
    c = java_compare(n1.first_name, n2.first_name)
    if c == -1:
        result = -1
    elif c == 0:
        result = java_compare(n1.last_name, n2.last_name)
    elif c == 1:
        result = 1
    return result


def show(items):
    return ", ".join(str(x) for x in items)


def main():
    case1 = [("Luffy", "Monkey D."), ("Garp", "Monkey D."), ("Dragon", "Monkey D."),
             ("Rouge", "Portgas D."), ("Ace", "Portgas D.")]
    case2 = [("Luffy", "Monkey D."), ("Rouge", "Portgas D."), ("Garp", "Monkey D."),
             ("Ace", "Portgas D."), ("Dragon", "Monkey D.")]
    print('java_compare("Luffy", "Garp") =', java_compare("Luffy", "Garp"))
```

Kode 13.12 (Lanjutan) *name_order.py: @dataclass(order=True) untuk urutan alami field demi field, kunci tuple untuk urutan lain, dan perbandingan rusak yang direproduksi dengan cmp_to_key.*

```
    for case_no, data in enumerate([case1, case2], start=1):
        names = [Name(f, l) for f, l in data]
        print(f"Case {case_no}")
        print("  original order:  ", show(names))
        print("  broken compare:  ", show(sorted(names, key=cmp_to_key(broken_compare))))
        print("  first, last:     ", show(sorted(names)))
        print("  last, first:     ",
              show(sorted(names, key=lambda n: (n.last_name, n.first_name))))

    tricky = [Name("Luffy", "Monkey D."), Name("Zed", "Monkey")]
    glued = sorted(tricky, key=lambda n: n.last_name + n.first_name)
    print("concatenated:    ", glued[0].last_name, "first")
    proper = sorted(tricky, key=lambda n: (n.last_name, n.first_name))
    print("last, then first:", proper[0].last_name, "first")


if __name__ == "__main__":
    main()
```

```
java_compare("Luffy", "Garp") = 5
Case 1
  original order:   Luffy M., Garp M., Dragon M., Rouge P., Ace P.
  broken compare:   Ace P., Rouge P., Dragon M., Garp M., Luffy M.
  first, last:      Ace P., Dragon M., Garp M., Luffy M., Rouge P.
  last, first:      Dragon M., Garp M., Luffy M., Ace P., Rouge P.
Case 2
  original order:   Luffy M., Rouge P., Garp M., Ace P., Dragon M.
  broken compare:   Dragon M., Ace P., Garp M., Rouge P., Luffy M.
  first, last:      Ace P., Dragon M., Garp M., Luffy M., Rouge P.
  last, first:      Dragon M., Garp M., Luffy M., Ace P., Rouge P.
concatenated:     Monkey D. first
last, then first: Monkey first
```

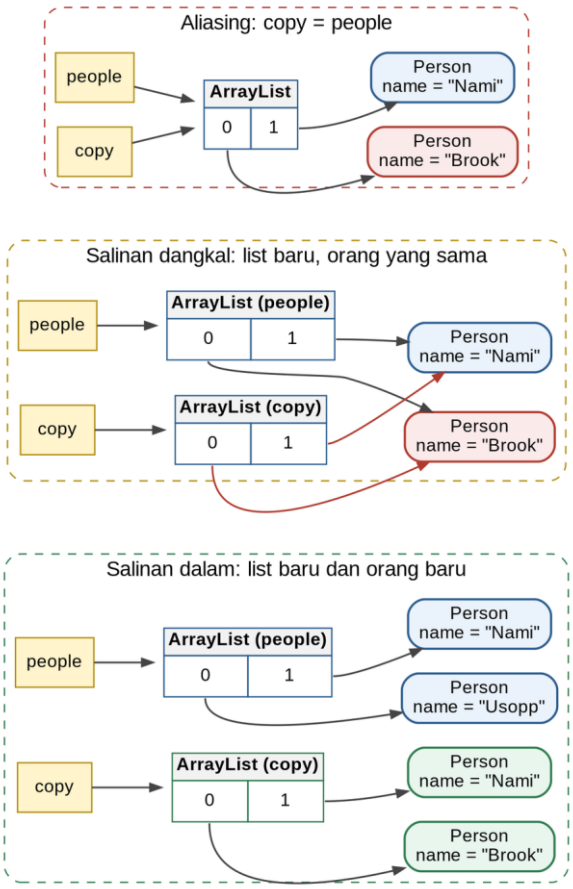
`@dataclass(frozen=True, order=True)` meminta Python membangkitkan `<`, `<=`, `>`, dan `>=` yang membandingkan field sesuai urutan deklarasinya, pertama `first_name` lalu `last_name` — persis `compareTo` yang benar pada record Java, kali ini ditulis oleh pustaka. Untuk urutan lain kita mengembalikan sebuah **tuple** dari fungsi kunci: `key=lambda n: (n.last_name, n.first_name)`. Tuple dibandingkan field demi field, dari kiri ke kanan, sehingga kunci tuple adalah ejaan Python untuk `comparing(...).thenComparing(...)`. Untuk mereproduksi perbandingan yang rusak secara setia, `java_compare` meniru `String.compareTo` (perbandingan string bawaan Python hanya mengembalikan `True` atau `False`), dan `cmp_to_key(broken_compare)` menyuapkannya kepada `sorted`. Hasilnya adalah masukan terbalik yang sama, dengan alasan yang sama: pengurutan Python adalah Timsort yang asli, dan ia pun mulai dengan mendeteksi deretan menurun lalu membaliknya [10].

13.7 Salinan: Aliasing, Dangkal, dan Dalam

Dalam pemrograman, sering kali berguna untuk membuat salinan sebuah objek — misalnya list berisi orang — agar satu bagian program dapat mengubah salinannya tanpa mengusik yang asli. Pada list berisi objek, menyalin memunculkan pertanyaan yang mengejutkan hampir setiap pemula: **seberapa dalam salinan itu?**

Perhatikan urutan berikut. Kita membangun list `people` yang berisi dua objek `Person`, Usopp dan Nami; kita membuat `copy` darinya; kita mengambil orang pertama dari salinan itu dan mengganti namanya menjadi Brook. Seperti apa `copy`? Jelas `[Brook, Nami]`. Seperti apa list *asli*-nya? Kebanyakan orang menduga `[Usopp, Nami]`. Namun `[Brook, Nami]` sama mungkinnya —

semuanya bergantung pada jenis salinan yang dibuat. Ada tiga tingkat, dan **Gambar 13.3** menggambarkannya.



Gambar 13.3 Tiga jenis "salinan" dari list berisi orang. Aliasing: dua variabel, satu list. Salinan dangkal: dua list, tetapi slot-slotnya merujuk ke objek `Person` yang sama, sehingga penggantian nama lewat satu list terlihat lewat list yang lain. Salinan dalam: dua list dan objek `Person` yang terpisah.

> **DEFINISI. Aliasing** berarti memiliki dua referensi ke objek yang sama; tidak ada yang disalin. **Salinan dangkal** (*shallow copy*) sebuah koleksi adalah objek koleksi baru yang memegang *referensi elemen yang sama* dengan aslinya: urutan objeknya disalin, tetapi objek-objeknya sendiri dipakai bersama. **Salinan dalam** (*deep copy*) menyalin koleksinya *dan* setiap elemennya (serta, secara rekursif, apa pun yang dirujuk elemen-elemen itu), sehingga salinan tidak berbagi apa pun yang mutabel dengan aslinya.

Kode 13.13 menjalankan percobaan yang sama pada ketiga tingkat. Pada setiap percobaan, program memeriksa dengan `==` apakah kedua list dan elemen pertamanya adalah objek yang sama, mengganti nama orang pertama pada salinan menjadi Brook, menambahkan Franky hanya ke salinan, lalu mencetak kedua list.

Kode 13.13 `CopyDemo.java`: percobaan ganti-nama-dan-tambah yang sama pada alias, salinan dangkal, dan salinan dalam dari list berisi objek `Person`.

```java
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

public class CopyDemo {
    static final class Person {
        private String name;

        Person(String name) {
            this.name = name;
        }

        String getName() {
            return name;
        }

        void setName(String name) {
            this.name = name;
        }

        @Override
        public String toString() {
            return "[" + name + "]";
        }
    }

    // Dangkal: list baru, tetapi objek Person di dalamnya tetap sama.
    private static List<Person> copyList(List<Person> people) {
        List<Person> result = new ArrayList<Person>(people);
        Collections.copy(result, people);   // menyalin referensi, bukan orangnya
        return result;
    }

    // Dalam: list baru DAN Person baru untuk setiap elemen.
    private static List<Person> deepCopy(List<Person> people) {
        List<Person> result = new ArrayList<>(people);
        for (int i = 0; i < people.size(); i++) {
            result.set(i, new Person(people.get(i).getName()));
        }
        return result;
    }
```

Kode 13.13 (Lanjutan) *CopyDemo.java: percobaan ganti-nama-dan-tambah yang sama pada alias, salinan dangkal, dan salinan dalam dari list berisi objek Person.*

```
    static List<Person> crew() {
        List<Person> people = new ArrayList<>();
        people.add(new Person("Usopp"));
        people.add(new Person("Nami"));
        return people;
    }

    static void experiment(String title, List<Person> people, List<Person> copy) {
        System.out.println(title);
        System.out.println("  same list object?   " + (copy == people));
        System.out.println("  same first person?  " + (copy.get(0) == people.get(0)));
        copy.get(0).setName("Brook");        // Ubah list salinan: Usopp -> Brook
        copy.add(new Person("Franky"));      // dan tambahkan seseorang ke salinannya saja
        System.out.println("  the copied list:    " + copy);
        System.out.println("  the original list:  " + people);
    }

    public static void main(String[] args) {
        List<Person> people = crew();
        experiment("Aliasing (no copy at all)", people, people);
        people = crew();
        experiment("Shallow copying", people, copyList(people));
        people = crew();
        experiment("Deep copying", people, deepCopy(people));
    }
}
```

```
Aliasing (no copy at all)
  same list object?   true
  same first person?  true
  the copied list:    [[Brook], [Nami], [Franky]]
  the original list:  [[Brook], [Nami], [Franky]]
Shallow copying
  same list object?   false
  same first person?  true
  the copied list:    [[Brook], [Nami], [Franky]]
  the original list:  [[Brook], [Nami]]
Deep copying
  same list object?   false
  same first person?  false
  the copied list:    [[Brook], [Nami], [Franky]]
  the original list:  [[Usopp], [Nami]]
```

- **Aliasing.** `experiment(..., people, people)` menyerahkan list yang sama dua kali. `copy == people` bernilai `true`: hanya ada satu list dengan dua nama, sehingga penggantian nama maupun Franky yang ditambahkan muncul “di kedua list”.
- **Salinan dangkal.** `copyList` membuat `ArrayList` baru dari `people`. (Pemanggilan `Collections.copy(result, people)` sesudahnya menyalin referensi elemen sekali lagi ke list baru; tidak berbahaya tetapi mubazir, karena konstruktornya sudah melakukan hal itu — patut diperhatikan ketika Anda membaca kode lama.) Kini `copy == people` bernilai `false`: ada dua list, dan menambahkan Franky ke salah satunya tidak memengaruhi yang lain. Tetapi `copy.get(0) == people.get(0)` bernilai `true`: kedua list merujuk ke objek Usopp yang *sama*, sehingga mengganti namanya lewat salinan juga mengganti namanya di list asli. List asli menjadi `[Brook, Nami]`.
- **Salinan dalam.** `deepCopy` juga membuat list baru, lalu mengganti setiap slot dengan `new Person(people.get(i).getName())` — objek baru dengan data yang sama. Kini tidak ada yang dipakai bersama, dan list asli tetap `[Usopp, Nami]`.

Tidak ada jenis salinan yang merupakan "yang benar". Salinan dangkal itu murah dan benar setiap kali elemennya **imutabel** (string, bilangan, objek nilai di Subbab 13.9): memakai bersama objek yang tidak dapat berubah sepenuhnya aman. Salinan dalam diperlukan bila elemennya mutabel dan salinannya harus mandiri. Perhatikan bagaimana imutabilitas menyederhanakan pertanyaan ini: seandainya `Person` tidak dapat diganti namanya, salinan dangkal dan salinan dalam tidak akan dapat dibedakan. Pengamatan itulah jembatan kita ke Subbab 13.8.

Java tidak memiliki salinan dalam yang serba guna. Setiap kelas harus memutuskan apa arti menyalin baginya — *konstruktor salinan* seperti `new Person(other)` adalah alat yang lazim — karena hanya kelas itu yang tahu field mana yang berupa nilai dan mana yang berupa referensi ke hal yang sengaja dipakai bersama [9]. Python memilikinya, dalam modul standar `copy`, seperti diperlihatkan **Kode 13.14**.

Kode 13.14 `copy_demo.py`: aliasing, `list(...)` sebagai salinan dangkal, `copy.deepcopy`, dan tiga tingkat yang sama untuk list bersarang.

```
"""copy_demo.py: aliasing, salinan dangkal, dan salinan dalam."""
import copy


class Person:
    def __init__(self, name):
        self.name = name

    def __repr__(self):
        return f"[{self.name}]"


def crew():
    return [Person("Usopp"), Person("Nami")]


def experiment(title, people, copied):
    print(title)
    print("  same list object?  ", copied is people)
    print("  same first person? ", copied[0] is people[0])
    copied[0].name = "Brook"             # Ubah list salinan: Usopp -> Brook
    copied.append(Person("Franky"))      # dan tambahkan seseorang ke salinannya saja
    print("  the copied list:   ", copied)
    print("  the original list: ", people)


def main():
    people = crew()
    experiment("Aliasing (no copy at all)", people, people)
    people = crew()
    experiment("Shallow copying", people, list(people))      # atau copy.copy(people)
    people = crew()
    experiment("Deep copying", people, copy.deepcopy(people))

    # List bersarang memperlihatkan tiga tingkat yang sama.
    grid = [[1, 2], [3, 4]]
    shallow, deep = copy.copy(grid), copy.deepcopy(grid)
    grid[0][0] = 99
    print("grid:", grid, " shallow:", shallow, " deep:", deep)


if __name__ == "__main__":
    main()
```

```
Aliasing (no copy at all)
  same list object?   True
  same first person?  True
  the copied list:    [[Brook], [Nami], [Franky]]
  the original list:  [[Brook], [Nami], [Franky]]
Shallow copying
  same list object?   False
  same first person?  True
  the copied list:    [[Brook], [Nami], [Franky]]
  the original list:  [[Brook], [Nami]]
Deep copying
  same list object?   False
  same first person?  False
  the copied list:    [[Brook], [Nami], [Franky]]
  the original list:  [[Usopp], [Nami]]
grid: [[99, 2], [3, 4]]  shallow: [[99, 2], [3, 4]]  deep: [[1, 2], [3, 4]]
```

`list(people)` (atau `people.copy()`, `people[:]`, dan `copy.copy(people)`, yang semuanya bermakna sama untuk list) membuat salinan dangkal; `copy.deepcopy(people)` menelusuri seluruh struktur dan menyalin setiap objek yang dijumpainya, sambil mencatat objek yang sudah disalin agar referensi bersama dan siklus tersalin dengan benar [4]. Baris terakhir memperlihatkan tiga tingkat yang sama pada list berisi list, struktur yang sering dibangun pemula untuk grid dan tabel: mengubah `grid[0][0]` tampak melalui salinan dangkal, karena kedua list luar memakai bersama list-list dalamnya, sedangkan salinan dalam tidak terpengaruh.

13.8 Imutabilitas dari Nol

Meskipun mungkin tidak kita sadari, kita sudah memakai objek imutabel sejak Bab 3.

> **DEFINISI.** Sebuah objek bersifat **imutabel** (*immutable*) jika keadaannya tidak dapat berubah setelah objek itu diciptakan. Objek yang keadaannya dapat berubah bersifat **mutabel** (*mutable*). (Ingatlah bahwa *objek* berbeda dari *variabel*: variabel yang merujuk ke objek imutabel tetap dapat dibuat merujuk ke objek lain.)

`String` adalah contoh sehari-harinya. Begitu sebuah string diciptakan, ia tidak dapat diubah; setiap metode yang tampak mengubahnya — `toUpperCase()`, `trim()`, `replace(...)` — mengembalikan string *baru* dan membiarkan yang asli tak tersentuh. Kita tetap dapat membuat string baru kapan pun diperlukan, dan variabel seperti `day` dalam `day = day.toUpperCase()` dapat dibuat merujuk ke string yang baru; tetapi string `"monday"` itu sendiri tetap `"monday"` selamanya. Hal yang sama berlaku bagi `Integer`, `Long`, `Double`, `LocalDate`, dan kelas-kelas tanggal-waktu lainnya di Java, serta bagi `int`, `float`, `str`, `tuple`, dan `frozenset` di Python. Sebaliknya, objek mutabel — `ArrayList`, `HashMap`, `StringBuilder`, `list`, `dict`, dan `set` milik Python, serta setiap kelas yang pernah kita tulis dengan setter — dapat diubah keadaannya setelah diciptakan.

13.8.1 Mengapa ada yang menginginkan objek yang tidak dapat berubah?

Dokumentasi Java menyatakannya dengan tegas: “Maximum reliance on immutable objects is widely accepted as a sound strategy for creating simple, reliable code” — sebanyak mungkin bersandar pada objek imutabel diterima luas sebagai strategi yang sehat untuk menciptakan kode yang sederhana dan andal [11]. Mengapa demikian? Manfaat apa yang mungkin timbul dari *tidak* diizinkannya kita mengubah sebuah objek? Ada empat, dan masing-masing layak mendapat kalimatnya sendiri.

1. Objek imutabel adalah kunci yang baik. Bab 9 memperingatkan bahwa sebuah objek tidak boleh berubah selagi dipakai sebagai kunci di dalam map atau sebagai elemen sebuah set. Objek imutabel *tidak dapat* berubah, sehingga peringatan itu terpenuhi dengan sendirinya. Subbab 13.9 memperlihatkan dengan persis apa yang rusak bila aturan itu dilanggar.

2. Kita tidak perlu khawatir tentang apa yang terjadi ketika sebuah objek diserahkan kepada sebuah metode. Misalkan kita menyerahkan sebuah list kepada metode bernama `medianAverage`. Untuk mencari median — nilai tengah — metode itu harus mengurutkan nilai-nilainya. Ia mungkin mengurutkan list *milik kita*, atau membuat salinannya sendiri lalu mengurutkan salinan itu. Perilaku pertama tidak diinginkan, karena urutan list kita mungkin bermakna. Dengan string hal ini tidak pernah menjadi masalah: metode itu dapat menciptakan string baru berdasarkan parameter string, tetapi tidak dapat mengubah string yang kita berikan. **Kode 13.15** memperlihatkan masalahnya dengan list kilogram cucian per hari, yang urutannya *justru* merupakan informasi.

Kode 13.15 `MedianDemo.java`: metode median yang mengurutkan list milik pemanggilnya, metode yang mengurutkan salinan, dan tampilan tak dapat diubah yang mengubah efek samping tersembunyi menjadi eksepsi.

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

public class MedianDemo {
    // Mengurutkan list pemanggil untuk mencari nilai tengah: efek samping tersembunyi.
    static double medianAverageCareless(List<Integer> values) {
        Collections.sort(values);
        return middle(values);
    }

    // Mengurutkan salinannya sendiri: list pemanggil dibiarkan persis seperti semula.
    static double medianAverage(List<Integer> values) {
        List<Integer> sorted = new ArrayList<>(values);
        Collections.sort(sorted);
        return middle(sorted);
    }

    private static double middle(List<Integer> sorted) {
        int n = sorted.size();
        if (n == 0) {
            throw new IllegalArgumentException("no values");
        }
        return n % 2 == 1 ? sorted.get(n / 2)
                          : (sorted.get(n / 2 - 1) + sorted.get(n / 2)) / 2.0;
    }

    public static void main(String[] args) {
        // Kilogram yang dicuci per hari, Senin sampai Sabtu: URUTAN itulah informasinya.
        List<Integer> kgPerDay = new ArrayList<>(List.of(42, 35, 51, 38, 60, 47));
        System.out.println("week:    " + kgPerDay);
        System.out.println("median:  " + medianAverageCareless(kgPerDay));
        System.out.println("week:    " + kgPerDay + "   <- which day was Monday?");

        kgPerDay = new ArrayList<>(List.of(42, 35, 51, 38, 60, 47));
        System.out.println("median:  " + medianAverage(kgPerDay));
        System.out.println("week:    " + kgPerDay + "   <- untouched");
```

Kode 13.15 (Lanjutan) *MedianDemo.java: metode median yang mengurutkan list milik pemanggilnya, metode yang mengurutkan salinan, dan tampilan tak dapat diubah yang mengubah efek samping tersembunyi menjadi eksepsi.*

```
        // Tampilan yang tak dapat diubah mengubah efek samping tersembunyi menjadi galat
yang lantang.
        List<Integer> guarded = Collections.unmodifiableList(kgPerDay);
        try {
            medianAverageCareless(guarded);
        } catch (UnsupportedOperationException e) {
            System.out.println("careless median on a guarded list: "
                    + e.getClass().getSimpleName());
        }

        // String imutabel: sebuah metode hanya dapat membangun string BARU darinya.
        String day = "monday";
        String shout = day.toUpperCase();
        System.out.println(day + " " + shout);
    }
}
```

```
week:    [42, 35, 51, 38, 60, 47]
median:  44.5
week:    [35, 38, 42, 47, 51, 60]   <- which day was Monday?
median:  44.5
week:    [42, 35, 51, 38, 60, 47]   <- untouched
careless median on a guarded list: UnsupportedOperationException
monday MONDAY
```

`medianAverageCareless` mengembalikan median yang benar, 44.5 (rata-rata nilai tengah 42 dan 47), tetapi sesudahnya minggu itu sudah terurut dan tidak ada yang dapat mengatakan hari mana yang Senin. **Efek samping** (*side effect*) semacam itu pada sebuah argumen sulit ditemukan, karena metode yang merusak list itu mengembalikan jawaban yang benar. `medianAverage` mengurutkan salinannya sendiri dan membiarkan list pemanggil. Dan blok terakhir memperlihatkan langkah defensif dari sisi pemanggil: serahkan sebuah *tampilan yang tak dapat diubah* (Subbab 13.8.2), dan metode ceroboh yang mencoba mengurutkannya gagal dengan lantang melalui `UnsupportedOperationException` alih-alih diam-diam merusak data. **Kode 13.16** menuturkan kisah yang sama dalam Python.

Kode 13.16 `median_demo.py`: `list.sort()` mengubah list milik pemanggil, `sorted()` tidak, dan tuple sama sekali tidak dapat diurutkan di tempat.

```
"""median_demo.py: fungsi yang mengubah argumennya, dan yang tidak."""


def median_average_careless(values):
    values.sort()                       # mengurutkan list pemanggil: efek samping
tersembunyi
    return _middle(values)


def median_average(values):
    return _middle(sorted(values))      # sorted() membangun list baru


def _middle(ordered):
    n = len(ordered)
    if n == 0:
        raise ValueError("no values")
    if n % 2 == 1:
        return ordered[n // 2]
    return (ordered[n // 2 - 1] + ordered[n // 2]) / 2


def main():
    # Kilogram yang dicuci per hari, Senin sampai Sabtu: URUTAN itulah informasinya.
    kg_per_day = [42, 35, 51, 38, 60, 47]
    print("week:   ", kg_per_day)
    print("median: ", median_average_careless(kg_per_day))
    print("week:   ", kg_per_day, "  <- which day was Monday?")

    kg_per_day = [42, 35, 51, 38, 60, 47]
    print("median: ", median_average(kg_per_day))
    print("week:   ", kg_per_day, "  <- untouched")

    # Tuple tidak dapat diurutkan di tempat, maka versi ceroboh gagal dengan lantang.
    guarded = tuple(kg_per_day)
    try:
        median_average_careless(guarded)
    except AttributeError as e:
        print("careless median on a tuple:", e)
    print("median of the tuple:", median_average(guarded))

    day = "monday"
    shout = day.upper()
    print(day, shout)
```

Kode 13.16 (Lanjutan) *median_demo.py: list.sort() mengubah list milik pemanggil, sorted() tidak, dan tuple sama sekali tidak dapat diurutkan di tempat.*

```
if __name__ == "__main__":
    main()
```

```
week:    [42, 35, 51, 38, 60, 47]
median:  44.5
week:    [35, 38, 42, 47, 51, 60]   <- which day was Monday?
median:  44.5
week:    [42, 35, 51, 38, 60, 47]   <- untouched
careless median on a tuple: 'tuple' object has no attribute 'sort'
median of the tuple: 44.5
monday MONDAY
```

Python memperlihatkan perbedaannya lewat nama: *metode* `values.sort()` mengurutkan di tempat dan mengembalikan `None`, sedangkan *fungsi* `sorted(values)` membangun dan

mengembalikan list baru. Tuple sama sekali tidak memiliki metode `sort`, sehingga menyerahkan sebuah tuple mengubah versi yang ceroboh menjadi `AttributeError` seketika.

3. Objek imutabel dapat dipakai bersama oleh beberapa thread. Bab 14 memperkenalkan *thread*: potongan kode yang berjalan bersamaan dengan program utama. Jika dua thread dapat mengubah objek yang sama, perubahan-perubahan mereka dapat saling menyela dan merusaknya. Analogi sederhananya adalah lembar kerja nilai bersama yang dibuka dua petugas pada saat yang sama. Petugas A selesai lebih dulu dan menyimpan; petugas B selesai semenit kemudian dan menyimpan salinannya, menimpa perubahan A, sehingga lembar itu tertinggal dalam keadaan rusak. Objek imutabel ibarat dokumen tercetak yang sudah ditandatangani: berapa pun banyaknya orang dapat membacanya pada saat yang sama, dan tak seorang pun dapat mencoretinya. Memakai bersama objek imutabel di antara thread sama sekali tidak memerlukan kunci (*lock*) [12].

4. Objek imutabel itu sederhana. Objek imutabel hanya memiliki satu keadaan, yaitu keadaan saat ia lahir. Jika konstruktornya memeriksa bahwa keadaan itu sah, keadaan itu sah selamanya; tidak ada rangkaian panggilan metode yang dapat merusaknya, sehingga tidak ada rangkaian semacam itu yang perlu diuji. Bab 7 menyebut janji jenis ini *invarian*; objek imutabel memelihara invariannya secara cuma-cuma [9].

Para pemrogram kadang khawatir bahwa menciptakan objek baru alih-alih mengubah yang lama itu mahal. Memang ada biaya tambahan, tetapi mesin virtual modern menciptakan dan membuang objek kecil yang berumur pendek dengan sangat efisien, dan biayanya terbayar oleh salinan-salinan yang tidak perlu lagi kita buat: objek imutabel tidak pernah memerlukan salinan defensif.

Mutabilitas tetap merupakan pilihan yang tepat untuk sebagian hal. **Rekening bank** ada justru untuk mengubah saldonya dari waktu ke waktu; **mesin cuci** memiliki pintu yang terbuka dan muatan yang bertambah. Objek semacam itu memiliki *identitas* — mesin yang ini, bukan mesin lain yang setara — dan riwayat hidup. Pembedaan antara objek yang ditentukan oleh identitasnya dan objek yang ditentukan oleh nilainya adalah tema Subbab 13.9, dan Soal Latihan 13.7 meminta Anda menggolongkan beberapa kelas.

13.8.2 final bukan berarti imutabel; tampilan dan salinan

Alat pertama Java untuk imutabilitas adalah kata kunci `final`. Pada sebuah field atau variabel, `final` memberi tahu kompilator bahwa nilainya hanya boleh ditugaskan sekali — untuk field, di deklarasinya atau di konstruktor — dan tidak pernah diubah sesudahnya. Itulah cara paling kokoh untuk menegakkan imutabilitas dalam sebuah kelas; cara paling sederhana adalah tidak menyediakan metode setter, dan kelas yang cermat melakukan keduanya.

Namun `final` mengunci *variabelnya*, bukan *objek yang dirujuknya*. Sebuah `final List<String>` selalu merujuk ke list yang sama, dan list itu tetap dapat membesar dan menyusut. Untuk koleksi, Java karena itu menawarkan dua alat terpisah yang mudah tertukar, dan Python memiliki padanan untuk keduanya. **Kode 13.17** memaparkannya.

Kode 13.17 `ImmutableCollections.java`: `final` versus imutabel, tampilan tak dapat diubah yang memperlihatkan perubahan belakangan, salinan tak dapat diubah yang tidak, metode pabrik `of`, dan ketakterubahan yang dangkal.

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import java.util.Map;
import java.util.Set;

public class ImmutableCollections {
    static void attempt(String what, Runnable action) {
        try {
            action.run();
            System.out.println(what + ": allowed");
        } catch (RuntimeException e) {
            System.out.println(what + ": " + e.getClass().getSimpleName());
        }
    }

    public static void main(String[] args) {
        // 1. final mengunci VARIABEL, bukan objek yang dirujuknya.
        final List<String> services = new ArrayList<>(List.of("WF", "IR"));
        services.add("DC");                      // diizinkan: list-nya sendiri mutabel
        // services = new ArrayList<>();         // tidak akan lolos kompilasi: services
bersifat final
        System.out.println("final list after add: " + services);

        // 2. TAMPILAN yang tak dapat diubah: tak ada perubahan lewat dia, tetapi perubahan
di belakangnya terlihat.
        List<String> view = Collections.unmodifiableList(services);
        attempt("view.add(\"WDI\")", () -> view.add("WDI"));
        services.add("WDI");                     // ubah list di belakang tampilan
        System.out.println("view now shows: " + view);

        // 3. SALINAN yang tak dapat diubah: potret yang tak dapat diubah oleh apa pun.
        List<String> snapshot = List.copyOf(services);
        services.remove("IR");
        System.out.println("services: " + services + "  snapshot: " + snapshot);
        attempt("snapshot.set(0, \"XX\")", () -> snapshot.set(0, "XX"));

        // 4. Metode pabrik of, beserta aturannya.
        List<Integer> kg = List.of(3, 2, 3);
        Set<String> codes = Set.of("WF", "IR", "DC");
        Map<String, Integer> rate = Map.of("WF", 7000, "IR", 5000);
        System.out.println(kg + " " + codes.size() + " codes, WF costs " + rate.get("WF"));
        attempt("List.of with a null", () -> List.of("WF", null));
        attempt("Set.of with a duplicate", () -> Set.of("WF", "WF"));
        attempt("rate.put(\"DC\", 15000)", () -> rate.put("DC", 15000));
```

Kode 13.17 (Lanjutan) *ImmutableCollections.java: final versus imutabel, tampilan tak dapat diubah yang memperlihatkan perubahan belakangan, salinan tak dapat diubah yang tidak, metode pabrik of, dan ketakterubahan yang dangkal.*

```
        // 5. Ketakterubahan itu dangkal: elemen-elemennya masih bisa mutabel.
        List<StringBuilder> notes = List.of(new StringBuilder("rush"));
        notes.get(0).append(" order");
        System.out.println("inside an unmodifiable list: " + notes);
    }
}
```

```
final list after add: [WF, IR, DC]
view.add("WDI"): UnsupportedOperationException
view now shows: [WF, IR, DC, WDI]
services: [WF, DC, WDI]  snapshot: [WF, IR, DC, WDI]
snapshot.set(0, "XX"): UnsupportedOperationException
[3, 2, 3] 3 codes, WF costs 7000
List.of with a null: NullPointerException
Set.of with a duplicate: IllegalArgumentException
rate.put("DC", 15000): UnsupportedOperationException
inside an unmodifiable list: [rush order]
```

- **Blok 1** menambahkan `"DC"` ke list `final` tanpa keberatan. Hanya baris yang dijadikan komentar, yang akan membuat `services` merujuk ke list lain, yang terlarang.
- **Blok 2** membungkus list dengan `Collections.unmodifiableList(services)`. Hasilnya adalah sebuah **tampilan** (*view*): objek tipis yang meneruskan metode pembaca ke list sebenarnya dan melemparkan `UnsupportedOperationException` dari setiap metode pengubah (`add`, `remove`, `set`, `clear`, `sort`, ...). Tampilan bersifat *hanya-baca*, bukan *imutabel*: ketika list sebenarnya di belakangnya berubah, tampilan itu memperlihatkan perubahannya, seperti ditunjukkan `[WF, IR, DC, WDI]`. Tampilan adalah alat yang tepat ketika sebuah kelas ingin membiarkan pihak lain **melihat** data hidupnya tanpa membiarkan mereka **menyentuhnya**.
- **Blok 3** memakai `List.copyOf(services)`, yang membuat **salinan tak dapat diubah**: potret yang tidak dapat diubah oleh pemanggil maupun oleh siapa pun. Menghapus `IR` dari `services` sesudahnya tidak memengaruhi `snapshot`. Salinan adalah alat yang tepat ketika sebuah kelas harus yakin bahwa data yang diterimanya tidak akan pernah berubah di belakang punggungnya.
- **Blok 4** memperlihatkan metode pabrik `List.of`, `Set.of`, dan `Map.of`, yang ditambahkan pada Java 9 dan langsung menciptakan koleksi kecil yang tak dapat diubah [13]. Metode-metode itu ketat: mereka menolak elemen `null` (`NullPointerException`), `Set.of` menolak duplikat (`IllegalArgumentException`), dan setiap metode pengubah melemparkan `UnsupportedOperationException`.
- **Blok 5** mengingatkan bahwa semua ini bersifat **dangkal**: list tak dapat diubah yang berisi `StringBuilder` tidak dapat bertambah atau berkurang elemennya, tetapi setiap `StringBuilder` di dalamnya masih dapat diubah. Sebuah koleksi hanya seimutabel elemen-elemennya.

Kode 13.18 memperlihatkan gagasan yang sama dalam Python.

Kode 13.18 `immutable_collections.py`: `Final` sebagai petunjuk, `MappingProxyType` sebagai tampilan hanya-baca, `tuple` dan `frozenset` sebagai salinan imutabel, kemampuan di-*hash*, dan imutabilitas yang dangkal.

```
"""immutable_collections.py: tuple, frozenset, dan tampilan hanya-baca."""
from types import MappingProxyType
from typing import Final


def attempt(what, action):
    try:
        action()
        print(f"{what}: allowed")
    except (TypeError, AttributeError) as e:
        print(f"{what}: {type(e).__name__}: {e}")


def main():
    # 1. Final hanyalah janji kepada pemeriksa tipe; Python sendiri tidak menegakkannya.
    SERVICES: Final = ["WF", "IR"]
    SERVICES.append("DC")                  # list-nya tetap saja mutabel
    print("Final list after append:", SERVICES)

    # 2. TAMPILAN hanya-baca atas dict: tak ada perubahan lewat dia, tetapi perubahan di
belakangnya terlihat.
    rates = {"WF": 7000, "IR": 5000}
    view = MappingProxyType(rates)
    def change_view():
        view["DC"] = 15000
    attempt('view["DC"] = 15000', change_view)
    rates["DC"] = 15000
    print("view now shows:", dict(view))

    # 3. SALINAN imutabel: tuple dan frozenset.
    snapshot = tuple(SERVICES)
    SERVICES.remove("IR")
    print("services:", SERVICES, " snapshot:", snapshot)
    def change_snapshot():
        snapshot[0] = "XX"
    attempt("snapshot[0] = 'XX'", change_snapshot)
    codes = frozenset({"WF", "IR", "DC"})
    attempt("codes.add('WDI')", lambda: codes.add("WDI"))
    print("frozenset keeps one of each:", sorted(frozenset(["WF", "WF", "IR"])))

    # 4. Benda imutabel dapat menjadi kunci dictionary; yang mutabel tidak.
    price_by_pair = {("WF", "express"): 10500, frozenset({"wash", "iron"}): 10000}
    print("tuple key:", price_by_pair[("WF", "express")])
    attempt("list as a key", lambda: {["WF", "express"]: 10500})
```

Kode 13.18 (Lanjutan) *immutable_collections.py: Final sebagai petunjuk, MappingProxyType sebagai tampilan hanya-baca, tuple dan frozenset sebagai salinan imutabel, kemampuan di-hash, dan imutabilitas yang dangkal.*

```
    # 5. Tuple itu dangkal: elemen-elemennya masih bisa mutabel.
    notes = (["rush"],)
    notes[0].append("order")
    print("inside a tuple:", notes)
    attempt("hash(notes)", lambda: hash(notes))


if __name__ == "__main__":
    main()
```

```
Final list after append: ['WF', 'IR', 'DC']
view["DC"] = 15000: TypeError: 'mappingproxy' object does not support item assignment
view now shows: {'WF': 7000, 'IR': 5000, 'DC': 15000}
services: ['WF', 'DC']  snapshot: ('WF', 'IR', 'DC')
snapshot[0] = 'XX': TypeError: 'tuple' object does not support item assignment
codes.add('WDI'): AttributeError: 'frozenset' object has no attribute 'add'
frozenset keeps one of each: ['IR', 'WF']
tuple key: 10500
list as a key: TypeError: unhashable type: 'list'
inside a tuple: (['rush', 'order'],)
hash(notes): TypeError: unhashable type: 'list'
```

Python tidak memiliki kata kunci `final`; anotasi `typing.Final` hanya diperiksa oleh alat seperti mypy, tidak pernah saat program berjalan, dan bagaimanapun hanya akan mengunci namanya. `types.MappingProxyType(rates)` adalah **tampilan** hanya-baca Python atas sebuah dictionary: penugasan melaluinya membangkitkan `TypeError`, tetapi ia memperlihatkan perubahan belakangan pada `rates`. **Salinan** imutabelnya adalah tipe bawaan `tuple` (barisan imutabel) dan `frozenset` (himpunan imutabel), sedangkan `frozendict` tidak ada di pustaka standar. Karena tuple dan frozenset tidak dapat berubah, keduanya dapat menjadi kunci dictionary, seperti diperlihatkan `price_by_pair`; `list` tidak dapat, dan percobaannya membangkitkan `TypeError: unhashable type: 'list'`. Blok 5 memperlihatkan versi Python dari kedangkalan: tuple yang memuat list tidak dapat mengubah list *mana* yang dipegangnya, tetapi list itu sendiri dapat berubah, dan karena itu tuple semacam ini pun tidak dapat di-*hash*. **Tabel 13.3** merangkum alat-alatnya.

Tabel 13.3 Alat-alat imutabilitas di Java dan Python.

Kebutuhan	Java	Python
Nama yang ditugaskan sekali	variabel atau field `final`	konvensi (`UPPER_CASE`), `typing.Final` untuk pemeriksa tipe
Tampilan hidup hanya-baca atas koleksi	`Collections.unmodifiableList/Set/Map(c)`	`types.MappingProxyType(d)` untuk dictionary
Potret imutabel sebuah koleksi	`List.copyOf`, `Set.copyOf`, `Map.copyOf`	`tuple(x)`, `frozenset(x)`
Literal koleksi imutabel yang kecil	`List.of`, `Set.of`, `Map.of`	`(a, b, c)`, `frozenset({a, b})`
Kelas imutabel dengan sedikit kode	`record` (Java 16+)	`@dataclass(frozen=True)`, `typing.NamedTuple`
Kelas imutabel yang ditulis dengan tangan	kelas `final`, field `private final`, tanpa setter	`@property` hanya-baca atau `__slots__` tanpa setter

13.8.3 Menulis kelas imutabel

Untuk membuat sebuah kelas imutabel, ikutilah lima aturan [9]:

1. **Jangan sediakan metode yang mengubah keadaan objek** — tanpa setter, tanpa `add`, tanpa `increment`.
2. **Pastikan kelas itu tidak dapat diperluas**, biasanya dengan mendeklarasikannya `final`; jika tidak, sebuah subkelas dapat menambahkan keadaan mutabel dan meng-*override* metode sehingga objek imutabel "milik Anda" ternyata berubah juga.

3. **Jadikan semua field `private` dan `final`**, agar kompilator sendiri menjamin bahwa tidak ada kode, bahkan di dalam kelas itu, yang menugaskan ulang field tersebut setelah konstruksi.
4. **Jaga setiap komponen yang mutabel.** Jika sebuah field merujuk ke objek mutabel seperti list, buatlah **salinan defensif** ketika Anda menerimanya di konstruktor dan jangan pernah menyerahkan objek aslinya; kembalikan tampilan tak dapat diubah atau salinan sebagai gantinya.
5. **Bangun objek sepenuhnya di dalam konstruktor**, dan jangan biarkan `this` lolos sebelum konstruktor selesai.

Agar sebuah objek sepenuhnya imutabel, field-fieldnya harus berupa tipe primitif atau referensi ke objek imutabel (aturan 4 mengurus sisanya). **Kode 13.19** menerapkan aturan-aturan itu pada kelas yang ditulis dengan tangan, lalu memperlihatkan bagaimana *record* Java mengerjakan sebagian besar pekerjaannya untuk kita.

Kode 13.19 `ImmutableClasses.java`: `Point` imutabel yang ditulis dengan tangan, `Student` dengan id seumur hidup, kelas yang membocorkan list-nya, dan record dengan salinan defensif di konstruktor ringkasnya.

```
import java.util.ArrayList;
import java.util.List;

public class ImmutableClasses {
    // Kelas imutabel yang ditulis dengan tangan: kelima aturan beraksi.
    static final class Point {                  // aturan 2: kelas final, tanpa subkelas
        private final int x;                    // aturan 3: field private final
        private final int y;

        Point(int x, int y) {
            this.x = x;
            this.y = y;
        }

        int getX() {                            // aturan 1: getter, tanpa setter
            return x;
        }

        int getY() {
            return y;
        }

        Point withX(int newX) {                 // "mengubah" berarti membuat objek baru
            return new Point(newX, y);
        }

        @Override
        public String toString() {
            return "(" + x + ", " + y + ")";
        }
    }

    // Setiap mahasiswa mendapat id berikutnya sekali, saat lahir, dan memegangnya seumur
hidup.
    static final class Student {
        private static int ID_COUNTER = 1;
        private final int id = ID_COUNTER++;
        private final String name;

        Student(String name) {
            this.name = name;
        }

        int getId() {
            return id;
        }
```

Kode 13.19 (Lanjutan) *ImmutableClasses.java: Point imutabel yang ditulis dengan tangan, Student dengan id seumur hidup, kelas yang membocorkan list-nya, dan record dengan salinan defensif di konstruktor ringkasnya.*

```
        String getName() {
            return name;
        }
    }

    // Tampak imutabel, tetapi membocorkan list-nya ke dua arah.
    static final class LeakyCustomer {
        private final String name;
        private final List<String> phones;

        LeakyCustomer(String name, List<String> phones) {
            this.name = name;
            this.phones = phones;               // menyimpan list milik pemanggil
        }

        List<String> getPhones() {
            return phones;                      // menyerahkan list miliknya sendiri
        }
    }

    // Record dengan konstruktor ringkas yang membuat salinan defensif.
    record Customer(String name, List<String> phones) {
        Customer {                              // aturan 4: salin masukan yang mutabel
            if (name == null || name.isBlank()) {
                throw new IllegalArgumentException("name required");
            }
            phones = List.copyOf(phones);       // potret yang tidak dapat diubah
        }
    }

    public static void main(String[] args) {
        Point p = new Point(1, 2);
        Point q = p.withX(5);
        System.out.println("p = " + p + ", q = " + q);

        Student a = new Student("Robin");
        Student b = new Student("Franky");
        System.out.println(a.getName() + " has id " + a.getId()
                + ", " + b.getName() + " has id " + b.getId());

        List<String> phones = new ArrayList<>(List.of("0812-1111"));
        LeakyCustomer leaky = new LeakyCustomer("Siti", phones);
        phones.add("0813-2222");                // pemanggil mengubah list miliknya
sendiri...
        leaky.getPhones().add("0000-HACK");     // ...dan siapa pun dapat mengubah list
dari getter
        System.out.println("leaky phones: " + leaky.getPhones());
```

Kode 13.19 (Lanjutan) *ImmutableClasses.java: Point imutabel yang ditulis dengan tangan, Student dengan id seumur hidup, kelas yang membocorkan list-nya, dan record dengan salinan defensif di konstruktor ringkasnya.*

```
        phones = new ArrayList<>(List.of("0812-1111"));
        Customer safe = new Customer("Siti", phones);
        phones.add("0813-2222");
        System.out.println("safe phones:  " + safe.phones());
        try {
            safe.phones().add("0000-HACK");
        } catch (UnsupportedOperationException e) {
            System.out.println("safe.phones().add: " + e.getClass().getSimpleName());
        }

        // Record memberi equals, hashCode, dan toString secara cuma-cuma, semuanya dari
field.
        Customer twin = new Customer("Siti", List.of("0812-1111"));
        System.out.println(safe + " equals twin? " + safe.equals(twin));
        try {
            new Customer(" ", List.of());
        } catch (IllegalArgumentException e) {
            System.out.println("blank name: " + e.getMessage());
        }
    }
}
```

```
p = (1, 2), q = (5, 2)
Robin has id 1, Franky has id 2
leaky phones: [0812-1111, 0813-2222, 0000-HACK]
safe phones:  [0812-1111]
safe.phones().add: UnsupportedOperationException
Customer[name=Siti, phones=[0812-1111]] equals twin? true
blank name: name required
```

`Point` mengikuti aturan 1–3 dan 5 secara kasatmata, dan tidak memerlukan apa pun untuk aturan 4, karena `int` adalah nilai. Untuk "mengubah" sebuah titik kita memanggil `withX(5)`, yang mengembalikan `Point` *baru*; `p` sendiri tetap `(1, 2)`. Gaya `withX` — metode yang mengembalikan salinan yang telah dimodifikasi — adalah cara setiap tipe imutabel menawarkan perubahan: `String.toUpperCase`, `LocalDate.plusDays`, `BigDecimal.add` semuanya bekerja dengan cara ini.

`Student` memperlihatkan pola untuk pengenal. `ID_COUNTER` adalah field `static` yang dimiliki bersama oleh kelas dan dinaikkan sekali untuk setiap mahasiswa; field instans `private final int id = ID_COUNTER++;` mengambil nomor berikutnya ketika mahasiswa diciptakan dan memegangnya seumur hidup. `name` bertipe `String`, yang imutabel, sehingga seluruh `Student` imutabel.

`LeakyCustomer` adalah jebakan klasik. Field-fieldnya `private final`, ia tidak memiliki setter, dan ia *tampak* imutabel. Namun konstruktornya menyimpan list milik pemanggil, dan getter-nya menyerahkan list yang sama itu. Keluaran memperlihatkan akibatnya: pemanggil menambahkan `0813-2222` ke list miliknya *setelah* membangun pelanggan itu, dan seorang asing menambahkan `0000-HACK` lewat getter, dan kedua perubahan itu mendarat di dalam objek yang "imutabel". `final` menjaga *referensinya*, bukan list-nya.

`Customer` adalah sebuah **record**, yang diperkenalkan pada Java 16 [14]. Baris `record Customer(String name, List<String> phones) { ... }` mendeklarasikan kelas yang otomatis `final`, dengan field `private final` bernama `name` dan `phones`, konstruktor yang mengisinya, metode pengakses `name()` dan `phones()`, serta `equals`, `hashCode`, dan `toString` yang dihitung dari field-fieldnya — segala hal yang ditulis dengan tangan di Bab 7 untuk kelas nilai. Blok `Customer { ... }` adalah **konstruktor ringkas** (*compact constructor*): ia berjalan sebelum field ditugaskan,

dan ia boleh memvalidasi serta mengganti parameternya. Konstruktor kita menolak nama kosong dan mengganti `phones` dengan `List.copyOf(phones)`, yakni aturan 4 dalam satu baris. Kini perubahan belakangan oleh pemanggil tidak menjangkau record itu, dan `safe.phones().add(...)` melemparkan eksepsi. Record adalah cara yang diutamakan Java untuk menulis kelas data imutabel yang kecil; satu-satunya hal yang tidak dapat dikerjakannya untuk Anda adalah salinan defensif, karena kompilator tidak dapat mengetahui field mana yang mutabel.

Kode 13.20 menulis kelas-kelas yang sama dalam Python.

Kode 13.20 `immutable_classes.py`: dataclass beku dengan `replace`, id yang ditetapkan saat lahir, kelas yang bocor, salinan defensif di `__post_init__`, dan `NamedTuple`.

```
"""immutable_classes.py: dataclass beku, salinan defensif, dan NamedTuple."""
from dataclasses import FrozenInstanceError, dataclass, field, replace
from itertools import count
from typing import NamedTuple


@dataclass(frozen=True)
class Point:
    x: int
    y: int

    def with_x(self, new_x):
        return replace(self, x=new_x)    # Point baru; self tidak tersentuh


_ids = count(1)


@dataclass(frozen=True)
class Student:
    name: str
    id: int = field(default_factory=lambda: next(_ids))   # ditetapkan saat lahir


class LeakyCustomer:
    """Tampak imutabel, tetapi membocorkan list-nya ke dua arah."""

    def __init__(self, name, phones):
        self._name = name
        self._phones = phones            # menyimpan list milik pemanggil

    @property
    def phones(self):
        return self._phones              # menyerahkan list miliknya sendiri


@dataclass(frozen=True)
class Customer:
    name: str
    phones: tuple
```

Kode 13.20 (Lanjutan) *immutable_classes.py: dataclass beku dengan replace, id yang ditetapkan saat lahir, kelas yang bocor, salinan defensif di __post_init__, dan NamedTuple.*

```
    def __post_init__(self):
        if not self.name or not self.name.strip():
            raise ValueError("name required")
        # frozen memblokir penugasan biasa, jadi pakai object.__setattr__ sekali, di sini.
        object.__setattr__(self, "phones", tuple(self.phones))


class Money(NamedTuple):                # tuple dengan field bernama: imutabel juga
    rupiah: int


def main():
    p = Point(1, 2)
    q = p.with_x(5)
    print(f"p = {p}, q = {q}")
    try:
        p.x = 9
    except FrozenInstanceError as e:
        print("p.x = 9:", type(e).__name__, "-", e)

    a, b = Student("Robin"), Student("Franky")
    print(f"{a.name} has id {a.id}, {b.name} has id {b.id}")

    phones = ["0812-1111"]
    leaky = LeakyCustomer("Siti", phones)
    phones.append("0813-2222")
    leaky.phones.append("0000-HACK")
    print("leaky phones:", leaky.phones)

    phones = ["0812-1111"]
    safe = Customer("Siti", phones)
    phones.append("0813-2222")
    print("safe phones: ", safe.phones)
    try:
        safe.phones.append("0000-HACK")
    except AttributeError as e:
        print("safe.phones.append:", type(e).__name__)

    twin = Customer("Siti", ["0812-1111"])
    print(safe, "equals twin?", safe == twin, " same hash?", hash(safe) == hash(twin))
    try:
        Customer(" ", [])
    except ValueError as e:
        print("blank name:", e)
```

Kode 13.20 (Lanjutan) *immutable_classes.py: dataclass beku dengan replace, id yang ditetapkan saat lahir, kelas yang bocor, salinan defensif di __post_init__, dan NamedTuple.*

```
    m = Money(7000)
    print(m, m.rupiah, m._replace(rupiah=5000), m)


if __name__ == "__main__":
    main()
```

```
p = Point(x=1, y=2), q = Point(x=5, y=2)
p.x = 9: FrozenInstanceError - cannot assign to field 'x'
Robin has id 1, Franky has id 2
leaky phones: ['0812-1111', '0813-2222', '0000-HACK']
safe phones:  ('0812-1111',)
safe.phones.append: AttributeError
Customer(name='Siti', phones=('0812-1111',)) equals twin? True  same hash? True
blank name: name required
Money(rupiah=7000) 7000 Money(rupiah=5000) Money(rupiah=7000)
```

`@dataclass(frozen=True)` adalah padanan Python untuk record [15]: ia membangkitkan `__init__`, `__repr__`, `__eq__`, dan `__hash__` dari field-field beranotasi, dan `frozen=True` membuat setiap penugasan pada sebuah field membangkitkan `FrozenInstanceError`, seperti diperlihatkan `p.x = 9`. `dataclasses.replace(self, x=new_x)` membangun salinan yang telah dimodifikasi, yakni pola `withX`. `Student` memperoleh id-nya dari `field(default_factory=lambda: next(_ids))`, dengan `_ids = count(1)` sebagai iterator tak hingga dari `itertools` yang menghasilkan 1, 2, 3, … — pencacah versi Java, ditulis sebagai iterator.

Salinan defensif ditempatkan di `__post_init__`, metode yang dipanggil dataclass tepat setelah `__init__`. Karena objeknya sudah beku pada saat itu, `self.phones = ...` yang biasa akan gagal; jalan keluar yang terdokumentasi adalah `object.__setattr__(self, "phones", tuple(self.phones))`, dipakai sekali, di dalam kelas, selama konstruksi. Mengubah list menjadi tuple sekaligus menyalinnya dan membuatnya imutabel, sehingga `safe.phones.append` gagal dengan `AttributeError`. Akhirnya, `typing.NamedTuple` menciptakan subkelas tuple dengan field bernama; ia imutabel karena tuple imutabel, dan `_replace` membuat salinan yang telah dimodifikasi.

> **PERINGATAN:** Imutabilitas di Python adalah konvensi yang kuat, bukan tembok besi: `object.__setattr__` bekerja pada dataclass beku dari mana saja, tidak hanya dari `__post_init__`. Dataclass beku melindungi dari *kekeliruan*, dan itulah yang penting dalam praktik; ia tidak melindungi dari perusakan yang disengaja, dan field `final` di Java pun tidak sepenuhnya melindungi dari hal itu (refleksi dapat menjangkaunya). Rancanglah untuk pemrogram jujur yang bisa berbuat keliru.

13.9 Objek Nilai, Kesamaan, dan Hashing

Sebagian objek ditentukan oleh **siapa mereka**: dua mesin cuci bermodel sama dengan setelan sama tetaplah dua mesin, dan penting untuk tahu mesin mana yang rusak. Sebagian lain ditentukan sepenuhnya oleh **apa mereka**: dua jumlah Rp7,000 adalah jumlah yang sama; dua tanggal 21 September 2026 adalah tanggal yang sama; tidak masuk akal bertanya Rp7,000 *yang mana* yang Anda pegang.

> **DEFINISI. Objek nilai** (*value object*) adalah objek yang identitasnya tidak penting: ia sepenuhnya dijelaskan oleh nilai field-fieldnya, dua objek nilai dengan field yang sama dapat saling dipertukarkan, dan ia (hampir selalu) imutabel. Sebaliknya, sebuah **entitas** memiliki identitas yang bertahan selagi field-fieldnya berubah [16].

Uang, harga, tanggal, berat, koordinat, dan warna adalah objek nilai; pelanggan, mesin, dan rekening bank adalah entitas. Objek nilai semestinya imutabel: seandainya Anda dapat mengubah "sang" Rp7,000, setiap tempat yang memakainya bersama akan ikut berubah.

Karena objek nilai dijelaskan oleh field-fieldnya, `equals`-nya (di Python `__eq__`) harus membandingkan field-field itu, dan `hashCode`-nya (`__hash__`) harus dihitung dari field yang sama.

Bab 7 sudah menyatakan aturannya; kini kita dapat melihat mengapa aturan itu ada dan apa yang rusak bila kuncinya mutabel. Aturannya sendiri adalah satu implikasi:

$$a.\text{equals}(b) \;\Rightarrow\; h(a) = h(b) \tag{13.6}$$

dengan h sebagai fungsi hash. Persamaan (13.6) menyatakan bahwa objek-objek yang sama harus memiliki kode hash yang sama. (Kebalikannya tidak harus berlaku: objek yang berbeda boleh berbagi kode hash.) Alasannya adalah cara kerja tabel hash [17]. Sebuah `HashMap` atau `dict` Python menyimpan larik berisi m **bucket** (keranjang). Untuk menyimpan atau menemukan kunci k, ia menghitung kode hash kunci itu dan memakainya untuk memilih satu bucket:

$$i = h(k) \bmod m \tag{13.7}$$

lalu mencari *hanya* di bucket i, membandingkan kunci dengan `equals`. (`HashMap` di Java lebih dulu mencampurkan bit-bit tinggi $h(k)$ ke bit-bit rendahnya dan memakai perpangkatan dua untuk m, sehingga sisa bagi dihitung dengan *bit mask*; untuk kode hash kecil di bawah ini, hasilnya tepat Persamaan (13.7).) Inilah sebabnya tabel hash cepat — satu bucket alih-alih seluruh tabel — dan mengapa Persamaan (13.6) mutlak perlu: jika dua kunci yang sama dapat memiliki kode hash berbeda, Persamaan (13.7) akan mengirim keduanya ke bucket yang berbeda, dan tabel itu tidak akan pernah dapat menemukan yang satu bila diberi yang lain.

Sekarang misalkan sebuah kunci **berubah setelah disimpan**. Kode hash-nya dihitung dari field-fieldnya, sehingga kode hash-nya pun berubah. Entrinya masih berada di bucket yang dipilih oleh kode hash *lama*, tetapi setiap pencarian pergi ke bucket yang dipilih oleh kode hash *baru*. **Kode 13.21** melakukan persis hal itu.

Kode 13.21 `MutableKey.java`: `Spot` yang mutabel dipakai sebagai kunci di `HashMap` dan `HashSet`, diubah selagi tersimpan; lalu record imutabel yang dipakai dengan benar.

```java
import java.util.HashMap;
import java.util.HashSet;
import java.util.Map;
import java.util.Objects;
import java.util.Set;

public class MutableKey {
    // Titik mutabel dengan equals dan hashCode yang dihitung dari field-fieldnya.
    static final class Spot {
        int row, col;

        Spot(int row, int col) {
            this.row = row;
            this.col = col;
        }

        @Override
        public boolean equals(Object o) {
            return o instanceof Spot s && s.row == row && s.col == col;
        }

        @Override
        public int hashCode() {
            return Objects.hash(row, col);
        }

        @Override
        public String toString() {
            return "Spot(" + row + "," + col + ")";
        }
    }

    // Versi imutabel: field sebuah record bersifat final.
    record Shelf(int row, int col) { }

    public static void main(String[] args) {
        Spot spot = new Spot(1, 2);
        Map<Spot, String> rack = new HashMap<>();
        rack.put(spot, "Siti's bag");
        Set<Spot> used = new HashSet<>();
        used.add(spot);
        System.out.println("hashCode before: " + spot.hashCode());
```

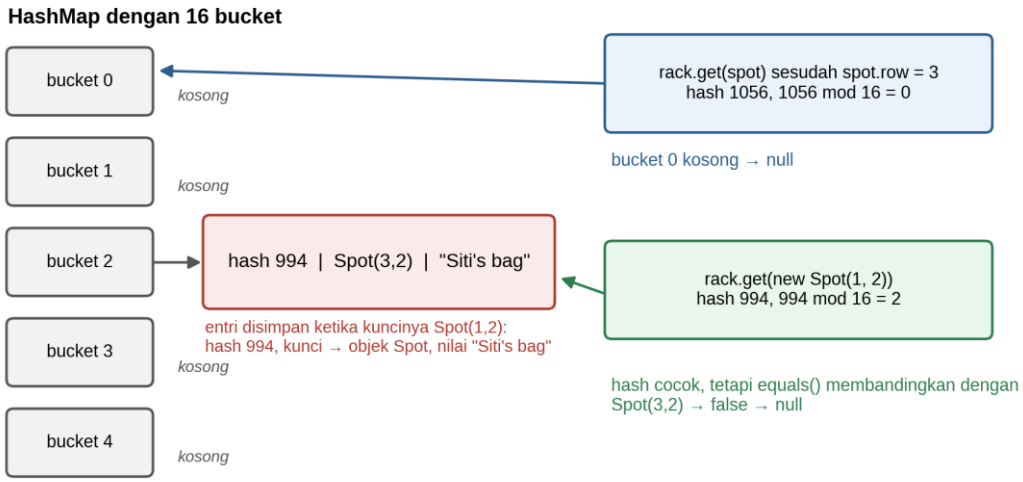
Kode 13.21 (Lanjutan) *MutableKey.java: Spot yang mutabel dipakai sebagai kunci di HashMap dan HashSet, diubah selagi tersimpan; lalu record imutabel yang dipakai dengan benar.*

```
        spot.row = 3;                          // ubah kunci selagi ia berada di dalam map
        System.out.println("hashCode after:  " + spot.hashCode());
        System.out.println("rack.get(spot):              " + rack.get(spot));
        System.out.println("rack.get(new Spot(1, 2)):    " + rack.get(new Spot(1, 2)));
        System.out.println("rack.containsKey(spot):      " + rack.containsKey(spot));
        System.out.println("but the entry is still there: " + rack);
        used.add(spot);                        // kunci yang "sama" masuk dua kali
        System.out.println("used = " + used + ", size " + used.size());

        Map<Shelf, String> shelves = new HashMap<>();
        Shelf shelf = new Shelf(1, 2);
        shelves.put(shelf, "Siti's bag");
        // shelf.row = 3;                      // tidak akan lolos kompilasi: record
imutabel
        Shelf moved = new Shelf(3, 2);         // "memindahkan" berarti kunci baru
        shelves.put(moved, shelves.remove(shelf));
        System.out.println("shelves = " + shelves);
        System.out.println("equal records, equal hashes: "
                + (new Shelf(3, 2).hashCode() == moved.hashCode()));
    }
}
```

```
hashCode before: 994
hashCode after:  1056
rack.get(spot):              null
rack.get(new Spot(1, 2)):    null
rack.containsKey(spot):      false
but the entry is still there: {Spot(3,2)=Siti's bag}
used = [Spot(3,2), Spot(3,2)], size 2
shelves = {Shelf[row=3, col=2]=Siti's bag}
equal records, equal hashes: true
```

`Spot` adalah posisi di rak — baris dan kolom — dengan `equals` dan `hashCode` yang dihitung dengan benar dari kedua field (`Objects.hash(row, col)` adalah $31 \cdot (31 \cdot 1 + \text{row}) + \text{col}$). Kita menyimpan tas Siti di bawah `Spot(1, 2)`, yang kode hash-nya $31 \cdot 32 + 2 = 994$. Lalu `spot.row = 3` mengubah kunci itu di tempat, dan kode hash-nya menjadi $31 \cdot 34 + 2 = 1056$. **Gambar 13.4** memperlihatkan letak segala sesuatunya dalam map dengan 16 bucket bawaan.



...

Gambar 13.4 Kunci yang diubah menjadi hilang. Entri disimpan di bucket 994 mod 16 = 2; hash kunci yang berubah mengirim pencarian ke bucket 1056 mod 16 = 0, yang kosong, sedangkan pencarian dengan nilai lama menemukan bucket yang benar tetapi gagal dalam uji `equals`.

- `rack.get(spot)` menghitung hash *baru*, 1056, mencari di bucket 1056 mod 16 = 0, tidak menemukan apa-apa, dan mengembalikan `null`.
- `rack.get(new Spot(1, 2))` menghitung hash *lama*, 994, dan mencari di bucket yang benar, 2. `HashMap` menyimpan kode hash setiap entri di sampingnya, sehingga hash-nya cocok; tetapi kemudian `equals` membandingkan kunci yang tersimpan — yang kini `Spot(3,2)` — dengan `Spot(1,2)`, dan menjawab `false`. Lagi-lagi `null`.
- Entri itu tidak pergi ke mana pun: mencetak map, yang sekadar menelusuri semua bucket, memperlihatkan `{Spot(3,2)=Siti's bag}`. Tas itu ada di rak, dan tidak ada pencarian yang pernah dapat menemukannya. Dengan satu penugasan kita menciptakan kebocoran memori sekaligus tas yang hilang.
- Set mengalami nasib yang sama: menambahkan spot yang "sama" sekali lagi berhasil, karena set mencari di bucket 0, tidak menemukan elemen yang sama, lalu menyisipkan elemen kedua. Set itu kini memegang dua elemen yang sama, hal yang tidak pernah boleh terjadi pada sebuah set.

Obatnya bukan fungsi hash yang lebih pintar; obatnya adalah **imutabilitas**. Record `Shelf` memiliki field `final`, sehingga kuncinya sama sekali tidak dapat berubah (`shelf.row = 3` yang dijadikan komentar tidak akan lolos kompilasi). Untuk "memindahkan" sebuah tas, kita menghapus entri di bawah kunci lama dan menaruhnya di bawah kunci baru. **Kode 13.22** menuturkan kisah yang sama dalam Python.

Kode 13.22 `mutable_key.py`: kunci hilang yang sama di `dict` dan `set`, obat berupa dataclass beku, dan penolakan Python untuk meng-*hash* sebuah list.

```
"""mutable_key.py: apa yang terjadi ketika sebuah kunci berubah selagi berada di dict atau
set."""
from dataclasses import dataclass


class Spot:
    """Titik mutabel yang kesamaan dan hash-nya berasal dari field-fieldnya."""

    def __init__(self, row, col):
        self.row = row
        self.col = col

    def __eq__(self, other):
        return isinstance(other, Spot) and (self.row, self.col) == (other.row, other.col)

    def __hash__(self):
        return hash((self.row, self.col))

    def __repr__(self):
        return f"Spot({self.row},{self.col})"


@dataclass(frozen=True)
class Shelf:                                # frozen + eq  =>  kunci yang aman dan dapat di-
hash
    row: int
    col: int


def main():
    spot = Spot(1, 2)
    rack = {spot: "Siti's bag"}
    used = {spot}
    before = hash(spot)
    spot.row = 3                            # ubah kunci selagi ia berada di dalam dict
    print("hash changed:", before != hash(spot))
    print("rack.get(spot):          ", rack.get(spot))
    print("rack.get(Spot(1, 2)):    ", rack.get(Spot(1, 2)))
    print("spot in rack:            ", spot in rack)
    print("but the entry is still there:", rack)
    used.add(spot)
    print("used =", used, "size", len(used))
```

Kode 13.22 (Lanjutan) *mutable_key.py: kunci hilang yang sama di dict dan set, obat berupa dataclass beku, dan penolakan Python untuk meng-hash sebuah list.*

```
    shelf = Shelf(1, 2)
    shelves = {shelf: "Siti's bag"}
    moved = Shelf(3, 2)
    shelves[moved] = shelves.pop(shelf)
    print("shelves =", shelves)
    print("equal frozen objects, equal hashes:", hash(Shelf(3, 2)) == hash(moved))

    # Python langsung menolak kunci mutabel yang klasik.
    try:
        {[1, 2]: "bag"}
    except TypeError as e:
        print("list as key:", e)


if __name__ == "__main__":
    main()
```

```
hash changed: True
rack.get(spot):           None
rack.get(Spot(1, 2)):     None
spot in rack:             False
but the entry is still there: {Spot(3,2): "Siti's bag"}
used = {Spot(3,2), Spot(3,2)} size 2
shelves = {Shelf(row=3, col=2): "Siti's bag"}
equal frozen objects, equal hashes: True
list as key: unhashable type: 'list'
```

Python berperilaku identik, dengan alasan yang identik. Python juga mengambil satu langkah pencegahan yang tidak diambil Java: wadah mutabel bawaan `list`, `dict`, dan `set` sengaja dibuat **tidak dapat di-*hash*** — `__hash__` mereka diatur menjadi `None` — sehingga kunci mutabel yang paling umum langsung ditolak dengan `TypeError: unhashable type: 'list'`. Kelas yang mendefinisikan `__eq__` tanpa `__hash__` menjadi tidak dapat di-*hash* dengan cara yang sama, seperti diperlihatkan Bab 7. Dataclass beku dengan `eq=True` (nilai bawaannya) memperoleh `__hash__` yang dihitung dari field-fieldnya, persis yang dibutuhkan objek nilai. Kelas `Spot` kita, yang mendefinisikan kedua metode itu dengan tangan pada field yang mutabel, adalah perpaduan yang berbahaya itu.

> **WAWASAN KUNCI:** Pakailah sebagai kunci (dan sebagai elemen set) hanya objek yang kesamaannya **tidak dapat berubah**: string, bilangan, tanggal, enum, record, dataclass beku, tuple berisi hal-hal tersebut. Jika sebuah objek harus berubah, ia adalah entitas; jadikan kuncinya sesuatu yang tidak berubah, misalnya sebuah id.

13.10 Stream, Comprehension, dan itertools

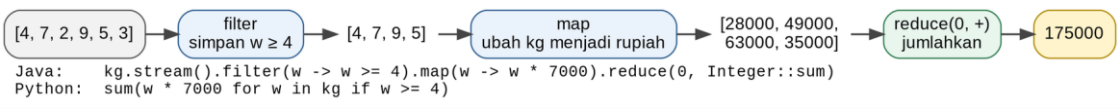
Sejauh ini, setiap kali kita ingin menghitung sesuatu dari sebuah koleksi, kita menulis perulangan: buat variabel hasil, telusuri elemen-elemennya, uji masing-masing dengan `if`, perbarui hasilnya. Gaya itu menyatakan *bagaimana* menghitung, langkah demi langkah. Namun banyak perhitungan atas koleksi dibangun hanya dari tiga gagasan:

- **filter** (saring) — simpan hanya elemen yang memenuhi suatu syarat;
- **map** (petakan) — ubah setiap elemen menjadi sesuatu yang lain;
- **reduce** (ringkas, juga disebut *fold* atau lipat) — gabungkan semua elemen menjadi satu hasil.

"Pendapatan dari muatan seberat sedikitnya 4 kg dengan tarif Rp7,000 per kilogram" berarti: saring muatan dengan $w \geq 4$, petakan setiap w menjadi $7000\,w$, ringkas dengan $+$. Langkah reduce dimulai dari nilai awal e (untuk penjumlahan, 0) lalu menggabungkannya dengan setiap elemen secara bergiliran memakai operasi f:

$$\text{reduce}(f, e, [x_1, x_2, \ldots, x_n]) = f(\cdots f(f(e, x_1), x_2) \cdots, x_n) \tag{13.8}$$

Untuk $f = +$ dan $e = 0$, Persamaan (13.8) tidak lain adalah $x_1 + x_2 + \cdots + x_n$. Untuk $f = \max$ hasilnya elemen terbesar; untuk f = "gabungkan dengan koma" hasilnya sebuah string. **Gambar 13.5** memperlihatkan seluruh perhitungan itu sebagai sebuah *pipeline*.



Gambar 13.5 Pipeline filter–map–reduce. Muatan sumber mengalir melalui sebuah filter dan sebuah map menuju reduksi yang melipatnya menjadi satu bilangan; pipeline yang sama dalam satu baris Java dan satu baris Python.

Java mengungkapkan pipeline semacam itu dengan **stream** (paket `java.util.stream`, sejak Java 8). Sebuah *stream* bukanlah koleksi: ia tidak menyimpan apa pun. Ia adalah deretan elemen yang mengalir dari sebuah *sumber* (koleksi, larik, fungsi pembangkit) melalui sejumlah *operasi antara* (`filter`, `map`, `sorted`, `distinct`, `limit`, ...) menuju satu *operasi terminal* (`reduce`, `sum`, `count`, `collect`, `forEach`, `findFirst`, ...) yang menghasilkan hasilnya [3]. Python mengungkapkannya dengan ***comprehension*** dan **ekspresi generator**, fungsi bawaan `map`, `filter`, `sum`, `min`, `max`, `any`, `all`, `functools.reduce`, serta modul `itertools` [4]. **Kode 13.23** adalah tur di sisi Java.

Kode 13.23 `StreamsDemo.java`: perulangan dan stream yang menghitung total yang sama, kemalasan yang diperlihatkan dengan `peek`, sekali pakai, collector, dan stream tak hingga yang dibatasi dengan `limit`.

```
import java.util.ArrayList;
import java.util.List;
import java.util.Map;
import java.util.TreeMap;
import java.util.stream.Collectors;
import java.util.stream.IntStream;
import java.util.stream.Stream;

public class StreamsDemo {
    record Ticket(String customer, String service, int kg) { }

    public static void main(String[] args) {
        List<Integer> kg = List.of(4, 7, 2, 9, 5, 3);

        // Cara perulangan: katakan BAGAIMANA, langkah demi langkah.
        int loopTotal = 0;
        for (int w : kg) {
            if (w >= 4) {
                loopTotal += w * 7000;
            }
        }
        System.out.println("loop:   " + loopTotal);

        // Cara stream: katakan APA - simpan, ubah, gabungkan.
        int streamTotal = kg.stream()
                .filter(w -> w >= 4)            // simpan muatan yang sedikitnya 4 kg
                .map(w -> w * 7000)             // ubah kilogram menjadi rupiah
                .reduce(0, Integer::sum);       // lipat semuanya menjadi satu bilangan
        System.out.println("stream: " + streamTotal);

        // Stream itu malas: tak ada yang berjalan sampai operasi terminal memintanya.
        Stream<Integer> pipeline = kg.stream()
                .peek(w -> System.out.print(" saw " + w))
                .filter(w -> w > 6);
        System.out.println("pipeline built, nothing printed yet");
        System.out.println(" -> first heavy load: " + pipeline.findFirst().orElse(-1));

        // Sebuah stream hanya dapat dipakai sekali.
        Stream<Integer> once = kg.stream();
        once.count();
        try {
            once.count();
        } catch (IllegalStateException e) {
            System.out.println("second use: " + e.getMessage());
        }
```

Kode 13.23 (Lanjutan) *StreamsDemo.java: perulangan dan stream yang menghitung total yang sama, kemalasan yang diperlihatkan dengan peek, sekali pakai, collector, dan stream tak hingga yang dibatasi dengan limit.*

```
        // Collector mengubah stream kembali menjadi koleksi, string, atau map.
        List<Ticket> tickets = List.of(
                new Ticket("Siti", "WF", 4), new Ticket("Budi", "IR", 2),
                new Ticket("Siti", "IR", 3), new Ticket("Ayu", "WF", 6),
                new Ticket("Budi", "WF", 5));
        List<String> heavy = tickets.stream()
                .filter(t -> t.kg() >= 4)
                .map(Ticket::customer)
                .distinct()
                .sorted()
                .toList();                          // Java 16+; list yang tak dapat diubah
        System.out.println("heavy customers: " + heavy);
        Map<String, Integer> kgPerService = tickets.stream()
                .collect(Collectors.groupingBy(Ticket::service, TreeMap::new,
                        Collectors.summingInt(Ticket::kg)));
        System.out.println("kg per service: " + kgPerService);
        System.out.println("names: " + tickets.stream().map(Ticket::customer)
                .collect(Collectors.joining(", ", "<", ">")));

        // Stream tak hingga tidak masalah, asalkan ada yang membatasinya.
        List<Integer> ticketNumbers = Stream.iterate(101, n -> n + 1)
                .filter(n -> n % 10 != 4)           // lewati nomor yang berakhiran 4
                .limit(5)
                .collect(Collectors.toCollection(ArrayList::new));
        System.out.println("ticket numbers: " + ticketNumbers);
        System.out.println("1 + 2 + ... + 100 = " + IntStream.rangeClosed(1, 100).sum());
    }
}
```

```
loop:   175000
stream: 175000
pipeline built, nothing printed yet
 saw 4 saw 7 -> first heavy load: 7
second use: stream has already been operated upon or closed
heavy customers: [Ayu, Budi, Siti]
kg per service: {IR=5, WF=15}
names: <Siti, Budi, Siti, Ayu, Budi>
ticket numbers: [101, 102, 103, 105, 106]
1 + 2 + ... + 100 = 5050
```

- **Perulangan versus stream.** Keduanya menghitung Rp175,000 = $7000 \times (4 + 7 + 9 + 5)$. Stream itu dibaca dari atas ke bawah seperti kalimat awal kita: `filter(w -> w >= 4)`, `map(w -> w * 7000)`, `reduce(0, Integer::sum)`. Argumennya adalah *ekspresi lambda* — fungsi anonim yang kecil, ditulis `parameter -> hasil` — dan *rujukan metode* seperti `Integer::sum`, yang berarti "metode `sum` milik kelas `Integer`", fungsi yang menjumlahkan dua `int`. Lambda dan stream hadir bersamaan di Java 8, dan keduanya dirancang untuk dipakai bersama [18].
- **Kemalasan.** Stream kedua memuat `peek`, yang memungkinkan kita menyaksikan elemen yang lewat. Membangun pipeline tidak mencetak apa pun: operasi antara hanya *menggambarkan* pekerjaannya. Ketika operasi terminal `findFirst()` berjalan, elemen-elemen mengalir satu per satu — 4 dilihat dan ditolak, 7 dilihat dan diterima — dan stream **berhenti**, karena `findFirst` sudah mendapat jawabannya. Elemen 2, 9, 5, dan 3 tidak pernah diperiksa. (Kata-kata "saw 4 saw 7" muncul sebelum "first heavy load" karena Java mengevaluasi seluruh argumen `println`, termasuk `findFirst()`, sebelum mencetaknya.)
- **Sekali pakai.** Sebuah stream, seperti iterator, hanya dapat ditelusuri sekali. `count()` yang kedua melemparkan `IllegalStateException: stream has already been operated upon or closed`. Untuk menjalankan kueri lain, mintalah stream baru kepada koleksinya.

- **Collector.** `toList()` (Java 16+) mengumpulkan ke dalam list yang tak dapat diubah; `distinct()` dan `sorted()` membuang duplikat dan mengurutkan hasilnya. `Collectors.groupingBy(Ticket::service, TreeMap::new, Collectors.summingInt(Ticket::kg))` membangun map dari setiap kode layanan ke total kilogramnya — "group by" pada basis data, dalam satu ekspresi — dan `Collectors.joining(", ", "<", ">")` meringkas string-string menjadi satu.
- **Stream tak hingga.** `Stream.iterate(101, n -> n + 1)` adalah deretan tanpa akhir 101, 102, 103, ...; `filter` membuang nomor tiket yang berakhiran 4, dan `limit(5)` memotong stream setelah lima elemen. Karena stream itu malas, sumber yang tak hingga tidak menjadi masalah: hanya enam bilangan (101 sampai 106) yang pernah dibangkitkan. `IntStream.rangeClosed(1, 100).sum()` adalah jumlah $1 + 2 + \cdots + 100 = 5050$, stream berisi `int` primitif yang menghindari *boxing*.

Kode 13.24 adalah sisi Python dari tur yang sama.

Kode 13.24 `streams_demo.py`: comprehension dan ekspresi generator, `map`, `filter`, dan `reduce`, kemalasan, sekali pakai, pengelompokan, dan iterator tak hingga dari `itertools`.

```
"""streams_demo.py: comprehension, ekspresi generator, reduce, dan itertools."""
from collections import namedtuple
from functools import reduce
from itertools import accumulate, count, groupby, islice
from operator import add, attrgetter

Ticket = namedtuple("Ticket", "customer service kg")


def main():
    kg = [4, 7, 2, 9, 5, 3]

    # Cara perulangan: katakan BAGAIMANA, langkah demi langkah.
    loop_total = 0
    for w in kg:
        if w >= 4:
            loop_total += w * 7000
    print("loop:  ", loop_total)

    # Cara deklaratif: katakan APA - simpan, ubah, gabungkan.
    print("comp:  ", sum(w * 7000 for w in kg if w >= 4))
    print("reduce:", reduce(add, map(lambda w: w * 7000, filter(lambda w: w >= 4, kg)), 0))

    # Ekspresi generator itu malas; list comprehension tidak.
    def seen(w):
        print(f" saw {w}", end="")
        return w

    pipeline = (w for w in map(seen, kg) if w > 6)
    print("pipeline built, nothing printed yet")
    print(" -> first heavy load:", next(pipeline, -1))

    # Sebuah generator hanya dapat dipakai sekali.
    once = (w for w in kg)
    print("first use:", sum(1 for _ in once), " second use:", sum(1 for _ in once))

    tickets = [Ticket("Siti", "WF", 4), Ticket("Budi", "IR", 2), Ticket("Siti", "IR", 3),
               Ticket("Ayu", "WF", 6), Ticket("Budi", "WF", 5)]
    heavy = sorted({t.customer for t in tickets if t.kg >= 4})    # sebuah set
comprehension
    print("heavy customers:", heavy)
    by_service = sorted(tickets, key=attrgetter("service"))       # groupby memerlukan
masukan yang terurut
    kg_per_service = {s: sum(t.kg for t in group)
                      for s, group in groupby(by_service, key=attrgetter("service"))}
    print("kg per service:", kg_per_service)
    print("names:", "<" + ", ".join(t.customer for t in tickets) + ">")
```

Kode 13.24 (Lanjutan) *streams_demo.py: comprehension dan ekspresi generator, map, filter, dan reduce, kemalasan, sekali pakai, pengelompokan, dan iterator tak hingga dari itertools.*

```
    # Iterator tak hingga tidak masalah, asalkan ada yang membatasinya.
    ticket_numbers = list(islice((n for n in count(101) if n % 10 != 4), 5))
    print("ticket numbers:", ticket_numbers)
    print("1 + 2 + ... + 100 =", sum(range(1, 101)))
    print("running total of kg:", list(accumulate(kg)))


if __name__ == "__main__":
    main()
```

```
loop:   175000
comp:   175000
reduce: 175000
pipeline built, nothing printed yet
 saw 4 saw 7 -> first heavy load: 7
first use: 6  second use: 0
heavy customers: ['Ayu', 'Budi', 'Siti']
kg per service: {'IR': 5, 'WF': 15}
names: <Siti, Budi, Siti, Ayu, Budi>
ticket numbers: [101, 102, 103, 105, 106]
1 + 2 + ... + 100 = 5050
running total of kg: [4, 11, 13, 22, 27, 30]
```

- **Comprehension.** `sum(w * 7000 for w in kg if w >= 4)` terbaca hampir seperti notasi matematika $\sum_{w \in kg,\, w \geq 4} 7000\ w$. Bagian di dalam kurung adalah sebuah **ekspresi generator**: bentuk `ekspresi for variabel in iterable if syarat` adalah sebuah map (ekspresinya), sebuah sumber (`for`-nya), dan sebuah filter (`if`-nya) sekaligus. Taruh di dalam kurung siku, dan ia menjadi ***list comprehension*** yang membangun list; di dalam kurung kurawal, ia menjadi ***set comprehension*** (`{t.customer for t in tickets if t.kg >= 4}`) atau, dengan `key: value`, ***dictionary comprehension***.
- **map, filter, reduce.** Baris berlabel `reduce:` menulis pipeline yang sama dengan fungsi bawaan `map` dan `filter` serta dengan `functools.reduce(add, ..., 0)`, yaitu Persamaan (13.8) dengan f = `operator.add` dan $e = 0$. Cara itu berjalan, tetapi kebanyakan pemrogram Python lebih menyukai comprehension, yang tidak memerlukan lambda.
- **Kemalasan.** Ekspresi generator itu malas, seperti stream Java: membangun `pipeline` tidak mencetak apa pun, dan `next(pipeline, -1)` menarik elemen melalui `seen` hanya sampai elemen pertama di atas 6. Sebaliknya, list comprehension bersifat **rajin** (*eager*): ia berjalan sampai selesai dan membangun seluruh list sekaligus. Pakailah kurung siku bila Anda memerlukan list-nya; pakailah kurung biasa bila Anda hanya perlu menelusuri hasilnya sekali, terutama atas sumber yang besar atau tak hingga.
- **Sekali pakai.** Ekspresi generator adalah iterator, sehingga ia habis setelah satu penelusuran: `sum` yang kedua menghitung 0 elemen. (Berbeda dengan stream Java, ia tidak melemparkan eksepsi; ia sekadar kosong — jebakan Subbab 13.4.3.)
- **Pengelompokan.** `itertools.groupby` mengelompokkan kunci-kunci sama yang *berurutan*, sehingga tiket-tiket lebih dulu diurutkan menurut layanan; dictionary comprehension lalu menjumlahkan kilogram di setiap kelompok. (`collections.Counter` dan `defaultdict` adalah alat pengelompokan lain yang lazim.)
- **Iterator tak hingga.** `itertools.count(101)` menghasilkan 101, 102, … tanpa henti; ekspresi generator menyaringnya, dan `itertools.islice(..., 5)` mengambil lima elemen — ejaan Python untuk `Stream.iterate(...).filter(...).limit(5)`. `itertools.accumulate(kg)` menghasilkan total berjalan, yaitu nilai-nilai antara dari Persamaan (13.8).

Tabel 13.4 adalah kamus percakapan antara kedua bahasa.

Tabel 13.4 Operasi stream di Java dan padanan lazimnya di Python.

Operasi	Stream Java	Python
Simpan elemen yang cocok	`.filter(x -> p(x))`	`(x for x in xs if p(x))`, `filter(p, xs)`
Ubah setiap elemen	`.map(x -> f(x))`	`(f(x) for x in xs)`, `map(f, xs)`
Ratakan koleksi bersarang	`.flatMap(x -> x.items().stream())`	`(y for x in xs for y in x.items)`, `itertools.chain.from_iterable`
Lipat menjadi satu nilai	`.reduce(e, f)`	`functools.reduce(f, xs, e)`
Jumlah, banyak, min, maks	`.mapToInt(...).sum()`, `.count()`, `.min(c)`, `.max(c)`	`sum`, `len(list)`, `min(xs, key=...)`, `max(...)`
Ada / semua yang cocok	`.anyMatch(p)`, `.allMatch(p)`	`any(p(x) for x in xs)`, `all(...)`
Kecocokan pertama	`.filter(p).findFirst()`	`next((x for x in xs if p(x)), default)`
Ambil *k* yang pertama	`.limit(k)`	`itertools.islice(xs, k)`
Buang duplikat, urutkan	`.distinct()`, `.sorted(c)`	`set(...)` / `dict.fromkeys(...)`, `sorted(xs, key=...)`
Kelompokkan	`Collectors.groupingBy(key, collector)`	`itertools.groupby` pada data terurut, `Counter`, `defaultdict`
Kumpulkan ke dalam list	`.toList()` (tak dapat diubah), `Collectors.toList()`	`[... for ...]`, `list(...)`
Sumber tak hingga	`Stream.iterate(seed, f)`, `Stream.generate(s)`	`itertools.count`, `itertools.cycle`, fungsi generator

Sepatah nasihat terakhir. Pipeline paling baik ketika setiap langkahnya sederhana dan **tanpa efek samping**: `filter` hanya menguji, `map` hanya menghitung (`peek` kita hanya untuk keperluan pengajaran). Bila perulangan lebih jelas — banyak cabang, cara keluar lebih awal yang rumit — tulislah perulangan. Kejelasan, bukan mode, yang menentukan.

13.11 Profitina Laundry: Uang, Riwayat Pesanan, dan Pipeline Laporan

Tibalah saatnya merangkai seluruh bab. Binatu memerlukan tiga hal.

1. **Uang dan harga yang tidak dapat berubah karena kekeliruan.** Sejumlah uang adalah objek nilai. Aturan harga — Wash & Fold Rp7,000 per kilogram, minimum 3 kg — juga objek nilai. Begitu sebuah pesanan dicatat, baris-baris dan totalnya harus dibekukan.
2. **Riwayat pesanan** yang dapat ditelusuri mulai dari yang terbaru untuk layar kasir, dan disaring menjadi pesanan satu pelanggan untuk pertanyaan "apa yang dibawa Siti minggu lalu?" — keduanya secara malas, tanpa menyalin riwayat.
3. **Laporan** yang dibangun sebagai pipeline: pendapatan per layanan, tiga pelanggan teratas, kilogram yang dicuci per hari, rata-rata pesanan, dan daftar pesanan besar.

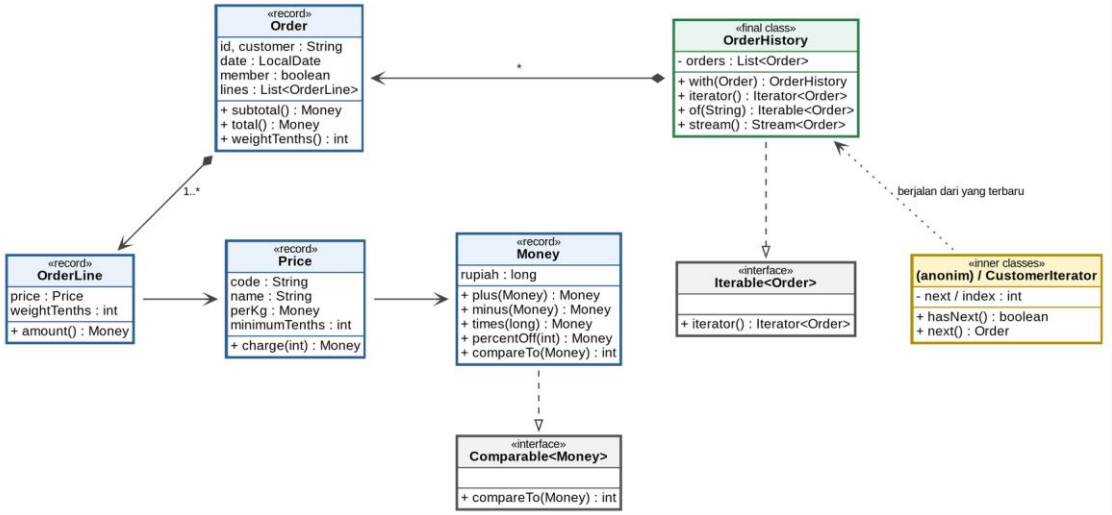
Aturan harganya adalah aturan Bab 7: Wash & Fold (`WF`) Rp7,000 per kilogram dengan minimum 3 kg, Iron Only (`IR`) Rp5,000 dengan minimum 2 kg, Wash, Dry & Iron (`WDI`) Rp10,000 dengan minimum 3 kg, dan anggota membayar 10% lebih murah, dengan potongan dibulatkan ke bawah. Berat yang ditagihkan adalah berat sebenarnya yang dinaikkan ke minimum lalu dibulatkan ke

atas ke setengah kilogram berikutnya. Agar setiap perhitungan eksak, kali ini kita menyimpan berat sebagai bilangan bulat dalam **persepuluhan kilogram** (4.2 kg menjadi 42) dan uang sebagai rupiah bulat dalam `long` — tidak pernah dalam `double`, yang pecahan binernya tidak dapat menyatakan sebagian besar jumlah desimal secara tepat. Dengan w dan $w_{\min}$ dalam persepuluhan, berat yang ditagihkan dalam persepuluhan adalah

$$\hat{w} = 5\left\lceil \frac{\max(w, w_{\min})}{5} \right\rceil \tag{13.9}$$

dan harga untuk tarif r per kilogram adalah $r\hat{w}/10$. Untuk 4.2 kg Wash & Fold, $\max(42,30) = 42$, $\lceil 42/5 \rceil = 9$, sehingga $\hat{w} = 45$ persepuluhan, dan harganya $7000 \cdot 45/10 = 31{,}500$ rupiah. Dalam aritmetika bilangan bulat, $\lceil a/5 \rceil$ dihitung sebagai `(a + 4) / 5`. Persamaan (13.9) adalah aturan yang sama dengan aturan Bab 7, dalam bilangan bulat.

Gambar 13.6 adalah rancangannya. Empat record merupakan objek nilai: `Money`, `Price`, `OrderLine`, dan `Order`. `OrderHistory` adalah koleksi imutabel yang hanya dapat ditambah dan mengimplementasikan `Iterable<Order>`, dan iterator-iteratornya berupa kelas dalam.



Gambar 13.6 Diagram kelas UML buku besar binatu. `Money`, `Price`, `OrderLine`, dan `Order` adalah record (nilai imutabel); `OrderHistory` memegang list pesanan yang tak dapat diubah, mengimplementasikan `Iterable<Order>`, dan memberikan dua jenis iterator.

Kode 13.25 adalah program lengkapnya.

Kode 13.25 `LaundryLedger.java`: Profitina Laundry dengan record `Money` yang imutabel, aturan harga, pesanan beku, `OrderHistory` yang hanya dapat ditambah dengan dua iterator buatan sendiri, dan laporan yang dibangun dari stream.

```
import java.time.LocalDate;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import java.util.Locale;
import java.util.Map;
import java.util.NoSuchElementException;
import java.util.TreeMap;
import java.util.stream.Collectors;
import java.util.stream.Stream;

public class LaundryLedger {
    static final Price WF = new Price("WF", "Wash & Fold", Money.of(7_000), 30);
    static final Price IR = new Price("IR", "Iron Only", Money.of(5_000), 20);
    static final Price WDI = new Price("WDI", "Wash, Dry & Iron", Money.of(10_000), 30);

    static OrderHistory septemberHistory() {
        LocalDate d21 = LocalDate.of(2026, 9, 21);
        LocalDate d22 = d21.plusDays(1);
        LocalDate d23 = d21.plusDays(2);
        return OrderHistory.EMPTY
            .with(new Order("ORD-1301", "Siti Aminah", d21, true,
                    List.of(new OrderLine(WF, 42), new OrderLine(IR, 13))))
            .with(new Order("ORD-1302", "Budi Santoso", d21, false,
                    List.of(new OrderLine(WDI, 50))))
            .with(new Order("ORD-1303", "Ayu Lestari", d21, false,
                    List.of(new OrderLine(WF, 20))))
            .with(new Order("ORD-1304", "Siti Aminah", d22, true,
                    List.of(new OrderLine(WDI, 34))))
            .with(new Order("ORD-1305", "Dewi Anggraini", d22, false,
                    List.of(new OrderLine(WF, 79), new OrderLine(IR, 25))))
            .with(new Order("ORD-1306", "Budi Santoso", d22, false,
                    List.of(new OrderLine(IR, 30))))
            .with(new Order("ORD-1307", "Ayu Lestari", d23, false,
                    List.of(new OrderLine(WF, 51), new OrderLine(WDI, 22))))
            .with(new Order("ORD-1308", "Siti Aminah", d23, true,
                    List.of(new OrderLine(WF, 60))));
    }

    public static void main(String[] args) {
        OrderHistory history = septemberHistory();
```

Kode 13.25 (Lanjutan) *LaundryLedger.java: Profitina Laundry dengan record Money yang imutabel, aturan harga, pesanan beku, OrderHistory yang hanya dapat ditambah dengan dua iterator buatan sendiri, dan laporan yang dibangun dari stream.*

```
        System.out.println("== Order history, newest first ==");
        for (Order o : history) {                  // memakai OrderHistory.iterator()
            System.out.println(o.summary());
        }

        System.out.println("== Siti Aminah's orders ==");
        for (Order o : history.of("Siti Aminah")) {
            System.out.println(o.summary());
        }

        System.out.println("== Revenue per service ==");
        Map<String, Money> perService = history.stream()
                .flatMap(o -> o.lines().stream())
                .collect(Collectors.groupingBy(l -> l.price().code(), TreeMap::new,
                        Collectors.reducing(Money.ZERO, OrderLine::amount, Money::plus)));
        perService.forEach((code, money) ->
                System.out.printf(Locale.US, "%-4s %12s%n", code, money));

        System.out.println("== Top three customers ==");
        Map<String, Money> perCustomer = history.stream()
                .collect(Collectors.groupingBy(Order::customer,
                        Collectors.reducing(Money.ZERO, Order::total, Money::plus)));
        perCustomer.entrySet().stream()
                .sorted(Map.Entry.<String, Money>comparingByValue().reversed()
                        .thenComparing(Map.Entry.comparingByKey()))
                .limit(3)
                .forEach(e -> System.out.printf(Locale.US, "%-15s %10s%n",
                        e.getKey(), e.getValue()));

        System.out.println("== Kilograms per day ==");
        Map<LocalDate, Integer> tenthsPerDay = history.stream()
                .collect(Collectors.groupingBy(Order::date, TreeMap::new,
                        Collectors.summingInt(Order::weightTenths)));
        tenthsPerDay.forEach((day, tenths) ->
                System.out.printf(Locale.US, "%s %5.1f kg%n", day, tenths / 10.0));

        Money revenue = history.stream().map(Order::total).reduce(Money.ZERO, Money::plus);
        long orders = history.stream().count();
        System.out.println("Revenue " + revenue + " from " + orders + " orders, average "
                + Money.of(revenue.rupiah() / orders));
        System.out.println("Orders of Rp50,000 or more: " + history.stream()
                .filter(o -> o.total().compareTo(Money.of(50_000)) >= 0)
                .map(Order::id)
                .toList());
```

Kode 13.25 (Lanjutan) *LaundryLedger.java: Profitina Laundry dengan record Money yang imutabel, aturan harga, pesanan beku, OrderHistory yang hanya dapat ditambah dengan dua iterator buatan sendiri, dan laporan yang dibangun dari stream.*

```
        System.out.println("== Immutability at work ==");
        OrderHistory longer = history.with(new Order("ORD-1309", "Budi Santoso",
                LocalDate.of(2026, 9, 23), false, List.of(new OrderLine(IR, 21))));
        System.out.println("old history: " + history.size() + " orders, new history: "
                + longer.size() + " orders");
        Order first = history.iterator().next();
        try {
            first.lines().add(new OrderLine(WF, 10));
        } catch (UnsupportedOperationException e) {
            System.out.println("adding a line to a stored order: "
                    + e.getClass().getSimpleName());
        }
        try {
            Money.of(5_000).minus(Money.of(7_000));
        } catch (IllegalArgumentException e) {
            System.out.println("Rp5,000 - Rp7,000: " + e.getMessage());
        }
    }
}

// Sejumlah rupiah. Imutabel: setiap operasi mengembalikan Money baru.
record Money(long rupiah) implements Comparable<Money> {
    static final Money ZERO = new Money(0);

    Money {
        if (rupiah < 0) {
            throw new IllegalArgumentException("money cannot be negative");
        }
    }

    static Money of(long rupiah) {
        return new Money(rupiah);
    }

    Money plus(Money other) {
        return new Money(Math.addExact(rupiah, other.rupiah));
    }

    Money minus(Money other) {
        return new Money(rupiah - other.rupiah);   // konstruktor menolak < 0
    }

    Money times(long factor) {
        return new Money(Math.multiplyExact(rupiah, factor));
    }
```

Kode 13.25 (Lanjutan) *LaundryLedger.java: Profitina Laundry dengan record Money yang imutabel, aturan harga, pesanan beku, OrderHistory yang hanya dapat ditambah dengan dua iterator buatan sendiri, dan laporan yang dibangun dari stream.*

```
    Money percentOff(int percent) {
        return new Money(rupiah - rupiah * percent / 100);  // diskon dibulatkan ke bawah
    }

    @Override
    public int compareTo(Money other) {
        return Long.compare(rupiah, other.rupiah);
    }

    @Override
    public String toString() {
        return String.format(Locale.US, "Rp%,d", rupiah);
    }
}

// Aturan harga: tarif per kilogram dan minimum, berat dalam persepuluhan kilogram.
record Price(String code, String name, Money perKg, int minimumTenths) {
    Money charge(int weightTenths) {
        int w = Math.max(weightTenths, minimumTenths);
        int chargeable = (w + 4) / 5 * 5;          // bulatkan ke atas ke 0,5 kg berikutnya
        return Money.of(perKg.rupiah() * chargeable / 10);
    }
}

record OrderLine(Price price, int weightTenths) {
    OrderLine {
        if (weightTenths <= 0) {
            throw new IllegalArgumentException("weight must be positive");
        }
    }

    Money amount() {
        return price.charge(weightTenths);
    }
}

record Order(String id, String customer, LocalDate date, boolean member,
             List<OrderLine> lines) {
    Order {
        lines = List.copyOf(lines);                // salinan defensif yang tak dapat
diubah
    }

    Money subtotal() {
        return lines.stream().map(OrderLine::amount).reduce(Money.ZERO, Money::plus);
    }
```

Kode 13.25 (Lanjutan) *LaundryLedger.java: Profitina Laundry dengan record Money yang imutabel, aturan harga, pesanan beku, OrderHistory yang hanya dapat ditambah dengan dua iterator buatan sendiri, dan laporan yang dibangun dari stream.*

```
    Money total() {
        return member ? subtotal().percentOff(10) : subtotal();
    }

    int weightTenths() {
        return lines.stream().mapToInt(OrderLine::weightTenths).sum();
    }

    String summary() {
        return String.format(Locale.US, "%s %s %-15s %10s", id, date, customer, total());
    }
}

// Riwayat yang hanya bisa ditambah. with() mengembalikan riwayat BARU; yang ini tak pernah
berubah.
final class OrderHistory implements Iterable<Order> {
    static final OrderHistory EMPTY = new OrderHistory(List.of());
    private final List<Order> orders;

    private OrderHistory(List<Order> orders) {
        this.orders = List.copyOf(orders);
    }

    OrderHistory with(Order order) {
        List<Order> longer = new ArrayList<>(orders);
        longer.add(order);
        return new OrderHistory(longer);
    }

    int size() {
        return orders.size();
    }

    Stream<Order> stream() {
        return orders.stream();
    }

    // Terbaru lebih dulu: iterator menelusuri list dari indeks terakhir turun ke 0.
    @Override
    public Iterator<Order> iterator() {
        return new Iterator<>() {
            private int next = orders.size() - 1;
```

Kode 13.25 (Lanjutan) *LaundryLedger.java: Profitina Laundry dengan record Money yang imutabel, aturan harga, pesanan beku, OrderHistory yang hanya dapat ditambah dengan dua iterator buatan sendiri, dan laporan yang dibangun dari stream.*

```
            @Override
            public boolean hasNext() {
                return next >= 0;
            }

            @Override
            public Order next() {
                if (!hasNext()) {
                    throw new NoSuchElementException();
                }
                return orders.get(next--);
            }
        };
    }

    // Hanya pesanan satu pelanggan, terlama lebih dulu, ditemukan secara malas satu per
satu.
    Iterable<Order> of(String customer) {
        return () -> new CustomerIterator(customer);
    }

    private final class CustomerIterator implements Iterator<Order> {
        private final String customer;
        private int index = 0;

        CustomerIterator(String customer) {
            this.customer = customer;
            skipOthers();
        }

        private void skipOthers() {
            while (index < orders.size() && !orders.get(index).customer().equals(customer))
{
                index++;
            }
        }

        @Override
        public boolean hasNext() {
            return index < orders.size();
        }
```

Kode 13.25 (Lanjutan) *LaundryLedger.java: Profitina Laundry dengan record Money yang imutabel, aturan harga, pesanan beku, OrderHistory yang hanya dapat ditambah dengan dua iterator buatan sendiri, dan laporan yang dibangun dari stream.*

```
        @Override
        public Order next() {
            if (!hasNext()) {
                throw new NoSuchElementException();
            }
            Order found = orders.get(index++);
            skipOthers();                          // lihat ke depan, ke kecocokan
berikutnya
            return found;
        }
    }
}
```

```
== Order history, newest first ==
ORD-1308 2026-09-23 Siti Aminah       Rp37,800
ORD-1307 2026-09-23 Ayu Lestari       Rp68,500
ORD-1306 2026-09-22 Budi Santoso      Rp15,000
ORD-1305 2026-09-22 Dewi Anggraini    Rp68,500
ORD-1304 2026-09-22 Siti Aminah       Rp31,500
ORD-1303 2026-09-21 Ayu Lestari       Rp21,000
ORD-1302 2026-09-21 Budi Santoso      Rp50,000
ORD-1301 2026-09-21 Siti Aminah       Rp37,350
== Siti Aminah's orders ==
ORD-1301 2026-09-21 Siti Aminah       Rp37,350
ORD-1304 2026-09-22 Siti Aminah       Rp31,500
ORD-1308 2026-09-23 Siti Aminah       Rp37,800
== Revenue per service ==
IR       Rp37,500
WDI     Rp115,000
WF      Rp189,000
== Top three customers ==
Siti Aminah      Rp106,650
Ayu Lestari       Rp89,500
Dewi Anggraini    Rp68,500
== Kilograms per day ==
2026-09-21  12.5 kg
2026-09-22  16.8 kg
2026-09-23  13.3 kg
Revenue Rp329,650 from 8 orders, average Rp41,206
Orders of Rp50,000 or more: [ORD-1302, ORD-1305, ORD-1307]
== Immutability at work ==
old history: 8 orders, new history: 9 orders
adding a line to a stored order: UnsupportedOperationException
Rp5,000 - Rp7,000: money cannot be negative
```

Mari kita membacanya dari bagian terkecil ke atas.

`Money` adalah record yang memegang sejumlah rupiah bertipe `long`. Konstruktor ringkasnya menegakkan invarian "uang tidak pernah negatif", sehingga tidak ada objek `Money` yang dapat memegang jumlah negatif — tidak saat konstruksi, tidak pula setelah perhitungan apa pun, karena setiap perhitungan membangun `Money` baru melalui konstruktor yang sama. Itulah sebabnya `minus` tidak memerlukan pemeriksaan sendiri: `Money.of(5_000).minus(Money.of(7_000))` mencoba membangun −2000 dan konstruktornya menolak, seperti diperlihatkan baris terakhir keluaran. `plus` dan `times` memakai `Math.addExact` dan `Math.multiplyExact`, yang melemparkan eksepsi bila meluap alih-alih membungkus seperti pada Persamaan (13.5). `percentOff(10)` mengurangkan `rupiah * 10 / 100`, potongan yang dibulatkan ke bawah, aturan Bab 7. `Money` mengimplementasikan `Comparable<Money>`, karena jumlah uang memiliki satu urutan alami yang jelas, dan `toString` mencetak `Rp37,350`. Martin Fowler menggambarkan pola *Money* ini sebagai salah satu objek nilai paling berguna yang dapat dimiliki program bisnis [19].

`Price` adalah record berisi kode, nama tampilan, tarif sebagai `Money`, dan minimum dalam persepuluhan. Metode `charge` miliknya adalah Persamaan (13.9) dalam bentuk kode. `OrderLine` memasangkan sebuah `Price` dengan sebuah berat dan menolak berat yang tidak positif. `Order` memegang id, nama pelanggan, sebuah `LocalDate` (yang juga nilai imutabel), penanda keanggotaan, dan list baris; konstruktor ringkasnya mengganti list itu dengan `List.copyOf(lines)`, sehingga sebuah pesanan, begitu dibangun, tidak pernah dapat bertambah atau berkurang barisnya — keluaran menegaskan bahwa `first.lines().add(...)` melemparkan eksepsi. `subtotal()` adalah stream kecil: petakan setiap baris ke jumlahnya, lalu ringkas dengan `Money::plus` mulai dari `Money.ZERO`.

`OrderHistory` adalah jantung bab ini.

- Ia **imutabel dan hanya dapat ditambah**. Konstruktornya privat dan menyalin list dengan `List.copyOf`; `with(order)` tidak mengubah riwayat, melainkan mengembalikan riwayat *baru* yang lebih panjang satu pesanan. Blok "Immutability at work" memperlihatkan bahwa menambahkan ORD-1309 ke `history` membiarkan `history` tetap berisi 8 pesanan, sementara `longer` berisi 9. Ini pola `withX` yang sama seperti pada `Point`, diterapkan pada sebuah koleksi. Karena setiap riwayat imutabel, kode laporan, layar kasir, dan proses cadangan di latar belakang dapat memegang `OrderHistory` yang sama sekaligus tanpa risiko apa pun. (Menyalin list pada setiap `with` memakan waktu sebanding dengan panjangnya; struktur data *persisten* menghindari biaya itu dengan berbagi struktur.)
- Ia **dapat ditelusuri mulai dari yang terbaru**. `iterator()` mengembalikan sebuah *kelas dalam anonim* — `new Iterator<>() { ... }` menciptakan objek dari kelas yang tidak bernama — yang kursornya, `next`, mulai dari indeks terakhir lalu bergerak turun. Bab 10 memakai kelas anonim untuk *listener*; di sini salah satunya berperan sebagai iterator. Karena `OrderHistory` adalah `Iterable<Order>`, layar kasir cukup berupa satu perulangan for-each: `for (Order o : history)`.
- Ia menawarkan **tampilan tersaring**: `of(customer)` mengembalikan `Iterable<Order>` yang ditulis sebagai lambda, `() -> new CustomerIterator(customer)`. `Iterable` hanya memiliki satu metode abstrak, sehingga sebuah lambda dapat mengimplementasikannya; setiap perulangan for-each atas hasilnya memanggil lambda itu dan memperoleh `CustomerIterator` yang baru. Iterator itu memakai **pandangan ke depan** (*look-ahead*): konstruktornya dan `next()`-nya sama-sama memanggil `skipOthers()`, yang menggeser `index` maju melewati pesanan pelanggan lain, sehingga `hasNext()` dapat menjawab dengan satu perbandingan. Tidak ada yang disalin: ketiga pesanan Siti ditemukan satu per satu, langsung di dalam list riwayat.
- Ia menawarkan `stream()`, agar laporan dapat memakai seluruh kosakata stream.

Laporan di dalam `main` adalah rangkaian pipeline, masing-masing menjawab satu pertanyaan:

- **Pendapatan per layanan** memakai `flatMap(o -> o.lines().stream())` untuk mengubah pesanan-pesanan menjadi seluruh barisnya, lalu mengelompokkan menurut kode harga ke dalam `TreeMap` (kunci terurut), meringkas setiap kelompok dengan `Money::plus`. Angka per layanan, Rp37,500 + Rp115,000 + Rp189,000 = Rp341,500, adalah jumlah baris *sebelum* potongan anggota.
- **Tiga pelanggan teratas** mengelompokkan pesanan menurut pelanggan dan menjumlahkan *total*-nya (sesudah potongan), lalu mengurutkan entri map menurut nilai, menurun, dengan nama sebagai penentu bila seri, dan menyimpan tiga dengan `limit(3)`. Siti berada di urutan pertama dengan Rp106,650.
- **Kilogram per hari** mengelompokkan menurut tanggal dan menjumlahkan berat dengan `summingInt`.
- **Pendapatan** meringkas semua total: Rp329,650, yaitu Rp341,500 dikurangi tiga potongan anggota Siti sebesar Rp4,150, Rp3,500, dan Rp4,200. Rata-rata pesanan adalah Rp329,650 / 8 = Rp41,206 (dibulatkan ke bawah).
- **Pesanan besar** menyaring dengan `compareTo(Money.of(50_000)) >= 0` lalu memetakan ke id. Kita membandingkan objek `Money` dengan `compareTo`, bukan field-fieldnya: objek nilai itulah yang menentukan urutannya sendiri.

Kode 13.26 adalah sistem yang sama dalam Python.

Kode 13.26 `laundry_ledger.py`: buku besar yang sama dengan dataclass beku, iterator terbaru-lebih-dulu berbasis generator, generator per pelanggan yang malas, dan laporan yang dibangun dari comprehension dan `itertools.groupby`.

```
"""laundry_ledger.py: Profitina Laundry dengan nilai-nilai imutabel, iterator riwayat,
dan pipeline laporan."""
from dataclasses import dataclass
from datetime import date, timedelta
from itertools import groupby
from operator import attrgetter


@dataclass(frozen=True, order=True)
class Money:
    """Sejumlah rupiah. Imutabel: setiap operasi mengembalikan Money baru."""
    rupiah: int

    def __post_init__(self):
        if self.rupiah < 0:
            raise ValueError("money cannot be negative")

    def __add__(self, other):
        return Money(self.rupiah + other.rupiah)

    def __sub__(self, other):
        return Money(self.rupiah - other.rupiah)   # __post_init__ menolak < 0

    def __mul__(self, factor):
        return Money(self.rupiah * factor)

    def percent_off(self, percent):
        return Money(self.rupiah - self.rupiah * percent // 100)  # diskon dibulatkan ke
bawah

    def __str__(self):
        return f"Rp{self.rupiah:,}"


ZERO = Money(0)


@dataclass(frozen=True)
class Price:
    """Aturan harga: tarif per kilogram dan minimum, berat dalam persepuluhan kilogram."""
    code: str
    name: str
    per_kg: Money
    minimum_tenths: int
```

Kode 13.26 (Lanjutan) *laundry_ledger.py: buku besar yang sama dengan dataclass beku, iterator terbaru-lebih-dulu berbasis generator, generator per pelanggan yang malas, dan laporan yang dibangun dari comprehension dan itertools.groupby.*

```
    def charge(self, weight_tenths):
        w = max(weight_tenths, self.minimum_tenths)
        chargeable = (w + 4) // 5 * 5            # bulatkan ke atas ke 0,5 kg berikutnya
        return Money(self.per_kg.rupiah * chargeable // 10)


@dataclass(frozen=True)
class OrderLine:
    price: Price
    weight_tenths: int

    def __post_init__(self):
        if self.weight_tenths <= 0:
            raise ValueError("weight must be positive")

    def amount(self):
        return self.price.charge(self.weight_tenths)


@dataclass(frozen=True)
class Order:
    id: str
    customer: str
    date: date
    member: bool
    lines: tuple

    def __post_init__(self):
        object.__setattr__(self, "lines", tuple(self.lines))   # salinan defensif

    def subtotal(self):
        return sum((line.amount() for line in self.lines), ZERO)

    def total(self):
        return self.subtotal().percent_off(10) if self.member else self.subtotal()

    def weight_tenths(self):
        return sum(line.weight_tenths for line in self.lines)

    def summary(self):
        return f"{self.id} {self.date} {self.customer:<15} {str(self.total()):>10}"


class OrderHistory:
    """Riwayat yang hanya bisa ditambah. with_order() mengembalikan riwayat BARU."""
```

Kode 13.26 (Lanjutan) *laundry_ledger.py: buku besar yang sama dengan dataclass beku, iterator terbaru-lebih-dulu berbasis generator, generator per pelanggan yang malas, dan laporan yang dibangun dari comprehension dan itertools.groupby.*

```
    def __init__(self, orders=()):
        self._orders = tuple(orders)

    def with_order(self, order):
        return OrderHistory(self._orders + (order,))

    def __len__(self):
        return len(self._orders)

    def __iter__(self):
        """Terbaru lebih dulu: generator yang menelusuri tuple secara mundur."""
        for i in range(len(self._orders) - 1, -1, -1):
            yield self._orders[i]

    def of(self, customer):
        """Hanya pesanan satu pelanggan, terlama lebih dulu, ditemukan secara malas satu
per satu."""
        return (o for o in self._orders if o.customer == customer)

    def oldest_first(self):
        return iter(self._orders)


WF = Price("WF", "Wash & Fold", Money(7_000), 30)
IR = Price("IR", "Iron Only", Money(5_000), 20)
WDI = Price("WDI", "Wash, Dry & Iron", Money(10_000), 30)


def september_history():
    d21 = date(2026, 9, 21)
    d22, d23 = d21 + timedelta(days=1), d21 + timedelta(days=2)
    history = OrderHistory()
    for order in [
        Order("ORD-1301", "Siti Aminah", d21, True, [OrderLine(WF, 42), OrderLine(IR,
13)]),
        Order("ORD-1302", "Budi Santoso", d21, False, [OrderLine(WDI, 50)]),
        Order("ORD-1303", "Ayu Lestari", d21, False, [OrderLine(WF, 20)]),
        Order("ORD-1304", "Siti Aminah", d22, True, [OrderLine(WDI, 34)]),
        Order("ORD-1305", "Dewi Anggraini", d22, False, [OrderLine(WF, 79), OrderLine(IR,
25)]),
        Order("ORD-1306", "Budi Santoso", d22, False, [OrderLine(IR, 30)]),
        Order("ORD-1307", "Ayu Lestari", d23, False, [OrderLine(WF, 51), OrderLine(WDI,
22)]),
        Order("ORD-1308", "Siti Aminah", d23, True, [OrderLine(WF, 60)]),
    ]:
        history = history.with_order(order)
    return history
```

Kode 13.26 (Lanjutan) *laundry_ledger.py: buku besar yang sama dengan dataclass beku, iterator terbaru-lebih-dulu berbasis generator, generator per pelanggan yang malas, dan laporan yang dibangun dari comprehension dan itertools.groupby.*

```
def main():
    history = september_history()

    print("== Order history, newest first ==")
    for o in history:
        print(o.summary())

    print("== Siti Aminah's orders ==")
    for o in history.of("Siti Aminah"):
        print(o.summary())

    print("== Revenue per service ==")
    lines = [line for o in history.oldest_first() for line in o.lines]
    codes = sorted({line.price.code for line in lines})
    per_service = {code: sum((ln.amount() for ln in lines if ln.price.code == code), ZERO)
                   for code in codes}
    for code in codes:
        print(f"{code:<4} {str(per_service[code]):>12}")

    print("== Top three customers ==")
    by_customer = sorted(history.oldest_first(), key=attrgetter("customer"))
    per_customer = {name: sum((o.total() for o in group), ZERO)
                    for name, group in groupby(by_customer, key=attrgetter("customer"))}
    ranking = sorted(per_customer.items(), key=lambda kv: (-kv[1].rupiah, kv[0]))
    for name, money in ranking[:3]:
        print(f"{name:<15} {str(money):>10}")

    print("== Kilograms per day ==")
    for day, group in groupby(history.oldest_first(), key=attrgetter("date")):
        tenths = sum(o.weight_tenths() for o in group)
        print(f"{day} {tenths / 10:5.1f} kg")

    revenue = sum((o.total() for o in history), ZERO)
    count = len(history)
    print(f"Revenue {revenue} from {count} orders, average {Money(revenue.rupiah //
count)}")
    print("Orders of Rp50,000 or more:",
          [o.id for o in history.oldest_first() if o.total() >= Money(50_000)])
```

Kode 13.26 (Lanjutan) *laundry_ledger.py: buku besar yang sama dengan dataclass beku, iterator terbaru-lebih-dulu berbasis generator, generator per pelanggan yang malas, dan laporan yang dibangun dari comprehension dan itertools.groupby.*

```
    print("== Immutability at work ==")
    longer = history.with_order(Order("ORD-1309", "Budi Santoso", date(2026, 9, 23), False,
                                      [OrderLine(IR, 21)]))
    print(f"old history: {len(history)} orders, new history: {len(longer)} orders")
    first = next(iter(history))
    try:
        first.lines.append(OrderLine(WF, 10))
    except AttributeError as e:
        print("adding a line to a stored order:", type(e).__name__)
    try:
        Money(5_000) - Money(7_000)
    except ValueError as e:
        print("Rp5,000 - Rp7,000:", e)


if __name__ == "__main__":
    main()
```

```
== Order history, newest first ==
ORD-1308 2026-09-23 Siti Aminah       Rp37,800
ORD-1307 2026-09-23 Ayu Lestari       Rp68,500
ORD-1306 2026-09-22 Budi Santoso      Rp15,000
ORD-1305 2026-09-22 Dewi Anggraini    Rp68,500
ORD-1304 2026-09-22 Siti Aminah       Rp31,500
ORD-1303 2026-09-21 Ayu Lestari       Rp21,000
ORD-1302 2026-09-21 Budi Santoso      Rp50,000
ORD-1301 2026-09-21 Siti Aminah       Rp37,350
== Siti Aminah's orders ==
ORD-1301 2026-09-21 Siti Aminah       Rp37,350
ORD-1304 2026-09-22 Siti Aminah       Rp31,500
ORD-1308 2026-09-23 Siti Aminah       Rp37,800
== Revenue per service ==
IR       Rp37,500
WDI     Rp115,000
WF      Rp189,000
== Top three customers ==
Siti Aminah      Rp106,650
Ayu Lestari       Rp89,500
Dewi Anggraini    Rp68,500
== Kilograms per day ==
2026-09-21  12.5 kg
2026-09-22  16.8 kg
2026-09-23  13.3 kg
Revenue Rp329,650 from 8 orders, average Rp41,206
Orders of Rp50,000 or more: ['ORD-1302', 'ORD-1305', 'ORD-1307']
== Immutability at work ==
old history: 8 orders, new history: 9 orders
adding a line to a stored order: AttributeError
Rp5,000 - Rp7,000: money cannot be negative
```

Keluarannya sama dengan keluaran Java, kecuali cara mencetak list berisi string dan nama eksepsinya. Perbedaan-perbedaan dalam kodenya bersifat mendidik.

- `Money` adalah `@dataclass(frozen=True, order=True)`: `frozen` untuk imutabilitas, `order=True` untuk urutan alami (ia membandingkan satu-satunya field, `rupiah`). Alih-alih metode bernama `plus` dan `minus`, ia mendefinisikan operator `__add__`, `__sub__`, dan `__mul__`, sehingga jumlah uang ditulis `a + b`. Dan karena `Money` mendukung `+`, fungsi bawaan `sum` dapat dipakai padanya bila diberi nilai awal: `sum((line.amount() for line in self.lines), ZERO)` adalah Persamaan (13.8) dengan e = `ZERO`.
- `Order.__post_init__` mengubah baris-baris menjadi tuple, yakni salinan defensif dari Subbab 13.8.3.
- `OrderHistory` menyimpan sebuah tuple, dan `with_order` mengembalikan riwayat baru yang dibangun dari `self._orders + (order,)`. `__iter__` miliknya adalah **generator** yang menelusuri indeks secara mundur — seluruh kelas iterator anonim Java dalam dua baris. `of(customer)` mengembalikan sebuah **ekspresi generator**, yang malas persis seperti `CustomerIterator` di Java. `oldest_first()` mengembalikan iterator biasa atas tuple itu untuk keperluan laporan.
- Laporan memakai set comprehension untuk menemukan kode-kode layanan, dictionary comprehension untuk menjumlahkannya, `sorted` dengan kunci tuple `(-money.rupiah, name)` untuk "terbesar lebih dulu, lalu menurut nama", dan `groupby` untuk total per pelanggan dan per hari. Pesanan-pesanan sudah terurut menurut tanggal, sehingga pengelompokan menurut tanggal tidak memerlukan pengurutan; pengelompokan menurut pelanggan memerlukannya.

13.12 Profitina dalam Praktik

Buku besar binatu itu kecil, tetapi ketiga gagasannya berlaku langsung pada Profitina, platform visibilitas AI (GEO) yang pemindainya mengukur seberapa siap sebuah situs web untuk dikutip oleh mesin penjawab AI.

Hasil pemindaian sebagai nilai. Ketika Profitina memindai sebuah situs, setiap sinyal — kemudahan ekstraksi konten, data terstruktur, akses `robots.txt` bagi mesin penjawab, dan seterusnya — menghasilkan sebuah pengukuran dan skor beserta penjelasannya. Begitu sebuah pemindaian selesai, hasilnya adalah *fakta tentang masa lalu*: situs itu tampak seperti ini pada hari itu. Hasil semacam itu secara alami merupakan objek nilai yang imutabel, seperti `Order` pada **Kode 13.25**. Itulah yang membuat **dasbor analitik historis** dapat dipercaya: tren skor mingguan dihitung dari hasil yang tidak dapat disunting oleh kode mana pun sesudahnya, minggu yang sama selalu memperlihatkan angka yang sama, dan ekspor CSV menuliskan persis apa yang diperlihatkan grafiknya.

Menelusuri riwayat besar secara malas. Tren sebuah merek adalah penelusuran atas riwayat pemindaian yang terus bertambah, mulai dari yang terbaru, disaring untuk satu merek — iterator-iterator `OrderHistory` dalam skala yang lebih besar. Penelusuran yang malas memperlihatkan minggu-minggu terakhir seketika tanpa memuat hasil bertahun-tahun.

Laporan sebagai pipeline. Angka-angka dasbor — skor rata-rata per minggu, sinyal yang paling meningkat — adalah pipeline filter–map–reduce atas hasil pemindaian yang imutabel, ditulis dengan comprehension dalam basis kode Python milik Profitina, seperti laporan pada **Kode 13.26**.

Imutabilitas dan data pribadi. Profitina mematuhi Undang-Undang Pelindungan Data Pribadi (UU PDP No. 27/2022): persetujuan dinyatakan secara eksplisit, dan pengguna dapat mengekspor atau menghapus seluruh datanya. Kode yang menangani data pribadi mendapat manfaat dari salinan defensif Subbab 13.8: sebuah ekspor menerima potret yang imutabel, sehingga apa yang diunduh pengguna persis sama dengan apa yang tersimpan pada saat itu. Seperti biasa, angka-angka dasbor adalah nasihat: platform menjelaskan apa yang diukurnya; penggunalah yang memutuskan apa yang akan diubah.

13.13 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya

1. Memanggil `next()` dua kali dalam satu badan perulangan. `while (it.hasNext()) { if (it.next().isBig()) list.add(it.next()); }` menguji satu elemen tetapi menambahkan elemen berikutnya — dan dapat melemparkan `NoSuchElementException` di ujungnya. *Hindari:* panggil `next()` tepat sekali per iterasi dan simpan hasilnya dalam variabel.

2. Mengubah ukuran koleksi di dalam perulangan for-each. Di Java hal ini biasanya melemparkan `ConcurrentModificationException` dan kadang diam-diam melompati satu elemen (blok 4 **Kode 13.7**); di Python hal ini diam-diam melompati elemen pada list dan membangkitkan `RuntimeError` pada dict atau set. *Hindari:* pakai `Iterator.remove()`, `removeIf`, comprehension, atau perulangan atas salinan (**Tabel 13.2**).

3. Menelusuri iterator Python dua kali. Generator atau hasil `map`, `filter`, `zip`, maupun `iter(...)` kosong pada kali kedua, tanpa galat apa pun. *Hindari:* peganglah iterable (list, tuple, objek ber-`__iter__`), atau panggil `list(...)` sekali lalu pakai ulang list-nya.

4. Menulis `compareTo` atau `compare` yang hanya mengembalikan −1, 0, 1 — atau menganggapnya demikian. Metode pembanding boleh mengembalikan `int` berapa pun; hanya tandanya yang penting. Kode yang melakukan `switch` pada −1 dan 1 melanggar kontrak Persamaan (13.3), dan hasil "terurut"-nya bergantung pada urutan masukan (**Kode 13.11**). *Hindari:* teruskan hasil `compareTo` apa adanya, atau bangun comparator dengan `Comparator.comparing(...).thenComparing(...)` (Python: kunci tuple).

5. Membandingkan dengan pengurangan. `return a - b;` dapat meluap (Persamaan (13.5)). *Hindari:* `Integer.compare(a, b)`, `Long.compare`, atau `Comparator.comparingInt`.

6. Mengira salinan dangkal itu mandiri. `new ArrayList<>(people)` dan `list(people)` menyalin list-nya, bukan orang-orang di dalamnya. *Hindari:* pakai salinan dalam (konstruktor salinan di Java, `copy.deepcopy` di Python) bila elemennya mutabel — atau, lebih baik lagi, jadikan elemennya imutabel, maka salinan dangkal sudah cukup.

7. Percaya bahwa `final` membuat objek imutabel. `final List` tetap dapat diubah; hanya referensinya yang terkunci. *Hindari:* pakai `List.copyOf` atau `List.of` untuk isinya, dan ingatlah bahwa list yang tak dapat diubah pun hanya seimutabel elemen-elemennya.

8. Mencampuradukkan tampilan tak dapat diubah dengan salinan imutabel. `Collections.unmodifiableList(x)` dan `MappingProxyType(d)` memperlihatkan perubahan belakangan pada `x` atau `d`. *Hindari:* serahkan tampilan bila pihak lain perlu melihat data hidup; simpan salinan (`List.copyOf`, `tuple(...)`) bila *Anda* memerlukan data yang tidak pernah berubah.

9. Menyimpan atau mengembalikan field mutabel tanpa salinan defensif. `LeakyCustomer` pada **Kode 13.19** diubah dari luar dua kali. *Hindari:* salin argumen yang mutabel di konstruktor (konstruktor ringkas record, `__post_init__` dataclass) dan jangan pernah mengembalikan objek internalnya.

10. Mengubah objek selagi ia menjadi kunci di map atau elemen di set. Entrinya menjadi tak terjangkau (**Gambar 13.4**). *Hindari:* pakai kunci imutabel — string, record, dataclass beku, tuple — dan jadikan id yang tidak berubah sebagai kunci entitas.

11. Menaruh efek samping di dalam stream atau comprehension. `list.forEach(x -> other.add(...))` atau comprehension yang menambahkan ke list di luarnya adalah perulangan yang menyamar, dengan semua risiko perulangan tanpa kejelasannya. *Hindari:* biarkan `map` dan `filter` hanya menghitung, dan biarkan collector atau comprehension itu sendiri yang membangun hasilnya.

12. Memakai ulang stream. Stream Java melemparkan `IllegalStateException` pada operasi terminal keduanya. *Hindari:* panggil `collection.stream()` lagi untuk setiap kueri, atau kumpulkan stream ke dalam list lebih dulu.

13.14 Ringkasan Bab

- **Iterator** menelusuri koleksi satu elemen demi satu elemen tanpa menyingkapkan cara penyimpanannya. Java: `Iterator` dengan `hasNext()` dan `next()` (Persamaan (13.1)); Python: `iter()` dan `next()`, diakhiri dengan `StopIteration` (**Tabel 13.1**). Iterator biasa hanya bergerak maju dan hanya membaca; `ListIterator` juga dapat bergerak mundur, `set`, `add`, dan `remove`.

- **Iterable** menghasilkan iterator: `Iterable` di Java (yang membuat perulangan for-each berjalan), `__iter__` di Python. Perulangan for-each adalah perulangan iterator yang diterjemahkan. Iterable dapat ditelusuri berkali-kali; iterator hanya sekali.
- Untuk menulis koleksi iterable, pisahkan **koleksinya** (yang mendefinisikan data) dari **iteratornya** (posisi satu penelusuran); di Java, pakai kelas dalam yang privat. Di Python, **generator** — fungsi dengan `yield` — menuliskan iteratornya untuk Anda, dan berjalan secara malas.
- Mengubah ukuran koleksi selagi menelusurinya adalah kesalahan. Iterator Java mendeteksinya dengan `modCount` (Persamaan (13.2)) dan melemparkan `ConcurrentModificationException`, sebatas upaya terbaik; list Python diam-diam melompati elemen, sedangkan dict dan set membangkitkan `RuntimeError`. Pakai `Iterator.remove`, `removeIf`, comprehension, atau salinan (**Tabel 13.2**).
- `Comparable` memberi sebuah kelas urutan alaminya; `Comparator` adalah urutan yang diberikan dari luar. Perbandingan harus antisimetris dan transitif (Persamaan (13.3), Persamaan (13.4)); hanya tandanya yang penting; jangan pernah memakai pengurangan (Persamaan (13.5)). Python mengurutkan dengan **fungsi kunci**, kunci tuple untuk beberapa kriteria, dan `cmp_to_key` untuk comparator; pengurutannya, seperti milik Java, stabil.
- **Aliasing** berbagi segalanya, **salinan dangkal** berbagi elemen, **salinan dalam** tidak berbagi apa pun (**Gambar 13.3**). Dengan elemen imutabel, salinan dangkal sudah memadai.
- Objek **imutabel** tidak dapat berubah setelah dikonstruksi. Ia kunci yang aman, argumen yang aman, aman dibagi antar-thread, dan memelihara invariannya secara cuma-cuma. Aturannya: tanpa pengubah, kelas `final`, field `private final`, salinan defensif untuk bagian yang mutabel, konstruksi yang tuntas. `final` mengunci referensi, bukan objek; **tampilan** yang tak dapat diubah memperlihatkan perubahan belakangan, **salinan** yang tak dapat diubah tidak (**Tabel 13.3**). **Record** dan **dataclass beku** membuat kelas nilai imutabel menjadi ringkas.
- **Objek nilai** ditentukan oleh field-fieldnya; nilai yang sama harus memiliki hash yang sama (Persamaan (13.6)), karena tabel hash hanya mencari di bucket Persamaan (13.7). Kunci yang berubah selagi tersimpan menjadi hilang (**Gambar 13.4**); jadikan kunci imutabel.
- **Stream** (Java) serta **comprehension, ekspresi generator, dan** `itertools` (Python) mengungkapkan perhitungan sebagai pipeline filter–map–reduce (Persamaan (13.8), **Gambar 13.5**, **Tabel 13.4**). Semuanya malas, sekali pakai, dan paling baik tanpa efek samping.
- Di binatu, `Money`, `Price`, `OrderLine`, dan `Order` adalah nilai imutabel, `OrderHistory` adalah iterable imutabel dengan iterator terbaru-lebih-dulu dan per pelanggan, dan laporannya berupa sekumpulan pipeline (**Gambar 13.6**).

13.15 Soal Latihan

Soal Latihan 13.1 (menelusuri iterator). Untuk list `[10, 20, 30]`, tuliskan keluaran penggalan Java berikut, atau sebutkan di mana dan mengapa ia gagal. Lalu tulislah padanannya dalam Python dengan `iter` dan `next`, dan ramalkan keluarannya.

```
Iterator<Integer> it = list.iterator();
System.out.println(it.next());
System.out.println(it.hasNext());
System.out.println(it.next() + it.next());
System.out.println(it.hasNext());
System.out.println(it.next());
```

Soal Latihan 13.2 (ListIterator). Dengan sebuah `ListIterator`, tulislah metode Java `void doubleLong(List<String> list)` yang menyisipkan salinan setiap elemen yang panjangnya lebih dari tiga karakter tepat sesudah elemen itu, dalam satu lintasan, sehingga `[wash, dry, iron]`

menjadi `[wash, wash, dry, iron, iron]`. Tulislah padanannya dalam Python. Mengapa versi Python tidak boleh sekadar menelusuri list sambil menyisipkan?

Soal Latihan 13.3 (iterable buatan sendiri). Binatu menjemput dan mengantar setiap hari kecuali hari Minggu. Tulislah `PickupDays`, sebuah iterable atas tanggal-tanggal dari tanggal pertama sampai tanggal terakhir (inklusif), yang melewati hari Minggu, dalam Java (mengimplementasikan `Iterable<LocalDate>`) dan dalam Python (dengan generator). Cetak hari-hari dari 25 September sampai 1 Oktober 2026 beserta nama harinya, dan hitunglah banyaknya hari penjemputan pada September 2026.

Soal Latihan 13.4 (modifikasi bersamaan). Untuk setiap penggalan, tentukan apakah ia melemparkan `ConcurrentModificationException`, diam-diam memberikan hasil yang salah, atau berjalan baik, untuk list `[1, 2, 3, 4]`; jelaskan dengan Persamaan (13.1) dan Persamaan (13.2). (a) `for (Integer x : list) if (x == 3) list.remove(x);` (b) `for (Integer x : list) if (x == 2) list.remove(x);` (c) `for (Integer x : list) if (x == 4) list.add(5);` (d) `for (int i = 0; i < list.size(); i++) if (list.get(i) % 2 == 0) list.remove(i);` Perhatikan bahwa `list.remove(x)` dengan argumen `Integer` menghapus berdasarkan nilai.

Soal Latihan 13.5 (comparator). Tulislah comparator, di Java dengan `Comparator.comparing`/`thenComparing` dan di Python dengan fungsi kunci, yang mengurutkan objek `Order` milik binatu (a) menurut total, terbesar lebih dulu, id sebagai penentu bila seri; (b) menurut nama pelanggan, lalu menurut tanggal, terbaru lebih dulu; (c) anggota lebih dulu, lalu menurut berat. Kemudian tulislah, untuk data nama di **Kode 13.11**, comparator yang mengurutkan menurut nama belakang lalu nama depan *tanpa memedulikan huruf besar-kecil*.

Soal Latihan 13.6 (kontrak). Tunjukkan bahwa perbandingan `(a, b) -> a - b` untuk `int` melanggar Persamaan (13.3) dengan memberikan nilai-nilai konkret, dan jelaskan mengapa `Integer.compare(a, b)` tidak melanggarnya. Kemudian tunjukkan bahwa perbandingan kode cucian berikut melanggar kontrak: `compare(a, b)` mengembalikan -1 jika `a.length() < b.length()`, 1 jika `a.charAt(0) > b.charAt(0)`, dan 0 untuk kasus lainnya, diperiksa dengan urutan itu. Temukan kode-kode di antara `"WF"`, `"IR"`, `"WDI"`, `"DC"` yang memperlihatkan pelanggarannya.

Soal Latihan 13.7 (mutabel atau imutabel?). Untuk setiap hal berikut, tentukan apakah berguna untuk menjadikannya mutabel atau imutabel, dan mengapa: (a) sebuah warna; (b) sebuah rekening bank; (c) koordinat 2D atau "titik"; (d) kelas `Integer`, yang memegang sebuah nilai `int` (berguna untuk generik, karena generik tidak dapat memegang tipe primitif); (e) kelas mahasiswa yang wajib memiliki nama, tanggal lahir, alamat, dan nomor ID.

Soal Latihan 13.8 (temukan kebocorannya). Kelas berikut dimaksudkan sebagai kelas imutabel. Sebutkan setiap cara yang memungkinkan kelas ini berubah setelah dikonstruksi, lalu tulislah ulang dengan benar dalam Java (sebagai record) dan dalam Python (sebagai dataclass beku).

```
public class Voucher {
    public final String code;
    private final java.util.Date expires;
    private final List<String> services;
    public Voucher(String code, java.util.Date expires, List<String> services) {
        this.code = code; this.expires = expires; this.services = services;
    }
    public java.util.Date getExpires() { return expires; }
    public List<String> getServices() { return services; }
}
```

Soal Latihan 13.9 (salinan). Ramalkan keluaran program Python berikut, lalu periksalah. Gambarkan objek-objeknya dengan gaya **Gambar 13.3**.

```
import copy
a = [[1], [2]]
b = a
c = a[:]
d = copy.deepcopy(a)
a[0].append(10)
a.append([3])
print(a, b, c, d)
```

Soal Latihan 13.10 (hashing). `String.hashCode` Java untuk `"WF"` adalah $31 \cdot 87 + 70 = 2767$ dan untuk `"IR"` adalah $31 \cdot 73 + 82 = 2345$. Dengan Persamaan (13.7) untuk $m = 16$, tentukan bucket masing-masing. Lalu jelaskan apa yang akan rusak seandainya `String` mutabel dan seseorang mengubah kunci `"WF"` menjadi `"IR"` setelah memakainya dalam sebuah `HashMap<String, Price>`.

Soal Latihan 13.11 (pertanyaan lanjutan untuk buku besar). Tanpa mengubah `LaundryLedger.java` maupun `laundry_ledger.py`, tulislah program yang mencetak: (a) total yang dibayar anggota dan bukan anggota; (b) satu baris pesanan termahal di seluruh riwayat; (c) banyaknya pesanan per hari dan hari tersibuk (yang paling awal bila seri); (d) untuk setiap pelanggan, himpunan terurut kode layanan yang pernah dipakainya; (e) id pesanan pertama yang totalnya melebihi Rp60,000, ditemukan secara malas.

Soal Latihan 13.12 (kemalasan). Di Java, `Stream.of(1, 2, 3, 4).filter(x -> { System.out.println("f" + x); return x % 2 == 0; }).map(x -> { System.out.println("m" + x); return x * 10; }).findFirst();` mencetak beberapa baris. Ramalkan baris-baris itu. Lalu tulislah pipeline yang sama dalam Python dengan ekspresi generator dan ramalkan apa yang dicetak `next(...)`. Apa yang akan dicetak seandainya versi Python memakai list comprehension?

Soal Latihan 13.13 (uang tanpa kehilangan satu rupiah pun). Tiga sahabat berbagi tagihan binatu sebesar Rp100,000. `Money.of(100_000 / 3)` adalah Rp33,333, dan tiga kali jumlah itu hanya Rp99,999. Tulislah fungsi `split(total, k)` yang mengembalikan k jumlah yang berselisih paling banyak satu rupiah dan berjumlah tepat sama dengan total, serta fungsi `allocate(total, weights...)` yang membagi secara sebanding dengan bobot bilangan bulat dan memberikan sisa rupiah kepada bagian-bagian pertama. Pakailah `Money` dari Subbab 13.11 tanpa mengubahnya. Terapkan `allocate` pada pendapatan buku besar dengan bobot 50, 30, 20.

13.16 Penyelesaian Soal Latihan

Penyelesaian 13.1. Penggalan itu mencetak `10`, `true`, `50` (20 + 30), `false`, lalu `it.next()` yang terakhir melemparkan `NoSuchElementException`, karena kursor berada di celah 3 dari 3 dan Persamaan (13.1) menyatakan tidak ada elemen berikutnya. Dalam Python:

```
it = iter([10, 20, 30])
print(next(it))              # 10
print(next(it) + next(it))   # 50
print(next(it, "done"))      # done   (tanpa nilai bawaan: StopIteration)
```

Python tidak memiliki `hasNext`, sehingga kedua baris `hasNext` tidak memiliki padanan langsung; `next(it, "done")` adalah cara pemeriksaan yang idiomatis.

Penyelesaian 13.2. Dalam Java:

```
static void doubleLong(List<String> list) {
    ListIterator<String> it = list.listIterator();
    while (it.hasNext()) {
        String s = it.next();
        if (s.length() > 3) {
            it.add(s);   // disisipkan tepat sebelum kursor, yakni tepat sesudah s
        }
    }
}
```

`add` menyisipkan elemen baru *sebelum* kursor, yang sesudah `next()` berarti tepat sesudah `s`, dan meninggalkan kursor sesudah salinan yang disisipkan; `next()` berikutnya karena itu mengembalikan elemen yang semula mengikuti `s`, dan salinannya tidak pernah dikunjungi. Karena perubahan dilakukan melalui iterator, tidak ada `ConcurrentModificationException`. Untuk `[wash, dry, iron]` hasilnya `[wash, wash, dry, iron, iron]`. Dalam Python, bangunlah list baru, `result = [y for x in items for y in ((x, x) if len(x) > 3 else (x,))]`, dan jika objek list yang lama harus diperbarui, tugaskan `items[:] = result`. Menyisipkan ke dalam list selagi perulangan `for` menelusurinya akan membuat perulangan mengunjungi setiap salinan yang disisipkan — yang juga lebih panjang dari tiga karakter dan akan disalin lagi, tanpa akhir — masalah Subbab 13.5 dalam bentuknya yang paling dramatis.

Penyelesaian 13.3. **Kode 13.27** dan **Kode 13.28**.

Kode 13.27 `PickupDays.java`: `Iterable<LocalDate>` yang melewati hari Minggu, dengan iterator dalam anonim.

```
import java.time.DayOfWeek;
import java.time.LocalDate;
import java.util.Iterator;
import java.util.NoSuchElementException;

public class PickupDays implements Iterable<LocalDate> {
    private final LocalDate first;
    private final LocalDate last;

    public PickupDays(LocalDate first, LocalDate last) {
        if (last.isBefore(first)) {
            throw new IllegalArgumentException("last is before first");
        }
        this.first = first;
        this.last = last;
    }

    private static LocalDate skipSunday(LocalDate day) {
        return day.getDayOfWeek() == DayOfWeek.SUNDAY ? day.plusDays(1) : day;
    }

    @Override
    public Iterator<LocalDate> iterator() {
        return new Iterator<>() {
            private LocalDate next = skipSunday(first);   // LocalDate imutabel

            @Override
            public boolean hasNext() {
                return !next.isAfter(last);
            }

            @Override
            public LocalDate next() {
                if (!hasNext()) {
                    throw new NoSuchElementException();
                }
                LocalDate result = next;
                next = skipSunday(next.plusDays(1));      // tanggal baru; result tidak
tersentuh
                return result;
            }
        };
    }
```

Kode 13.27 (Lanjutan) *PickupDays.java: Iterable<LocalDate> yang melewati hari Minggu, dengan iterator dalam anonim.*

```
    public static void main(String[] args) {
        PickupDays days = new PickupDays(LocalDate.of(2026, 9, 25), LocalDate.of(2026, 10,
1));
        for (LocalDate d : days) {
            System.out.println(d + " " + d.getDayOfWeek());
        }
        int n = 0;
        for (LocalDate ignored : new PickupDays(LocalDate.of(2026, 9, 1),
                                                LocalDate.of(2026, 9, 30))) {
            n++;
        }
        System.out.println("pickup days in September 2026: " + n);
    }
}
```

```
2026-09-25 FRIDAY
2026-09-26 SATURDAY
2026-09-28 MONDAY
2026-09-29 TUESDAY
2026-09-30 WEDNESDAY
2026-10-01 THURSDAY
pickup days in September 2026: 26
```

Kode 13.28 `pickup_days.py`: iterable yang sama sebagai generator.

```
"""pickup_days.py: iterable tanggal yang melewati hari Minggu."""
from datetime import date, timedelta


class PickupDays:
    def __init__(self, first, last):
        if last < first:
            raise ValueError("last is before first")
        self.first = first
        self.last = last

    def __iter__(self):
        day = self.first
        while day <= self.last:
            if day.weekday() != 6:          # Senin adalah 0, Minggu adalah 6
                yield day
            day += timedelta(days=1)        # objek tanggal baru; yang lama tidak tersentuh


def main():
    for d in PickupDays(date(2026, 9, 25), date(2026, 10, 1)):
        print(d, d.strftime("%A").upper())
    n = sum(1 for _ in PickupDays(date(2026, 9, 1), date(2026, 9, 30)))
    print("pickup days in September 2026:", n)


if __name__ == "__main__":
    main()
```

```
2026-09-25 FRIDAY
2026-09-26 SATURDAY
2026-09-28 MONDAY
2026-09-29 TUESDAY
2026-09-30 WEDNESDAY
2026-10-01 THURSDAY
pickup days in September 2026: 26
```

Iterator Java menyimpan tanggal berikutnya yang akan dikembalikan, yang sudah digeser melewati hari Minggu mana pun; `LocalDate` imutabel, sehingga `next.plusDays(1)` menciptakan tanggal baru dan `result` yang dikembalikan tidak akan pernah diubah oleh iterator di kemudian hari — manfaat imutabilitas yang kecil tetapi nyata. September 2026 memiliki 30 hari, empat di antaranya (tanggal 6, 13, 20, dan 27) hari Minggu, sehingga tersisa 26 hari penjemputan. Generator Python lebih sederhana lagi: telusuri hari-harinya dan `yield` yang bukan hari Minggu (`weekday()` bernilai 6 untuk hari Minggu).

Penyelesaian 13.4. (a) Menghapus 3, elemen *kedua dari akhir*: setelah `next()` mengembalikan 3, kursor $c = 3$; list menyusut menjadi $n = 3$; `hasNext()` adalah $3 \neq 3$, salah; perulangan berakhir tanpa memeriksa Persamaan (13.2), dan 4 tidak pernah dikunjungi. Ia *berjalan secara kebetulan* — tanpa eksepsi, dan dalam kasus ini hasil `[1, 2, 4]` bahkan benar, yang membuat *bug*-nya lebih sulit diperhatikan. (b) Menghapus 2: $c = 2$, n menjadi 3, `hasNext()` benar, dan `next()` mendapati `modCount` $\neq$ `expectedModCount`: `ConcurrentModificationException`. (c)

Menambahkan sesudah elemen terakhir: setelah `next()` mengembalikan 4, $c = 4$, n menjadi 5, `hasNext()` benar, `next()` memeriksa Persamaan (13.2): eksepsi. (d) Perulangan indeks tidak memiliki iterator, sehingga tidak ada yang dilemparkan, tetapi ia melompat: pada $i = 1$ ia menghapus 2, 3 berpindah ke indeks 1, i menjadi 2 dan melihat 4, yang kemudian dihapus; hasilnya `[1, 3]`, benar secara kebetulan. Dengan `[2, 4, 6]` hasilnya `[4]`. Menelusuri indeks secara menurun itu benar (Subbab 13.3.2); `list.removeIf(x -> x % 2 == 0)` paling sederhana.

Penyelesaian 13.5. Dalam Java, dengan record-record pada **Kode 13.25**:

```
Comparator<Order> a = Comparator.comparing(Order::total).reversed()
        .thenComparing(Order::id);
Comparator<Order> b = Comparator.comparing(Order::customer)
        .thenComparing(Order::date, Comparator.reverseOrder());
Comparator<Order> c = Comparator.comparing((Order o) -> !o.member())
        .thenComparingInt(Order::weightTenths);
Comparator<Name> ci = Comparator.comparing(Name::lastName, String.CASE_INSENSITIVE_ORDER)
        .thenComparing(Name::firstName, String.CASE_INSENSITIVE_ORDER);
```

Pada (c), `false` diurutkan sebelum `true`, sehingga `!member` menaruh anggota di depan. Dalam Python: (a) `key=lambda o: (-o.total().rupiah, o.id)`; (b) urutkan dua kali dengan mengandalkan kestabilan: pertama `sort(key=attrgetter("date"), reverse=True)`, lalu `sort(key=attrgetter("customer"))` — atau pakai `key=lambda o: (o.customer, -o.date.toordinal())`; (c) `key=lambda o: (not o.member, o.weight_tenths())`; urutan nama tanpa memedulikan huruf besar-kecil adalah `key=lambda n: (n.last_name.casefold(), n.first_name.casefold())`. Menegatifkan bilangan membalik satu bagian kunci tuple; untuk string, pakailah pengurutan stabil dua lintasan.

Penyelesaian 13.6. Ambil $a = 2^{31} - 1$ (`Integer.MAX_VALUE`) dan $b = -1$. Maka $a - b$ meluap menjadi -2^{31} (Persamaan (13.5)), sehingga `cmp(a, b) < 0`; dan $b - a = -2^{31}$, yang dapat dinyatakan, sehingga `cmp(b, a) < 0` juga. Keduanya berkata "lebih kecil", melanggar Persamaan (13.3). `Integer.compare(a, b)` diimplementasikan sebagai `(a < b) ? -1 : ((a == b) ? 0 : 1)`, yang membandingkan alih-alih mengurangkan dan tidak mungkin meluap.

Untuk kode-kode cucian, hitunglah kedua arah untuk satu pasangan. `compare("WF", "IR")`: panjangnya sama, sehingga aturan pertama tidak berlaku; 'W' > 'I', sehingga hasilnya 1. `compare("IR", "WF")`: panjangnya lagi-lagi sama; 'I' tidak lebih besar daripada 'W', sehingga hasilnya 0. Antisimetri menuntut -1, jadi Persamaan (13.3) dilanggar. Aturan ketiga kontrak pun gagal: `compare("WDI", "WF")` bernilai 0 ("setara"), sehingga `"WDI"` dan `"WF"` semestinya dibandingkan dengan cara yang sama terhadap setiap kode lain; padahal `compare("IR", "WDI")` bernilai -1 (lebih pendek) sementara `compare("IR", "WF")` bernilai 0. Perbandingan yang dirakit dari aturan-aturan seadanya yang diperiksa "dengan urutan tertentu" hampir tidak pernah membentuk urutan yang sah; bangunlah perbandingan dari kunci, field demi field.

Penyelesaian 13.7. (a) **Warna: imutabel.** Warna adalah nilai; "mengubah" merah menjadi biru berarti memakai warna lain, dan warna dipakai bersama di mana-mana (`java.awt.Color` di Java imutabel). (b) **Rekening bank: mutabel**, atau tepatnya sebuah *entitas*: saldonya harus berubah dari waktu ke waktu, dan ia memiliki identitas — rekening yang ini. Namun *transaksi*-transaksinya sebaiknya berupa nilai imutabel, dan saldonya dapat diturunkan dari transaksi itu, seperti `OrderHistory` milik binatu. (c) **Titik: imutabel.** Ia sebuah nilai, dan titik adalah kunci yang alami (map dari posisi ke mesin). (d) **`Integer`: imutabel** — dan memang demikian. Objek `Integer` di-*cache* dan dipakai bersama (untuk nilai kecil, `Integer.valueOf(7)` selalu mengembalikan objek yang sama); seandainya satu di antaranya dapat diubah, setiap angka 7 dalam program akan ikut berubah. (e) **Mahasiswa: sebagian besar imutabel.** Nama, tanggal lahir, dan ID mengidentifikasi mahasiswa dan semestinya `final`; ID tidak boleh berubah karena ia

menjadi kunci di setiap map mahasiswa. Alamat memang berubah, sehingga kelas itu menawarkan `withAddress` yang mengembalikan mahasiswa baru, atau alamatnya disimpan dalam map tersendiri dari ID mahasiswa ke alamat. Field-fieldnya sendiri harus berupa tipe primitif atau imutabel (`int`, `String`, `LocalDate` — bukan `java.util.Date`).

Penyelesaian 13.8. Kebocorannya: (1) konstruktor menyimpan `Date` milik pemanggil, yang dapat diubah pemanggil dengan `setTime`; (2) `getExpires()` mengembalikan `Date` mutabel yang sama; (3) konstruktor menyimpan list milik pemanggil; (4) `getServices()` mengembalikan list internalnya; (5) kelas itu tidak `final`, sehingga sebuah subkelas dapat menambahkan keadaan mutabel atau meng-*override* getter-nya. (`code` bersifat `public final` dan bertipe `String`, yang aman, meskipun gayanya tidak lazim.) Versi yang benar:

```
public record Voucher(String code, LocalDate expires, List<String> services) {
    public Voucher {
        Objects.requireNonNull(code);
        Objects.requireNonNull(expires);          // LocalDate imutabel: tidak perlu
disalin
        services = List.copyOf(services);         // salinan defensif yang tak dapat
diubah
    }
}
```

```
@dataclass(frozen=True)
class Voucher:
    code: str
    expires: date
    services: tuple
    def __post_init__(self):
        object.__setattr__(self, "services", tuple(self.services))
```

Mengganti `java.util.Date` (mutabel) dengan `java.time.LocalDate` (imutabel) menutup dua kebocoran sekaligus; jika Anda terpaksa mempertahankan `Date`, salinlah ke dua arah dengan `new Date(d.getTime())`.

Penyelesaian 13.9. Program itu mencetak `[[1, 10], [2], [3]] [[1, 10], [2], [3]] [[1, 10], [2]] [[1], [2]]`. `b` adalah alias dari `a`, sehingga ia memperlihatkan kedua perubahan. `c` adalah salinan dangkal: list luar yang berbeda (sehingga tidak memuat `[3]`) yang memegang list-list dalam yang sama (sehingga memperlihatkan angka 10). `d` adalah salinan dalam dan tidak memperlihatkan satu pun perubahan.

Penyelesaian 13.10. $2767 = 172 \cdot 16 + 15$, sehingga `"WF"` masuk ke bucket 15; $2345 = 146 \cdot 16 + 9$, sehingga `"IR"` masuk ke bucket 9. (Java juga mencampurkan $h \gg 16$, yang bernilai 0 untuk kode-kode kecil ini.) Seandainya objek kunci `"WF"` yang tersimpan dapat diubah menjadi `"IR"`, entrinya akan tetap berada di bucket 15 sementara setiap pencarian `"IR"` pergi ke bucket 9 dan tidak menemukan apa-apa, dan pencarian `"WF"` akan sampai di bucket 15 tetapi gagal dalam uji `equals` — keadaan pada **Gambar 13.4**. Lebih buruk lagi, `put("IR", ...)` berikutnya akan menciptakan entri kedua dengan kunci yang setara. Inilah salah satu alasan `String` dibuat imutabel.

Penyelesaian 13.11. **Kode 13.29** dan **Kode 13.30** menjawab kelima pertanyaan dengan hanya memakai bagian-bagian publik buku besar.

Kode 13.29 `LedgerQueries.java`: lima pertanyaan lanjutan yang dijawab dengan stream, termasuk `partitioningBy`, `flatMap`, `max`, `counting`, dan `flatMapping`.

```
import java.time.LocalDate;
import java.util.Comparator;
import java.util.Map;
import java.util.Set;
import java.util.TreeMap;
import java.util.TreeSet;
import java.util.stream.Collectors;

// Memerlukan LaundryLedger.java di folder yang sama (kompilasi keduanya dengan javac
*.java).
public class LedgerQueries {
    public static void main(String[] args) {
        OrderHistory history = LaundryLedger.septemberHistory();

        // (a) Anggota versus bukan anggota: partitioningBy membelah stream menjadi dua.
        Map<Boolean, Money> byMembership = history.stream()
                .collect(Collectors.partitioningBy(Order::member,
                        Collectors.reducing(Money.ZERO, Order::total, Money::plus)));
        System.out.println("members " + byMembership.get(true)
                + ", others " + byMembership.get(false));

        // (b) Satu baris termahal di seluruh riwayat.
        OrderLine biggest = history.stream()
                .flatMap(o -> o.lines().stream())
                .max(Comparator.comparing(OrderLine::amount))
                .orElseThrow();
        System.out.println("biggest line: " + biggest.price().name() + " "
                + biggest.weightTenths() / 10.0 + " kg = " + biggest.amount());

        // (c) Banyaknya pesanan per hari, dan hari tersibuk.
        Map<LocalDate, Long> ordersPerDay = history.stream()
                .collect(Collectors.groupingBy(Order::date, TreeMap::new,
                        Collectors.counting()));
        LocalDate busiest = ordersPerDay.entrySet().stream()
                .max(Map.Entry.<LocalDate, Long>comparingByValue()

.thenComparing(Map.Entry.comparingByKey(Comparator.reverseOrder())))
                .orElseThrow().getKey();
        System.out.println("orders per day " + ordersPerDay + ", busiest " + busiest);

        // (d) Himpunan layanan yang pernah dipakai setiap pelanggan, urut menurut nama.
        Map<String, Set<String>> services = history.stream()
                .collect(Collectors.groupingBy(Order::customer, TreeMap::new,
                        Collectors.flatMapping(o -> o.lines().stream().map(l ->
l.price().code()),
                                Collectors.toCollection(TreeSet::new))));
        services.forEach((name, codes) -> System.out.println("  " + name + " used " +
codes));
```

Kode 13.29 (Lanjutan) *LedgerQueries.java: lima pertanyaan lanjutan yang dijawab dengan stream, termasuk partitioningBy, flatMap, max, counting, dan flatMapping.*

```
        // (e) Pesanan pertama yang totalnya melebihi Rp60,000, ditemukan secara malas.
        String firstBig = history.stream()
                .filter(o -> o.total().compareTo(Money.of(60_000)) > 0)
                .map(Order::id)
                .findFirst()
                .orElse("none");
        System.out.println("first order over Rp60,000: " + firstBig
                + " (of " + history.size() + ")");
    }
}
```

```
members Rp106,650, others Rp223,000
biggest line: Wash & Fold 7.9 kg = Rp56,000
orders per day {2026-09-21=3, 2026-09-22=3, 2026-09-23=2}, busiest 2026-09-21
  Ayu Lestari used [WDI, WF]
  Budi Santoso used [IR, WDI]
  Dewi Anggraini used [IR, WF]
  Siti Aminah used [IR, WDI, WF]
first order over Rp60,000: ORD-1305 (of 8)
```

Kode 13.30 `ledger_queries.py`: kelima pertanyaan yang sama dengan comprehension, `max` dengan kunci, `Counter`, dan `next` dengan nilai bawaan.

```
"""ledger_queries.py: pertanyaan lanjutan untuk buku besar binatu (memerlukan
laundry_ledger.py)."""
from collections import Counter

from laundry_ledger import ZERO, september_history


def main():
    history = september_history()
    orders = list(history.oldest_first())

    # (a) Anggota versus bukan anggota.
    members = sum((o.total() for o in orders if o.member), ZERO)
    others = sum((o.total() for o in orders if not o.member), ZERO)
    print(f"members {members}, others {others}")

    # (b) Satu baris termahal di seluruh riwayat.
    biggest = max((line for o in orders for line in o.lines), key=lambda ln: ln.amount())
    print(f"biggest line: {biggest.price.name} {biggest.weight_tenths / 10} kg = "
          f"{biggest.amount()}")

    # (c) Banyaknya pesanan per hari, dan hari tersibuk (yang paling awal menang bila
seri).
    per_day = Counter(o.date for o in orders)
    busiest = min(per_day, key=lambda d: (-per_day[d], d))
    print("orders per day", {str(d): n for d, n in sorted(per_day.items())},
          "busiest", busiest)

    # (d) Himpunan layanan yang pernah dipakai setiap pelanggan, urut menurut nama.
    services = {name: sorted({ln.price.code for o in orders if o.customer == name
                              for ln in o.lines})
                for name in sorted({o.customer for o in orders})}
    for name, codes in services.items():
        print(f"  {name} used {codes}")

    # (e) Pesanan pertama yang totalnya melebihi Rp60,000, ditemukan secara malas.
    first_big = next((o.id for o in orders if o.total().rupiah > 60_000), "none")
    print(f"first order over Rp60,000: {first_big} (of {len(orders)})")


if __name__ == "__main__":
    main()
```

```
members Rp106,650, others Rp223,000
biggest line: Wash & Fold 7.9 kg = Rp56,000
orders per day {'2026-09-21': 3, '2026-09-22': 3, '2026-09-23': 2} busiest 2026-09-21
  Ayu Lestari used ['WDI', 'WF']
  Budi Santoso used ['IR', 'WDI']
  Dewi Anggraini used ['IR', 'WF']
  Siti Aminah used ['IR', 'WDI', 'WF']
first order over Rp60,000: ORD-1305 (of 8)
```

(a) `partitioningBy` adalah `groupingBy` dengan kunci `boolean`; Rp106,650 ditambah Rp223,000 adalah pendapatan sebesar Rp329,650. (b) Baris terbesar adalah Wash & Fold 7.9 kg milik

ORD-1305: $\lceil 79/5 \rceil = 16$, jadi 80 persepuluhan dan $7000 \cdot 80/10 = 56{,}000$. (c) Untuk mengutamakan hari seri yang paling awal, Java memutus seri dengan tanggal *terbalik* di dalam `max`; Python meminimumkan `(-count, date)`. (d) `flatMapping` (Java 9+) mengumpulkan kode setiap pelanggan ke dalam `TreeSet`; Python menyarangkan dua klausa `for`. (e) `findFirst` dan `next(..., "none")` berhenti di ORD-1305.

Penyelesaian 13.12. Stream Java meneruskan satu elemen setiap kali melalui seluruh pipeline, sehingga keluarannya `f1`, `f2`, `m2`, lalu `findFirst` sudah mendapat jawabannya dan stream berhenti: 1 gagal melewati filter, 2 lolos dan dipetakan, sedangkan 3 dan 4 tidak pernah tersentuh. (Jika Anda meramalkan `f1 f2 f3 f4 m2 m4`, Anda membayangkan setiap operasi berjalan atas seluruh list sebelum operasi berikutnya dimulai, yaitu cara kerja list yang rajin.) Dalam Python, dengan fungsi pembantu `f(x)` dan `m(x)` yang mencetak lalu mengembalikan nilai, `next(m(x) for x in [1, 2, 3, 4] if f(x))` mencetak `f1 f2 m2` yang sama, karena ekspresi generator juga malas. List comprehension `[m(x) for x in [1, 2, 3, 4] if f(x)][0]` berjalan sampai habis sebelum elemen pertamanya dapat diambil, dan mencetak `f1 f2 m2 f3 f4 m4`.

Penyelesaian 13.13. **Kode 13.31** dan **Kode 13.32**. Untuk membagi total T menjadi k bagian, misalkan

$$q = \left\lfloor \frac{T}{k} \right\rfloor, \qquad r = T - kq, \qquad \text{bagian} = \underbrace{q+1, \ldots, q+1}_{r}, \underbrace{q, \ldots, q}_{k-r} \tag{13.10}$$

Maka jumlah bagian-bagian itu adalah $r(q+1) + (k-r)q = kq + r = T$ secara tepat, dan setiap dua bagian berselisih paling banyak 1 (Persamaan (13.10)).

Kode 13.31 `MoneySplit.java`: membagi dan mengalokasikan `Money` sehingga bagian-bagiannya selalu berjumlah sama dengan total.

```
import java.util.ArrayList;
import java.util.List;

// Memerlukan LaundryLedger.java di folder yang sama (berkas itu mendefinisikan record
Money).
public class MoneySplit {
    // k bagian yang berselisih paling banyak satu rupiah dan berjumlah tepat sama dengan
total.
    static List<Money> split(Money total, int k) {
        if (k <= 0) {
            throw new IllegalArgumentException("k must be positive");
        }
        long q = total.rupiah() / k;
        long r = total.rupiah() % k;             // r bagian mendapat satu rupiah lebih
        List<Money> parts = new ArrayList<>();
        for (int i = 0; i < k; i++) {
            parts.add(Money.of(i < r ? q + 1 : q));
        }
        return List.copyOf(parts);
    }

    // Bagian sebanding dengan bobot; sisa rupiah diberikan kepada bagian-bagian pertama.
    static List<Money> allocate(Money total, int... weights) {
        long sum = 0;
        for (int w : weights) {
            sum += w;
        }
        List<Money> parts = new ArrayList<>();
        long given = 0;
        for (int w : weights) {
            long share = total.rupiah() * w / sum;
            parts.add(Money.of(share));
            given += share;
        }
        for (int i = 0; given < total.rupiah(); i++, given++) {
            parts.set(i, parts.get(i).plus(Money.of(1)));
        }
        return List.copyOf(parts);
    }

    static Money sum(List<Money> parts) {
        return parts.stream().reduce(Money.ZERO, Money::plus);
    }
```

Kode 13.31 (Lanjutan) *MoneySplit.java: membagi dan mengalokasikan Money sehingga bagian-bagiannya selalu berjumlah sama dengan total.*

```
    public static void main(String[] args) {
        Money bill = Money.of(100_000);
        List<Money> three = split(bill, 3);
        System.out.println("split in 3:     " + three + " sum " + sum(three));
        System.out.println("naive third x3: " + Money.of(bill.rupiah() / 3).times(3));
        List<Money> shares = allocate(Money.of(329_650), 50, 30, 20);
        System.out.println("50/30/20:       " + shares + " sum " + sum(shares));
        List<Money> odd = allocate(Money.of(10), 1, 1, 1);
        System.out.println("Rp10 by 1:1:1:  " + odd + " sum " + sum(odd));
    }
}
```

```
split in 3:     [Rp33,334, Rp33,333, Rp33,333] sum Rp100,000
naive third x3: Rp99,999
50/30/20:       [Rp164,825, Rp98,895, Rp65,930] sum Rp329,650
Rp10 by 1:1:1:  [Rp4, Rp3, Rp3] sum Rp10
```

Kode 13.32 `money_split.py`: hal yang sama dengan `divmod` dan tuple berisi nilai `Money`.

```
"""money_split.py: membagi jumlah uang tanpa kehilangan satu rupiah pun (memerlukan
laundry_ledger.py)."""
from laundry_ledger import ZERO, Money


def split(total, k):
    """k bagian yang berselisih paling banyak satu rupiah dan berjumlah tepat sama dengan
total."""
    if k <= 0:
        raise ValueError("k must be positive")
    q, r = divmod(total.rupiah, k)            # r bagian mendapat satu rupiah lebih
    return tuple(Money(q + 1 if i < r else q) for i in range(k))


def allocate(total, *weights):
    """Bagian sebanding dengan bobot; sisa rupiah diberikan kepada bagian-bagian
pertama."""
    s = sum(weights)
    shares = [total.rupiah * w // s for w in weights]
    for i in range(total.rupiah - sum(shares)):
        shares[i] += 1
    return tuple(Money(x) for x in shares)


def show(parts):
    return "[" + ", ".join(str(p) for p in parts) + "]"


def main():
    bill = Money(100_000)
    three = split(bill, 3)
    print("split in 3:    ", show(three), "sum", sum(three, ZERO))
    print("naive third x3:", Money(bill.rupiah // 3) * 3)
    shares = allocate(Money(329_650), 50, 30, 20)
    print("50/30/20:      ", show(shares), "sum", sum(shares, ZERO))
    odd = allocate(Money(10), 1, 1, 1)
    print("Rp10 by 1:1:1: ", show(odd), "sum", sum(odd, ZERO))


if __name__ == "__main__":
    main()
```

```
split in 3:     [Rp33,334, Rp33,333, Rp33,333] sum Rp100,000
naive third x3: Rp99,999
50/30/20:       [Rp164,825, Rp98,895, Rp65,930] sum Rp329,650
Rp10 by 1:1:1:  [Rp4, Rp3, Rp3] sum Rp10
```

Untuk Rp100,000 dibagi 3, $q = 33{,}333$ dan $r = 1$, sehingga bagian-bagiannya Rp33,334, Rp33,333, Rp33,333. `allocate` mula-mula memberi setiap bagian proporsinya yang dibulatkan ke bawah; bagian-bagian itu lalu kurang dari total sebanyak sejumlah rupiah yang lebih sedikit daripada banyaknya bagian, dan perulangan membagikan sisa itu satu per satu. Untuk 50/30/20 dari Rp329,650 proporsinya eksak, sehingga tidak ada sisa yang perlu dibagi; untuk Rp10 dalam tiga bagian sama, satu rupiah sisa diberikan kepada bagian pertama. Hasilnya imutabel (`List.copyOf`, `tuple`), dan `Money` sama sekali tidak perlu diubah: tipe nilai yang imutabel dapat diperluas dengan *fungsi-fungsi* baru selamanya tanpa risiko apa pun bagi kode yang sudah ada. Fowler menyebut operasi ini *alokasi* dan menganjurkan persis perlakuan sisa seperti ini [19].

Daftar Pustaka

[1] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Reading, MA, USA: Addison-Wesley, 1994.

[2] B. Liskov and J. Guttag, *Program Development in Java: Abstraction, Specification, and Object-Oriented Design*. Boston, MA, USA: Addison-Wesley, 2000.

[3] Oracle, *Java Platform, Standard Edition 21 API Specification*, packages `java.util` and `java.util.stream`. [Online]. Available: https://docs.oracle.com/en/java/javase/21/docs/api/

[4] Python Software Foundation, *The Python Language Reference* and *The Python Standard Library*, version 3.14, sections "Iterator Types", `itertools`, `functools`, `copy` and `dataclasses`. [Online]. Available: https://docs.python.org/3/

[5] K.-P. Yee and G. van Rossum, "PEP 234 – Iterators," Python Enhancement Proposal 234, 2001. [Online]. Available: https://peps.python.org/pep-0234/

[6] J. Gosling, B. Joy, G. Steele, G. Bracha, A. Buckley, D. Smith, and G. Bierman, *The Java Language Specification, Java SE 21 Edition*. Oracle America, 2023. [Online]. Available: https://docs.oracle.com/javase/specs/jls/se21/html/

[7] N. Schemenauer, T. Peters, and M. L. Hetland, "PEP 255 – Simple generators," Python Enhancement Proposal 255, 2001. [Online]. Available: https://peps.python.org/pep-0255/

[8] L. Ramalho, *Fluent Python*, 2nd ed. Sebastopol, CA, USA: O'Reilly Media, 2022.

[9] J. Bloch, *Effective Java*, 3rd ed. Boston, MA, USA: Addison-Wesley, 2018.

[10] T. Peters, "listsort.txt: description of the list sort algorithm (Timsort)," CPython source distribution, 2002. [Online]. Available: https://github.com/python/cpython/blob/main/Objects/listsort.txt

[11] Oracle, "Immutable objects," *The Java Tutorials*, Concurrency lesson. [Online]. Available: https://docs.oracle.com/javase/tutorial/essential/concurrency/immutable.html

[12] B. Goetz, T. Peierls, J. Bloch, J. Bowbeer, D. Holmes, and D. Lea, *Java Concurrency in Practice*. Upper Saddle River, NJ, USA: Addison-Wesley, 2006.

[13] S. Marks, "JEP 269: Convenience factory methods for collections," OpenJDK, 2017. [Online]. Available: https://openjdk.org/jeps/269

[14] G. Bierman, "JEP 395: Records," OpenJDK, 2020. [Online]. Available: https://openjdk.org/jeps/395

[15] E. V. Smith, "PEP 557 – Data classes," Python Enhancement Proposal 557, 2017. [Online]. Available: https://peps.python.org/pep-0557/

[16] E. Evans, *Domain-Driven Design: Tackling Complexity in the Heart of Software*. Boston, MA, USA: Addison-Wesley, 2003.

[17] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 4th ed. Cambridge, MA, USA: MIT Press, 2022.

[18] R. Warburton, *Java 8 Lambdas: Functional Programming for the Masses*. Sebastopol, CA, USA: O'Reilly Media, 2014.

[19] M. Fowler, *Patterns of Enterprise Application Architecture*. Boston, MA, USA: Addison-Wesley, 2002.

Bab 14
Thread, Race, Deadlock, dan Livelock

Pada akhir bab ini Anda akan sanggup menjelaskan apa itu *thread* dan apa bedanya dengan sebuah program, serta mengatakan kapan menjalankan beberapa thread membuat program lebih cepat, lebih tanggap, atau sekadar lebih mudah ditulis; menciptakan thread di Java dengan `Thread`, `Runnable`, dan `ExecutorService`, dan di Python dengan `threading.Thread` dan `concurrent.futures`; menghentikan thread dengan sopan melalui interupsi, bendera `volatile`, atau `Event`; mengikuti sebuah thread sepanjang hidupnya, dari `NEW` sampai `TERMINATED`; menjelaskan apa itu Global Interpreter Lock milik Python dan meramalkan kapan thread Python membantu dan kapan tidak; menghasilkan ulang sebuah *race condition* (kondisi balapan) pada pencacah bersama, membaca instruksi mesin yang menyebabkannya, lalu mengobatinya dengan `synchronized`, `ReentrantLock`, `AtomicInteger`, atau `threading.Lock`; membuat thread saling menunggu dengan `wait`/`notifyAll` dan `Condition`, lalu membiarkan `BlockingQueue` atau `queue.Queue` mengurus penantian itu untuk Anda; mengenali *deadlock* (kebuntuan) pada para filsuf yang bersantap dan pada dua petugas yang masing-masing memegang satu mesin, membuktikannya dengan graf tunggu, dan mencegahnya dengan pengurutan kunci, dengan `tryLock` berbatas waktu, atau dengan seorang penengah; mengenali kegagalan yang lebih senyap bernama *livelock* dan *starvation* (kelaparan); serta membangun binatu kecil tempat tiga mesin cuci, masing-masing sebuah thread, mengambil pesanan dari satu antrean dan membayar ke satu kasir tanpa pernah kehilangan satu rupiah pun. Sepanjang bab ini kita jujur tentang satu sifat yang membedakan pokok bahasan ini dari semua bab sebelumnya: program konkuren dapat mencetak hal yang berbeda setiap kali dijalankan, dan kita akan memperlihatkan hasil jalannya program yang sungguhan serta mengatakan dengan tepat apa yang berubah-ubah dan apa yang tidak boleh berubah sama sekali.

14.1 Di Mana Kita Berhenti

Setiap program dalam buku ini sejauh ini mengerjakan satu hal pada satu waktu. Ketika `main` memanggil sebuah metode, `main` berhenti dan menunggu sampai metode itu kembali; ketika sebuah perulangan berjalan, tidak ada hal lain yang terjadi sampai perulangan itu selesai. Bab 10 adalah satu-satunya pengecualian yang tidak kita bicarakan. Ketika Anda mengeklik tombol di jendela Swing atau Tkinter, *handler* Anda berjalan, tetapi jendela itu tetap menggambar ulang dirinya, dan Anda tidak pernah menulis perulangan yang mengawasi tetikus. Ada pihak lain yang menjalankan perulangan itu untuk Anda. Di Swing pihak itu adalah *thread* tersendiri, Event Dispatch Thread; di Tkinter ia adalah `mainloop`. Bab 11 memecah program menjadi model, tampilan, dan pengendali, dan Bab 13 menjadikan objek imutabel, sebuah sifat yang ternyata menjadi salah satu pertahanan terbaik kita di bab ini.

Sekarang misalkan Profitina Laundry membeli dua mesin cuci lagi. Tiga mesin dapat mencuci tiga muatan pada saat yang sama. Namun perangkat lunak binatu itu masih mengerjakan satu hal pada satu waktu: memulai satu cucian, menunggunya, mencatat pembayarannya, dan baru kemudian memulai cucian berikutnya. Dua dari tiga mesin menganggur hampir sepanjang hari. Yang diperlukan binatu itu adalah perangkat lunak tempat setiap mesin memiliki program

kecilnya sendiri yang berjalan *bersamaan* dengan yang lain, semuanya membaca dari satu daftar pesanan yang menunggu dan semuanya membayar ke satu mesin kasir.

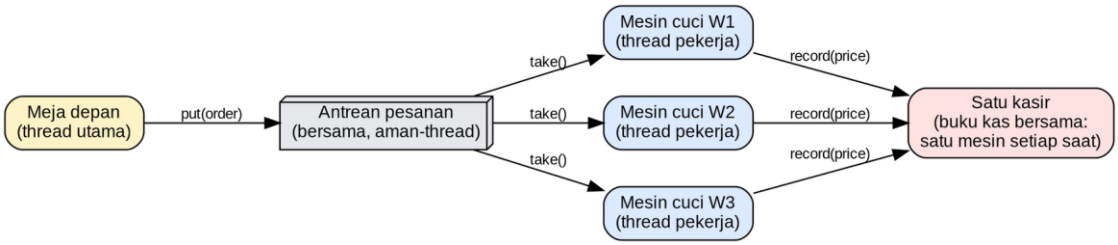
Itulah yang diberikan thread kepada kita, dan di situ pula masalah dimulai. Dua mesin yang mengambil pesanan yang sama, atau dua pembayaran yang ditambahkan ke kas pada saat yang sama dan hanya menyisakan salah satunya di dalam total, bukanlah *bug* khayalan. Kita akan menghasilkan keduanya dengan sengaja, di perangkat keras sungguhan, lalu mempelajari disiplin yang mencegahnya.

> **WAWASAN KUNCI:** Thread mudah *dimulai* dan sulit *dikoordinasikan*. Hampir semua yang salah dalam program konkuren berasal dari satu keadaan: dua thread menyentuh data yang sama, dan paling sedikit salah satunya mengubahnya. Setiap alat di bab ini (kunci, variabel atomik, kondisi, antrean, pengurutan kunci) ada untuk mengendalikan satu keadaan itu.

14.2 Apa Itu Thread?

Mari mulai dari binatunya sendiri, karena binatu adalah model komputer yang sangat baik. Bayangkan Profitina Laundry pada hari Sabtu yang ramai. Ada tiga mesin cuci. Setiap mesin mengerjakan pekerjaannya sendiri: menerima sekantong cucian, mencucinya, dan ketika selesai, seseorang membawa notanya ke kasir. Hanya ada **satu** kasir, dan kasir itu hanya dapat melayani satu orang pada satu waktu. Pelanggan membawa pesanan ke meja depan, dan pesanan itu menunggu di keranjang sampai ada mesin yang kosong.

Banyak hal di sini berlangsung *serentak*: ketiga mesin mencuci bersamaan, dan meja depan terus menerima pesanan baru. Namun ada hal yang harus terjadi *satu per satu*: kasir menulis satu pembayaran di buku, lalu yang berikutnya; dua orang yang menulis di baris yang sama pada saat yang sama akan membuat buku itu tak terbaca. **Gambar 14.1** menggambar binatu itu persis seperti ini, dan gambar itu juga, nyaris kata demi kata, rancangan program yang akan kita bangun di Subbab 14.14.



Gambar 14.1 Profitina Laundry sebagai sistem konkuren. Meja depan memasukkan pesanan ke satu antrean bersama; tiga mesin cuci, masing-masing sebuah thread pekerja, mengambil pesanan darinya; ketiganya membayar ke satu buku kas yang menerima satu mesin setiap saat.

Sekarang terjemahkan gambar itu ke dalam istilah komputasi.

> **DEFINISI.** Sebuah **proses** (*process*) adalah program yang sedang berjalan: sistem operasi memberinya memori sendiri, berkas terbuka sendiri, dan paling sedikit satu thread. Sebuah **thread** (utas) adalah satu urutan instruksi mandiri yang berjalan di dalam sebuah proses. Setiap thread memiliki *call stack* (tumpukan panggilan) sendiri, yakni variabel lokalnya sendiri dan catatannya sendiri tentang metode mana memanggil metode mana, tetapi semua thread dalam satu proses berbagi *heap* yang sama: objek yang sama, field yang sama, list yang sama.

Jadi setiap mesin cuci adalah sebuah thread dengan pekerjaannya sendiri di tangan (variabel lokalnya: pesanan yang mana, berapa kilogram). Keranjang pesanan dan buku kasir adalah objek

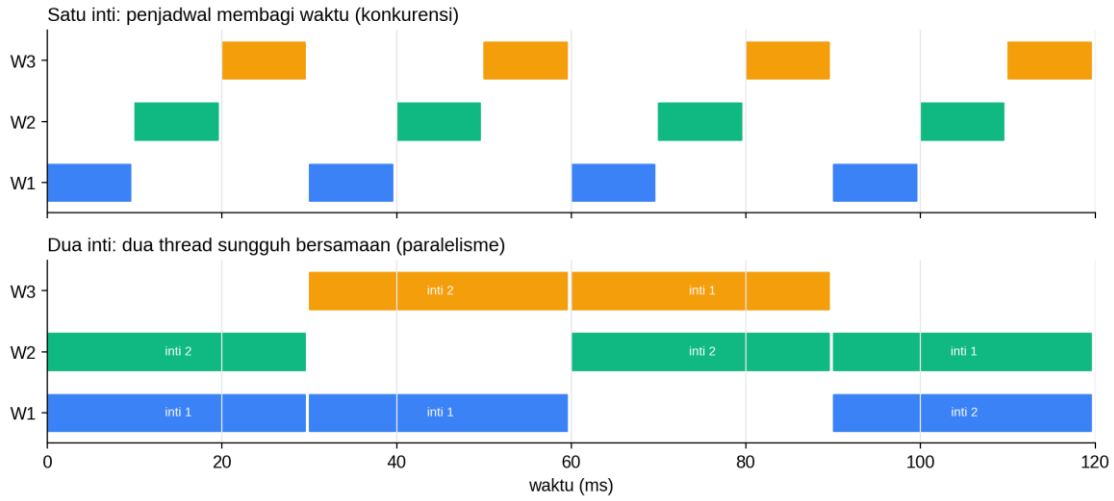
bersama di *heap* yang dapat dilihat dan diubah oleh setiap mesin. Berbagi seperti itulah yang membuat thread berguna (mesin-mesin bekerja sama melalui keranjang) sekaligus berbahaya (mereka dapat merusak buku dengan menulis pada saat yang sama).

Sebuah program selalu memiliki paling sedikit satu thread. Di Java namanya `main`; di Python, `MainThread`. Setiap program yang telah Anda tulis sejauh ini berjalan seluruhnya di satu thread itu. Di bab ini kita akan menciptakan lebih banyak.

14.2.1 Konkurensi dan paralelisme

Dua kata sering tertukar, dan ada baiknya kita pisahkan sekarang. **Konkurensi** (*concurrency*) berarti beberapa tugas *sedang berlangsung* pada saat yang sama: masing-masing sudah dimulai dan belum ada yang selesai. **Paralelisme** (*parallelism*) berarti beberapa tugas benar-benar *dieksekusi* pada detik yang sama, di inti prosesor yang berbeda.

Komputer dengan satu inti dapat menjalankan banyak thread secara konkuren, tetapi tidak pernah secara paralel. **Penjadwal** (*scheduler*) sistem operasi memberikan inti itu kepada satu thread selama sebuah *irisan waktu* (*time slice*) yang pendek (beberapa milidetik), lalu menghentikannya [1], menyimpan posisinya, dan memberikan inti itu kepada thread berikutnya. Karena irisannya pendek, seolah-olah semua thread berjalan bersamaan, seperti pemain sulap yang tampak memegang tiga bola padahal pada setiap saat hanya satu bola yang ada di tangannya. Komputer dengan dua, empat, atau enam belas inti dapat, di samping itu, menjalankan thread sebanyak itu secara sungguh-sungguh paralel. Bagian atas **Gambar 14.2** memperlihatkan tiga thread mesin berbagi satu inti; bagian bawahnya memperlihatkan ketiga thread yang sama pada dua inti, tempat dua di antaranya sungguh berjalan bersamaan dan yang ketiga menunggu gilirannya.



Gambar 14.2 Konkurensi dan paralelisme. Atas: satu inti; penjadwal memberi setiap thread irisan waktu pendek secara bergiliran, sehingga ketiganya maju tetapi pada setiap saat hanya satu yang berjalan. Bawah: dua inti; dua thread sungguh berjalan bersamaan, dan penjadwal memindahkan thread antarinti sesukanya.

Dua akibat mengalir dari **Gambar 14.2**, dan seluruh bab ini bertumpu padanya.

- **Bukan Anda yang menentukan urutannya.** Penjadwallah yang memutuskan kapan sebuah thread berjalan, berapa lama, dan kapan ia dijeda, dan keputusannya dapat berbeda setiap kali program dijalankan. Kode di dalam satu thread berjalan menurut urutan yang Anda tulis; kode di thread yang *berbeda* dapat berselang-seling dalam urutan apa pun.

- **Sebuah thread dapat dijeda di mana saja.** Bukan hanya di antara dua pernyataan program Anda, tetapi juga di tengah sesuatu yang tampak seperti satu pernyataan, misalnya `count++`. Subbab 14.7 memperlihatkan mengapa hal ini begitu penting.

14.2.2 Mengapa memakai thread?

Bila thread sesulit ini diramalkan, mengapa memakainya? Ada empat alasan yang baik [2].

1. **Ketanggapan.** Jendela yang menghitung laporan di satu-satunya thread miliknya akan membeku sampai laporan itu selesai; jendela yang menyerahkan laporan itu kepada thread kedua tetap menanggapi klik (Bab 10).
2. **Menunggu secara paralel.** Selagi satu thread menunggu disk, jaringan, atau mesin cuci, thread lain dapat bekerja. Sebuah server memberi setiap klien satu thread (Bab 15 melakukan persis hal itu) agar satu klien yang lambat tidak menahan klien-klien lain.
3. **Memakai setiap inti.** Laptop masa kini memiliki delapan inti atau lebih. Program dengan satu thread memakai satu di antaranya. Komputasi yang dapat dipecah menjadi bagian-bagian mandiri dapat selesai beberapa kali lebih cepat bila setiap bagian mendapat thread dan intinya sendiri.
4. **Model yang lebih sederhana.** Sebagian masalah pada dasarnya adalah beberapa kegiatan mandiri: tiga mesin cuci, lima filsuf, satu produsen dan satu konsumen. Menulis setiap kegiatan sebagai thread tersendiri, dengan perulangan sederhananya sendiri, dapat jauh lebih jelas daripada satu perulangan besar yang menyulap semuanya.

Harganya menjadi pokok bahasan sisa bab ini: ketika thread berbagi data, kita harus mengoordinasikannya, dan di koordinasi itulah *bug* bersarang.

14.3 Induk yang Tak Dapat Merebut Kembali Kendali

Contoh pertama kita adalah sebuah keluarga. Seorang induk ingin beberapa anak mencacah, masing-masing dengan kecepatannya sendiri, selama induk menghendaki, lalu berhenti. Mari mencoba menulisnya lebih dahulu tanpa thread, untuk melihat masalah yang dipecahkan thread. Seorang anak mempunyai nama dan metode `begin` yang mencacah tanpa henti:

```
public class Child {
    private String name;
    public Child(String name) { this.name = name; }
    public void begin() {
        int i = 0;
        while (true) {
            System.out.println(name + " has counted " + (i++));
        }
    }
}
```

dan sang induk menciptakan dua anak lalu meminta masing-masing memulai:

```
Child one = new Child("Luffy");
one.begin();
Child two = new Child("Ace");
two.begin();
```

Apa yang terjadi ketika program ini berjalan? Induk memanggil `one.begin()`, dan kendali berpindah ke anak pertama, yang masuk ke perulangan tak berujung. Panggilan itu tidak pernah kembali. Induk tidak pernah sampai ke baris yang menciptakan anak kedua, dan induk juga tidak punya cara untuk menghentikan anak pertama: induk sama sekali tidak sedang berjalan, karena satu-satunya thread miliknya sibuk di dalam perulangan si anak. (Program ini kita cetak sebagai

teks, bukan sebagai listing, karena ia tidak pernah berakhir; Anda boleh mengetiknya dan menghentikannya dengan Ctrl+C.)

Yang kita inginkan adalah agar `begin` memulai pencacahan *di tempat lain* dan segera mengembalikan kendali kepada induk. Itulah persisnya yang dilakukan sebuah thread.

14.3.1 Anak-anak sebagai thread

Di Java, sepotong pekerjaan yang dapat berjalan di thread-nya sendiri dideskripsikan oleh antarmuka `Runnable`, yang memiliki satu metode, `run()`. Kita tidak pernah memanggil `run()` sendiri. Sebagai gantinya kita memberikan `Runnable` itu kepada sebuah objek `Thread` dan memanggil metode `start()` milik thread tersebut. `start()` meminta Java Virtual Machine menciptakan thread baru, dan thread baru itulah yang memanggil `run()`; sementara itu `start()` sendiri langsung kembali, dan induk meneruskan pekerjaannya. **Kode 14.1** adalah program lengkapnya.

Kode 14.1 `ParentChild.java`: induk memulai dua anak yang mencacah sebagai thread, tidur tiga detik, lalu menginterupsi mereka dan menunggu mereka berakhir.

```
import java.util.Random;

public class ParentChild {
    // Seorang anak mencacah tanpa henti, tidur selama waktu acak setelah setiap bilangan.
    static class Child implements Runnable {
        private final String name;
        private final Random random;

        Child(String name, long seed) {
            this.name = name;
            this.random = new Random(seed);  // benih membuat waktu tidur dapat diulang
        }

        @Override
        public void run() {
            try {
                int i = 0;
                while (true) {
                    int rand = random.nextInt(1000);
                    System.out.println(name + " has counted to " + (i++)
                            + " and will now sleep for " + rand + "ms");
                    Thread.sleep(rand);
                }
            } catch (InterruptedException e) {
                System.out.println(name + " interrupted; ending");
            }
        }
    }

    public static void main(String[] args) throws InterruptedException {
        System.out.println("Parent started");
        System.out.println("Parent is starting Child: Luffy");
        Thread threadOne = new Thread(new Child("Luffy", 1));
        threadOne.start();  // start() langsung kembali; run() berlanjut di thread baru
        System.out.println("Parent is starting Child: Ace");
        Thread threadTwo = new Thread(new Child("Ace", 2));
        threadTwo.start();
        System.out.println("Parent will sleep for 3 seconds");
        try {
            Thread.sleep(3000);
            System.out.println("Parent has woken up");
        } finally {
            // Minta kedua anak berhenti; masing-masing memutuskan cara menanggapinya.
            threadOne.interrupt();
            threadTwo.interrupt();
        }
```

Kode 14.1 (Lanjutan) *ParentChild.java: induk memulai dua anak yang mencacah sebagai thread, tidur tiga detik, lalu menginterupsi mereka dan menunggu mereka berakhir.*

```
        threadOne.join();  // tunggu sampai setiap anak benar-benar selesai
        threadTwo.join();
        System.out.println("Parent ended");
    }
}
```

Kompilasi dan jalankan di folder `java` bab ini dengan `javac ParentChild.java` dan `java ParentChild`. Satu kali jalan yang sungguhan mencetak:

```
Parent started
Parent is starting Child: Luffy
Parent is starting Child: Ace
Parent will sleep for 3 seconds
Luffy has counted to 0 and will now sleep for 985ms
Ace has counted to 0 and will now sleep for 108ms
Ace has counted to 1 and will now sleep for 372ms
Ace has counted to 2 and will now sleep for 40ms
Ace has counted to 3 and will now sleep for 67ms
Ace has counted to 4 and will now sleep for 389ms
Ace has counted to 5 and will now sleep for 350ms
Luffy has counted to 1 and will now sleep for 588ms
Ace has counted to 6 and will now sleep for 606ms
Luffy has counted to 2 and will now sleep for 847ms
Ace has counted to 7 and will now sleep for 719ms
Luffy has counted to 3 and will now sleep for 313ms
Ace has counted to 8 and will now sleep for 847ms
Luffy has counted to 4 and will now sleep for 254ms
Parent has woken up
Ace interrupted; ending
Luffy interrupted; ending
Parent ended
```

Bacalah keluarannya perlahan, karena di dalamnya sudah terkandung inti konkurensi.

- Induk mencetak `Parent is starting Child: Ace` *segera* setelah memulai Luffy. `threadOne.start()` tidak menunggu perulangan Luffy; ia langsung kembali, persis seperti yang kita inginkan.
- Baris-baris Luffy dan Ace bercampur. Setiap anak mencetak satu baris, tidur selama waktu acak, lalu mencetak lagi, dan tidur kedua anak tidak saling bergantung, sehingga baris-baris mereka berselang-seling. Tidur-tidur pertama Ace pendek (108 ms, 372 ms, 40 ms, 67 ms), sehingga Ace sudah mencacah sampai 5 sebelum Luffy bangun dari tidur pertamanya yang 985 ms.
- Setelah tiga detik induk bangun, menginterupsi kedua anak, dan setiap anak mencetak `interrupted; ending`. Baru kemudian induk mencetak `Parent ended`, karena `join()` menunggu sampai sebuah thread selesai.

Sekarang pembacaan baris demi baris.

- `static class Child implements Runnable` mendeklarasikan pekerjaannya. *Static nested class* (kelas bersarang statis) hanyalah kelas yang ditulis di dalam kelas lain, sehingga seluruh contoh tetap berada dalam satu berkas; ia berperilaku seperti kelas tingkat atas mana pun.
- `this.random = new Random(seed)` memberi setiap anak pembangkit bilangan acaknya sendiri dengan *seed* (benih) yang tetap, sehingga waktu tidurnya (985, 588, 847, … untuk Luffy) sama pada setiap jalan dan juga di komputer Anda.
- `while (true) { … Thread.sleep(rand); }` adalah pekerjaan si anak: mencetak, lalu tidur selama `rand` milidetik. `Thread.sleep` menjeda *thread yang memanggilnya*, di sini si anak, dan membiarkan thread lain berjalan.
- `catch (InterruptedException e)` adalah tanggapan si anak ketika diinterupsi. `Thread.sleep` termasuk metode yang melempar `InterruptedException` bila thread lain memanggil `interrupt()` pada thread yang sedang tidur. Eksepsi itu melompat keluar dari perulangan, si anak mencetak salam perpisahan, `run()` kembali, dan thread berakhir.
- Di `main`, `new Thread(new Child("Luffy", 1))` membungkus pekerjaan itu dalam sebuah thread, dan `threadOne.start()` menjalankannya. Induk langsung melanjutkan.
- `Thread.sleep(3000)` menidurkan *sang induk*, yang tidak lain adalah thread `main`, selama tiga detik.

- Blok `finally` memanggil `interrupt()` pada kedua anak. Ia diletakkan di `finally` agar anak-anak tetap diminta berhenti sekalipun tidur sang induk entah bagaimana terpotong.
- `threadOne.join()` memblokir induk sampai thread Luffy berakhir. Tanpa kedua panggilan `join`, `Parent ended` bisa saja tercetak sebelum salam perpisahan anak-anak.

Apa yang *berubah-ubah* antarjalan? Bukan waktu tidurnya, yang ditetapkan oleh benih, melainkan pewaktuan halusnya: dua saat bangun yang jatuh dalam selisih satu milidetik dapat tercetak dalam urutan mana pun, seorang anak yang saat bangunnya dekat dengan 3,000 ms dapat mencetak satu baris lebih banyak atau lebih sedikit, dan kedua baris `interrupted; ending` muncul dalam urutan mana pun. Polanya sama pada setiap jalan; rinciannya tidak.

> **PERINGATAN:** Interupsi adalah *permintaan*, bukan perintah. `interrupt()` hanya memasang sebuah bendera di thread sasaran dan membangunkannya bila ia sedang tidur atau menunggu. Sepenuhnya terserah si anak apa yang akan dilakukannya: anak-anak kita berakhir, tetapi anak yang menangkap `InterruptedException` lalu begitu saja mengulang perulangannya akan mengabaikan induknya selamanya. Java tidak memiliki cara aman untuk membunuh thread dari luar; `Thread.stop()` yang lama dinyatakan usang karena thread yang dibunuh di tengah pembaruan meninggalkan objek dalam keadaan setengah berubah [3].

Kode 14.2 adalah keluarga yang sama dalam Python. Modul `threading` [4] menyediakan `threading.Thread`, yang menerima pekerjaan berupa sebuah *fungsi* (`target`), bukan objek yang mengimplementasikan antarmuka.

Kode 14.2 `parent_child.py`: induk dan anak-anak yang sama dalam Python; sebuah `threading.Event` bersama menggantikan interupsi Java.

```
"""parent_child.py: induk memulai dua anak yang mencacah, lalu menghentikan mereka."""
import random
import threading


class Child:
    """Seorang anak mencacah tanpa henti, tidur selama waktu acak setelah setiap
bilangan."""

    def __init__(self, name, seed, stop):
        self.name = name
        self.random = random.Random(seed)  # benih membuat waktu tidur dapat diulang
        self.stop = stop                   # threading.Event yang dipakai bersama induk

    def run(self):
        i = 0
        while True:
            rand = self.random.randrange(1000)
            print(f"{self.name} has counted to {i} and will now sleep for {rand}ms")
            i += 1
            # wait() tidur seperti time.sleep, tetapi bangun lebih awal bila event di-set.
            if self.stop.wait(rand / 1000):
                print(f"{self.name} interrupted; ending")
                return


def main():
    print("Parent started")
    stop = threading.Event()
    print("Parent is starting Child: Luffy")
    thread_one = threading.Thread(target=Child("Luffy", 1, stop).run)
    thread_one.start()  # start() langsung kembali; run() berlanjut di thread baru
    print("Parent is starting Child: Ace")
    thread_two = threading.Thread(target=Child("Ace", 2, stop).run)
    thread_two.start()
    print("Parent will sleep for 3 seconds")
    try:
        threading.Event().wait(3)  # sama dengan time.sleep(3)
        print("Parent has woken up")
    finally:
        stop.set()  # minta kedua anak berhenti
    thread_one.join()  # tunggu sampai setiap anak benar-benar selesai
    thread_two.join()
    print("Parent ended")
```

Kode 14.2 (Lanjutan) *parent_child.py: induk dan anak-anak yang sama dalam Python; sebuah threading.Event bersama menggantikan interupsi Java.*

```
if __name__ == "__main__":
    main()
```

Jalankan dengan `python3 parent_child.py` (di Windows, `python parent_child.py`):

```
Parent started
Parent is starting Child: Luffy
Parent is starting Child: Ace
Luffy has counted to 0 and will now sleep for 137ms
Ace has counted to 0 and will now sleep for 978ms
Parent will sleep for 3 seconds
Luffy has counted to 1 and will now sleep for 582ms
Luffy has counted to 2 and will now sleep for 867ms
Ace has counted to 1 and will now sleep for 883ms
Luffy has counted to 3 and will now sleep for 821ms
Ace has counted to 2 and will now sleep for 970ms
Luffy has counted to 4 and will now sleep for 782ms
Ace has counted to 3 and will now sleep for 869ms
Parent has woken up
Luffy interrupted; ending
Ace interrupted; ending
Parent ended
```

Strukturnya sama, dan keluarannya memperlihatkan selang-seling yang sama. Waktu tidurnya berbeda dari Java karena pembangkit bilangan acak Python memakai algoritme lain, tetapi waktu-waktu itu pun ditetapkan oleh benihnya.

Satu-satunya perbedaan sungguhan adalah cara induk meminta anak-anaknya berhenti. **Thread Python tidak dapat diinterupsi.** Tidak ada metode `interrupt()`, dan thread yang sedang berada di dalam `time.sleep(5)` akan tidur penuh lima detik, apa pun yang dilakukan thread lain. Idiom Python adalah `threading.Event`: objek yang memegang satu bendera Boolean, dengan metode `set()` yang menaikkan bendera dan metode `wait(timeout)` yang tidur *sampai bendera dinaikkan atau batas waktu lewat*, mana yang lebih dahulu, dan mengembalikan `True` bila benderanya dinaikkan. Anak-anak memanggil `self.stop.wait(rand / 1000)` alih-alih `time.sleep`, sehingga mereka tidur seperti sebelumnya tetapi langsung bangun ketika induk memanggil `stop.set()`. Jeda tiga detik milik induk sendiri, `threading.Event().wait(3)`, hanyalah tidur pada sebuah event baru yang tidak pernah di-set siapa pun.

Perhatikan juga `target=Child("Luffy", 1, stop).run`: kita memberikan *bound method* `run` milik objek anak, tanpa tanda kurung. Menulis `target=child.run()` akan *memanggil* `run` di thread induk, yang mencacah tanpa henti, sehingga masalah semula kembali. Subbab 14.4 membahas lagi kesalahan ini dalam bentuk Java-nya.

14.3.2 Berhenti dengan bendera, dan jebakan visibilitas

Kadang anak harus berhenti bukan karena diinterupsi, melainkan karena *disuruh*, melalui bendera yang diperiksanya di setiap putaran perulangannya. Cara yang paling jelas adalah field Boolean yang di-set `true` oleh induk dan diuji oleh anak. Di sinilah Java menyembunyikan salah satu jebakannya yang paling tidak intuitif. **Kode 14.3** menguji cara yang jelas itu terhadap cara yang benar.

Kode 14.3 `StopFlag.java`: thread yang berputar pada bendera `boolean` biasa mungkin tidak pernah melihat bendera itu berubah; perulangan yang sama pada bendera `volatile` langsung berhenti.

```
public class StopFlag {
    static boolean plainStop = false;            // tidak ada jaminan thread lain melihat
perubahan
    static volatile boolean volatileStop = false; // setiap penulisan terlihat oleh setiap
pembaca

    public static void main(String[] args) throws InterruptedException {
        Thread plain = new Thread(() -> {
            long spins = 0;
            while (!plainStop) {
                spins++;
            }
            System.out.println("plain spinner saw the flag after " + spins + " spins");
        });
        plain.setDaemon(true);  // thread daemon tidak membuat program tetap hidup
        plain.start();
        Thread.sleep(500);
        plainStop = true;
        plain.join(1000);       // tunggu paling lama satu detik
        System.out.println("plain flag: still running 1 s later? " + plain.isAlive());

        Thread careful = new Thread(() -> {
            long spins = 0;
            while (!volatileStop) {
                spins++;
            }
            System.out.println("volatile spinner stopped");
        });
        careful.start();
        Thread.sleep(500);
        volatileStop = true;
        careful.join(1000);
        System.out.println("volatile flag: still running 1 s later? " + careful.isAlive());
    }
}
```

```
plain flag: still running 1 s later? true
volatile spinner stopped
volatile flag: still running 1 s later? false
```

Pemutar pertama *tidak pernah berhenti*. Thread utama sudah memasang `plainStop = true`, menunggu satu detik penuh, dan si pemutar masih berjalan; program ini berakhir hanya karena thread itu kita jadikan *daemon* (Subbab 14.4 menjelaskan daemon). Pemutar kedua, yang identik kecuali benderanya dideklarasikan `volatile`, langsung berhenti.

Bagaimana mungkin sebuah thread tidak melihat perubahan pada suatu variabel? Aturan Java tidak menjanjikan bahwa operasi tulis oleh satu thread pernah terlihat oleh thread lain, kecuali kedua thread itu *bersinkronisasi* dengan salah satu dari beberapa cara yang telah ditentukan. Di sini perulangan yang berputar membaca `plainStop` jutaan kali dan tidak pernah menulisnya, sehingga kompiler *just-in-time* Java boleh membaca field itu sekali, menyimpan nilainya di register prosesor, dan mengubah perulangan itu menjadi `while (true)`. Itulah persisnya yang terjadi. Kata kunci **`volatile`** melarang optimasi ini: setiap pembacaan field volatile pergi ke memori dan melihat operasi tulis terakhir oleh thread mana pun [3], [5]. Inilah *memory model* (model memori) Java, dan kita kembali kepadanya di Subbab 14.8.

> **DEFINISI.** Masalah **visibilitas** terjadi ketika satu thread mengubah variabel bersama dan thread lain tidak melihat perubahan itu, atau melihatnya terlambat. Di Java, sebuah operasi tulis dijamin terlihat oleh thread lain hanya bila keduanya dihubungkan oleh relasi *happens-before* (terjadi-sebelum): misalnya, operasi tulis itu dilakukan pada field `volatile` dan thread

> lain membaca field tersebut, atau kedua thread memakai kunci yang sama, atau satu thread memulai atau menunggu (*join*) thread yang lain.

Kode 14.4 mencoba percobaan yang sama di Python.

Kode 14.4 `stop_flag.py`: di CPython, atribut biasa yang dipakai sebagai bendera berhenti memang menjadi terlihat; `threading.Event` tetap pilihan yang idiomatis.

```
"""stop_flag.py: menghentikan thread yang sibuk dengan atribut biasa dan dengan Event."""
import threading
import time


class Spinner:
    def __init__(self):
        self.stop = False  # atribut biasa, dibaca di setiap putaran

    def run(self):
        spins = 0
        while not self.stop:
            spins += 1
        print("plain spinner saw the flag")


def main():
    s = Spinner()
    t = threading.Thread(target=s.run, daemon=True)
    t.start()
    time.sleep(0.5)
    s.stop = True
    t.join(1)
    print("plain flag: still running 1 s later?", t.is_alive())

    done = threading.Event()  # sinyal berhenti yang idiomatis

    def careful():
        spins = 0
        while not done.is_set():
            spins += 1
        print("event spinner stopped")

    t2 = threading.Thread(target=careful)
    t2.start()
    time.sleep(0.5)
    done.set()
    t2.join(1)
    print("event: still running 1 s later?", t2.is_alive())


if __name__ == "__main__":
    main()
```

```
plain spinner saw the flag
plain flag: still running 1 s later? False
event spinner stopped
event: still running 1 s later? False
```

Kedua pemutar berhenti. Di CPython, interpreter Python yang standar, setiap pembacaan atribut sungguh-sungguh mencari nilai terkininya, dan Global Interpreter Lock (Subbab 14.6) membuat operasi tulis satu thread terlihat oleh thread lain, sehingga jebakan visibilitas **Kode 14.3** tidak muncul. Itu fakta tentang CPython masa kini, bukan janji bahasanya, dan perulangan yang berputar memboroskan satu inti penuh selama menunggu. Sinyal berhenti yang idiomatis di Python adalah `threading.Event`: `done.set()` untuk berhenti, `done.is_set()` untuk memeriksa,

dan `done.wait(timeout)` untuk tidur sampai dihentikan, seperti yang dilakukan `parent_child.py`. Di Java, pakailah `volatile` untuk bendera sederhana, atau pakailah interupsi.

14.4 Empat Cara Menciptakan Thread

Ada lebih dari satu cara menyerahkan pekerjaan kepada sebuah thread, dan masing-masing punya tempatnya. **Kode 14.5** memperlihatkan empat cara di Java, dengan pekerjaan-pekerjaan binatu sebagai contohnya.

Kode 14.5 `CreateThreads.java`: subkelas `Thread`, sebuah `Runnable`, sebuah lambda, dan `ExecutorService` yang mengembalikan hasil melalui objek `Future`; serta apa yang terjadi bila Anda memanggil `run()` alih-alih `start()`.

```
import java.util.ArrayList;
import java.util.List;
import java.util.concurrent.Callable;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.Future;
import java.util.concurrent.TimeUnit;

public class CreateThreads {
    // Cara 1: perluas Thread dan override run().
    static class Washer extends Thread {
        Washer(String name) {
            super(name);
        }

        @Override
        public void run() {
            System.out.println(getName() + ": washing (way 1, a subclass of Thread)");
        }
    }

    // Cara 2: implementasikan Runnable dan serahkan objeknya kepada sebuah Thread.
    static class Dryer implements Runnable {
        @Override
        public void run() {
            String me = Thread.currentThread().getName();
            System.out.println(me + ": drying (way 2, a Runnable)");
        }
    }

    public static void main(String[] args) throws Exception {
        Thread t1 = new Washer("washer-1");
        Thread t2 = new Thread(new Dryer(), "dryer-1");
        // Cara 3: lambda juga sebuah Runnable.
        Thread t3 = new Thread(() -> System.out.println(
                Thread.currentThread().getName() + ": ironing (way 3, a lambda)"), "iron-
1");
        t1.start();
        t1.join();
        t2.start();
        t2.join();
        t3.start();
        t3.join();
```

Kode 14.5 (Lanjutan) *CreateThreads.java: subkelas Thread, sebuah Runnable, sebuah lambda, dan ExecutorService yang mengembalikan hasil melalui objek Future; serta apa yang terjadi bila Anda memanggil run() alih-alih start().*

```
        // Cara 4: ExecutorService menjalankan tugas di kolam thread yang dapat dipakai
ulang.
        ExecutorService pool = Executors.newFixedThreadPool(2);
        int[] loadsKg = {5, 7, 3, 6};
        List<Future<Integer>> prices = new ArrayList<>();
        for (int kg : loadsKg) {
            Callable<Integer> job = () -> kg * 7000;  // Rp7,000 per kilogram
            prices.add(pool.submit(job));
        }
        int total = 0;
        for (Future<Integer> price : prices) {
            total += price.get();  // get() menunggu sampai tugas itu selesai
        }
        pool.shutdown();           // jangan terima tugas baru; selesaikan yang lama
        System.out.println("pool finished: " + pool.awaitTermination(1, TimeUnit.SECONDS));
        System.out.println("total of 4 loads = Rp" + total);

        // Kesalahan klasik: run() alih-alih start() tidak menciptakan thread baru.
        Thread notStarted = new Thread(() -> System.out.println(
                "run() executed by: " + Thread.currentThread().getName()), "helper");
        notStarted.run();
        notStarted.start();
        notStarted.join();
    }
}
```

```
washer-1: washing (way 1, a subclass of Thread)
dryer-1: drying (way 2, a Runnable)
iron-1: ironing (way 3, a lambda)
pool finished: true
total of 4 loads = Rp147000
run() executed by: main
run() executed by: helper
```

Keluaran ini sama pada setiap jalan, karena setelah setiap `start()` program segera memanggil `join()`, sehingga setiap saat hanya ada satu thread tambahan yang hidup. Mari telusuri keempat cara itu.

Cara 1: memperluas `Thread`. `Washer extends Thread` dan meng-*override* `run()`. Sebuah `Washer` *adalah* thread, sehingga `new Washer("washer-1")` diikuti `start()` sudah cukup. Konstruktornya meneruskan sebuah nama ke `super(name)`, dan `getName()` mengembalikannya. Inilah gaya tertua dan paling kaku: kelas yang memperluas `Thread` tidak dapat memperluas kelas lain (Bab 7), dan ia mencampur dua gagasan, *apa* yang dikerjakan dan *bagaimana* menjalankannya.

Cara 2: mengimplementasikan `Runnable`. `Dryer implements Runnable` hanya mendeskripsikan *apa* yang dikerjakan. Objek `Thread` yang terpisah menentukan *bagaimana* menjalankannya: `new Thread(new Dryer(), "dryer-1")`. Di dalam `run()`, `Thread.currentThread()` mengembalikan thread yang sedang mengeksekusi kode ini, sehingga `Thread.currentThread().getName()` mencetak `dryer-1`. Memisahkan pekerjaan dari thread adalah rancangan yang lebih baik, dan itulah yang dianjurkan dokumentasi Java [6].

Cara 3: sebuah lambda. `Runnable` memiliki tepat satu metode abstrak, sehingga ia adalah *functional interface*, dan ekspresi lambda `() -> …` dapat mewakilinya. Untuk pekerjaan pendek, inilah bentuk yang paling ringkas.

Cara 4: sebuah `ExecutorService`. Menciptakan thread memakan waktu dan memori, dan program yang menciptakan satu thread per tugas segera memiliki ribuan thread.

`ExecutorService` memelihara sebuah *pool* (kolam) thread pekerja dan menyuapkan tugas kepada mereka. `Executors.newFixedThreadPool(2)` membuat kolam berisi dua thread. `pool.submit(job)` menyerahkan sebuah tugas dan mengembalikan sebuah `Future`, semacam tiket untuk hasil yang mungkin belum ada. Tugas di sini adalah `Callable<Integer>`, yang mirip `Runnable` kecuali bahwa ia mengembalikan nilai (harga satu muatan, Rp7,000 per kilogram). `price.get()` menunggu sampai tugas itu selesai dan mengembalikan nilainya. Empat muatan seberat 5, 7, 3, dan 6 kg berharga total $21 \times 7000 = 147{,}000$ rupiah, sesuai yang tercetak. Terakhir, `pool.shutdown()` memberi tahu kolam agar tidak menerima tugas baru dan membiarkan thread-threadnya berakhir setelah tugas yang mengantre selesai, dan `awaitTermination` menunggu hal itu, lalu mengembalikan `true`.

Blok terakhir memperlihatkan kesalahan pemula yang paling umum. `notStarted.run()` adalah pemanggilan metode biasa: ia mengeksekusi lambda itu *di thread saat ini*, yaitu `main`, dan tidak ada thread baru yang tercipta. `start()` sesudahnya barulah menciptakan thread bernama `helper`, dan lambda yang sama kemudian melaporkan bahwa ia berjalan di sana. Program yang memanggil `run()` alih-alih `start()` biasanya tetap memberi jawaban yang benar, hanya saja lambat dan tanpa konkurensi apa pun, dan itulah sebabnya kesalahan ini bertahan begitu lama.

> **PERINGATAN:** Sebuah thread hanya dapat dimulai sekali. Memanggil `start()` untuk kedua kalinya pada objek `Thread` yang sama melempar `IllegalThreadStateException`, bahkan setelah jalan pertamanya selesai. Untuk menjalankan pekerjaan yang sama lagi, ciptakan `Thread` baru, atau serahkan tugas itu ke kolam sekali lagi.

Java 21 dan sesudahnya juga menawarkan **virtual thread** (`Thread.ofVirtual().start(task)`), yang dikelola oleh *runtime* Java, memakan sangat sedikit memori, dan membuat satu thread per tugas terjangkau bahkan untuk sejuta tugas. Race, kunci, dan deadlock bekerja persis sama dengannya; kita memakai thread biasa agar listing-listingnya juga berjalan di Java 17.

Kode 14.6 adalah padanannya di Python.

Kode 14.6 `create_threads.py`: subkelas `threading.Thread`, fungsi target, sebuah lambda, dan `ThreadPoolExecutor` dengan `submit` serta `map`.

```
"""create_threads.py: cara-cara menciptakan thread di Python."""
import threading
from concurrent.futures import ThreadPoolExecutor


class Washer(threading.Thread):
    """Cara 1: buat subkelas Thread dan override run()."""

    def __init__(self, name):
        super().__init__(name=name)

    def run(self):
        print(f"{self.name}: washing (way 1, a subclass of Thread)")


def dry():
    """Cara 2: fungsi apa pun dapat menjadi target sebuah Thread."""
    print(f"{threading.current_thread().name}: drying (way 2, a target function)")


def main():
    t1 = Washer("washer-1")
    t2 = threading.Thread(target=dry, name="dryer-1")
    # Cara 3: lambda juga sebuah fungsi.
    t3 = threading.Thread(
        target=lambda: print(f"{threading.current_thread().name}: ironing (way 3, a
lambda)"),
        name="iron-1")
    for t in (t1, t2, t3):
        t.start()
        t.join()

    # Cara 4: ThreadPoolExecutor menjalankan tugas di kolam thread yang dapat dipakai
ulang.
    loads_kg = [5, 7, 3, 6]
    with ThreadPoolExecutor(max_workers=2) as pool:
        prices = [pool.submit(lambda kg=kg: kg * 7000) for kg in loads_kg]  # Rp7,000 per
kg
        total = sum(p.result() for p in prices)  # result() menunggu tugas itu
        also = sum(pool.map(lambda kg: kg * 7000, loads_kg))
    # Keluar dari blok with mematikan kolam dan menunggu thread-threadnya.
    print(f"total of 4 loads = Rp{total} (map gives Rp{also})")

    # Kesalahan klasik: run() alih-alih start() tidak menciptakan thread baru.
    def who():
        print("run() executed by:", threading.current_thread().name)
```

Kode 14.6 (Lanjutan) *create_threads.py: subkelas threading.Thread, fungsi target, sebuah lambda, dan ThreadPoolExecutor dengan submit serta map.*

```
    threading.Thread(target=who, name="helper-1").run()  # salah: berjalan di main
    right = threading.Thread(target=who, name="helper-2")
    right.start()                                        # benar: berjalan di helper-2
    right.join()


if __name__ == "__main__":
    main()
```

```
washer-1: washing (way 1, a subclass of Thread)
dryer-1: drying (way 2, a target function)
iron-1: ironing (way 3, a lambda)
total of 4 loads = Rp147000 (map gives Rp147000)
run() executed by: MainThread
run() executed by: helper-2
```

Cara-caranya berpadanan satu lawan satu. Subkelas `threading.Thread` meng-*override* `run()` dan meneruskan namanya ke `super().__init__(name=name)`; fungsi target adalah `Runnable`-nya Python; lambda dapat menjadi target yang singkat. `ThreadPoolExecutor` dari `concurrent.futures` [4] adalah *executor* Python, dan `submit`-nya juga mengembalikan `Future` yang metode `result()`-nya menunggu nilainya. `pool.map(fn, items)` adalah jalan pintas yang menerapkan `fn` pada setiap butir di thread-thread kolam dan menghasilkan hasilnya menurut urutan semula. Memakai *executor* di dalam pernyataan `with` akan mematikannya dan menunggu thread-threadnya ketika blok itu berakhir, pekerjaan yang sama dengan `shutdown` ditambah `awaitTermination` di Java.

Ada satu rincian yang patut dikomentari: `lambda kg=kg: kg * 7000`. Lambda yang didefinisikan di dalam perulangan menangkap *variabel* `kg`, bukan nilainya pada saat itu, dan ketika kolam menjalankan lambda-lambda itu perulangannya mungkin sudah bergerak, sehingga setiap lambda akan melihat nilai terakhir. Menulis `kg=kg` mengubah nilai saat ini menjadi argumen bawaan, yang dievaluasi saat itu juga. Java menghindari masalah ini dengan hanya mengizinkan lambda menangkap variabel yang *effectively final*, sehingga perulangan Java dapat langsung menulis `() -> kg * 7000`.

Kesalahan `run()` juga ada di Python, dengan satu keanehan: setelah `Thread.run()` mengeksekusi targetnya, ia menghapus target itu, sehingga memanggil `start()` sesudahnya pada objek yang sama akan gagal. Karena itu listing ini memakai dua objek thread yang terpisah, `helper-1` untuk cara yang salah dan `helper-2` untuk cara yang benar.

Tabel 14.1 merangkum kosakata sejauh ini, berdampingan.

Tabel 14.1 Kosakata dasar thread di Java dan Python.

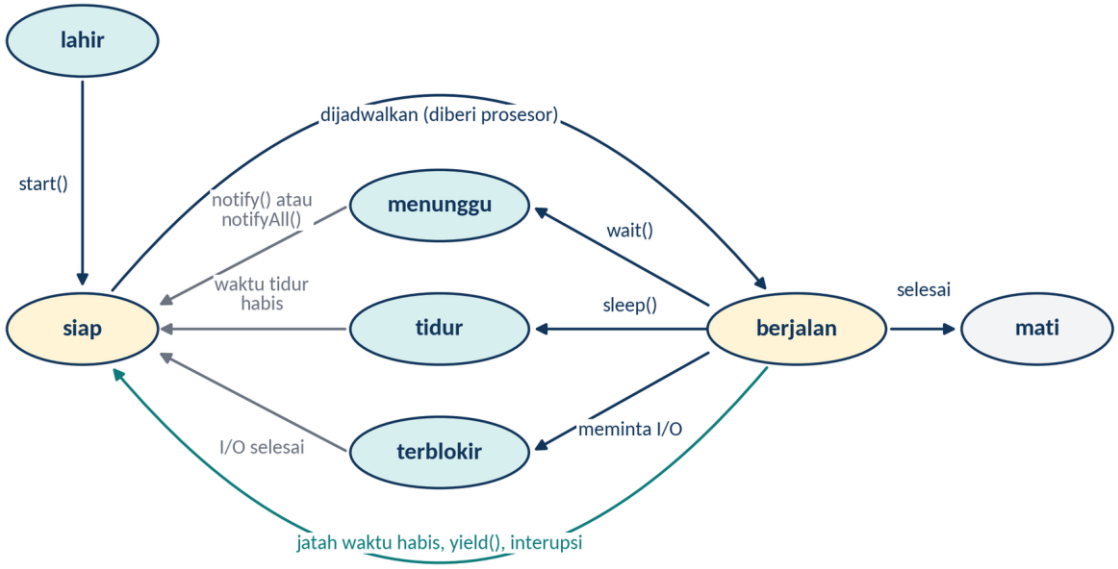
Tugas	Java	Python
mendeskripsikan pekerjaan	`Runnable` (`run()`), `Callable<T>` (`call()`)	sebuah fungsi (`target`)
sebuah thread	`new Thread(runnable, name)`	`threading.Thread(target=f, name=n)`
memulainya	`t.start()`	`t.start()`
menunggunya berakhir	`t.join()`, `t.join(ms)`	`t.join()`, `t.join(seconds)`
masih berjalan?	`t.isAlive()`, `t.getState()`	`t.is_alive()`
thread saat ini	`Thread.currentThread()`	`threading.current_thread()`
menjeda thread saat ini	`Thread.sleep(ms)`	`time.sleep(seconds)`
memintanya berhenti	`t.interrupt()`	set `threading.Event` yang diperiksanya
thread latar belakang	`t.setDaemon(true)` sebelum `start()`	`Thread(..., daemon=True)`
kolam thread	`ExecutorService`, `Executors.newFixedThreadPool(n)`	`concurrent.futures.ThreadPoolExecutor(n)`
hasil yang datang kemudian	`Future<T>`, `get()`	`Future`, `result()`

Baris tentang thread *daemon* perlu sepatah kata. Program Java atau Python berakhir ketika thread **non-daemon** terakhirnya berakhir. Thread **daemon** adalah pembantu latar belakang yang tidak membuat program tetap hidup: ketika yang tersisa hanya thread daemon, program

keluar dan thread-thread itu dihentikan di mana pun mereka berada. Karena itu daemon cocok untuk layanan latar belakang yang boleh ditinggalkan (tampilan jam, pembersih *cache*) dan tidak cocok untuk pekerjaan yang harus tuntas (menulis berkas). Kita memakai daemon di bab ini karena alasan lain: thread yang kita ketahui macet selamanya, dalam sebuah deadlock, tidak boleh menghalangi program peraga untuk berakhir.

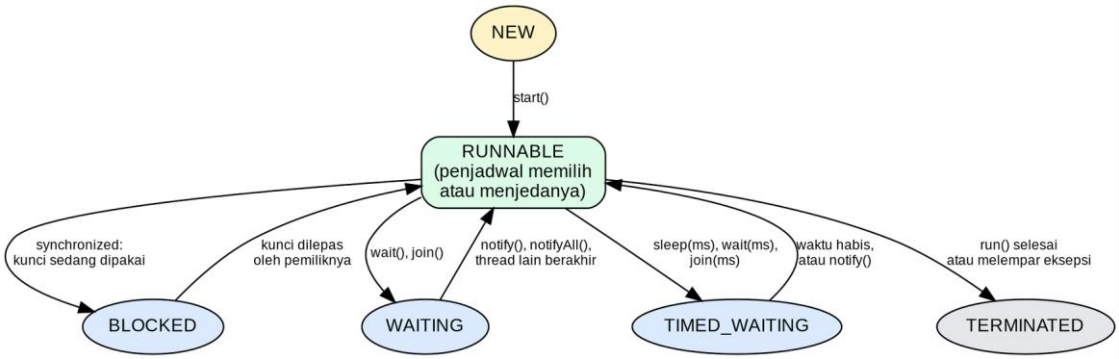
14.5 Kehidupan Sebuah Thread

Sebuah thread lahir, berjalan, menunggu dengan beberapa cara yang berbeda, lalu mati. **Gambar 14.3** adalah gambar klasik kehidupan itu, digambar ulang mengikuti versi yang dijumpai di banyak buku teks Java [7].



Gambar 14.3 Siklus hidup klasik sebuah thread Java. Thread lahir, menjadi siap ketika dimulai, berjalan ketika penjadwal memberinya prosesor, dan dapat meninggalkan keadaan berjalan untuk menunggu, tidur, atau terblokir oleh masukan dan keluaran sebelum siap lagi atau mati.

Gambar itu memisahkan *siap* (dapat berjalan, menunggu inti) dari *berjalan* (sedang di inti). Java sendiri tidak memperlihatkan perbedaan itu, karena penjadwal berpindah-pindah di antara keduanya ribuan kali per detik. Yang diperlihatkan Java adalah enumerasi `Thread.State`, dengan tepat enam nilai, dan **Gambar 14.4** menggambar ulang siklus hidup itu dalam istilah-istilah tersebut.



Gambar 14.4 Enam nilai `Thread.State` dan peristiwa yang memindahkan thread di antaranya. `RUNNABLE` mencakup "siap" dan "berjalan" pada **Gambar 14.3**; `BLOCKED` berarti menunggu kunci, `WAITING` berarti menunggu tanpa batas waktu, dan `TIMED_WAITING` berarti menunggu dengan batas waktu.

- `NEW`: objek `Thread` sudah ada tetapi `start()` belum dipanggil.
- `RUNNABLE`: sudah dimulai, dan sedang berjalan atau siap berjalan. Thread yang sedang menghitung berstatus `RUNNABLE`.
- `BLOCKED`: sedang mencoba memasuki blok atau metode `synchronized` yang kuncinya dipegang thread lain (Subbab 14.8).
- `WAITING`: menunggu, tanpa batas waktu, sampai thread lain melakukan sesuatu, misalnya di dalam `wait()` (Subbab 14.9) atau `join()`.
- `TIMED_WAITING`: sama, tetapi dengan batas waktu: `sleep(ms)`, `wait(ms)`, `join(ms)`.
- `TERMINATED`: `run()` sudah kembali atau melempar eksepsi. Thread yang sudah berakhir tidak dapat dimulai ulang.

Kode 14.7 menempatkan sebuah thread di setiap keadaan secara bergiliran dan menanyakan di mana ia berada.

Kode 14.7 `ThreadStates.java`: satu thread dalam setiap keadaan dari keenam keadaan `Thread.State`, diamati dengan `getState()`.

```
public class ThreadStates {
    public static void main(String[] args) throws InterruptedException {
        Thread sleeper = new Thread(() -> {
            try {
                Thread.sleep(300);
            } catch (InterruptedException e) {
                return;
            }
        }, "sleeper");
        System.out.println("created, not started : " + sleeper.getState());
        sleeper.start();
        Thread.sleep(100);
        System.out.println("inside sleep(300)    : " + sleeper.getState());
        sleeper.join();
        System.out.println("run() has returned   : " + sleeper.getState());

        Thread busy = new Thread(() -> {
            long n = 0;
            while (!Thread.currentThread().isInterrupted()) {
                n++;  // komputasi murni: thread selalu menginginkan CPU
            }
        }, "busy");
        busy.start();
        Thread.sleep(100);
        System.out.println("computing            : " + busy.getState());
        busy.interrupt();
        busy.join();

        Object lock = new Object();
        Thread waiter = new Thread(() -> {
            synchronized (lock) {
                try {
                    lock.wait();  // mengembalikan kunci dan menunggu notify
                } catch (InterruptedException e) {
                    return;
                }
            }
        }, "waiter");
        waiter.start();
        Thread.sleep(100);
        System.out.println("inside wait()        : " + waiter.getState());
```

Kode 14.7 (Lanjutan) *ThreadStates.java: satu thread dalam setiap keadaan dari keenam keadaan Thread.State, diamati dengan getState().*

```
        Thread blocked = new Thread(() -> {
            synchronized (lock) {
                System.out.println("blocked thread finally got the lock");
            }
        }, "blocked");
        synchronized (lock) {  // main memegang kunci, jadi "blocked" tidak dapat masuk
            blocked.start();
            Thread.sleep(100);
            System.out.println("waiting for a lock   : " + blocked.getState());
            lock.notifyAll();
        }
        waiter.join();
        blocked.join();
        System.out.println("both finished        : " + waiter.getState() + " " +
blocked.getState());
    }
}
```

```
created, not started : NEW
inside sleep(300)    : TIMED_WAITING
run() has returned   : TERMINATED
computing            : RUNNABLE
inside wait()        : WAITING
waiting for a lock   : BLOCKED
blocked thread finally got the lock
both finished        : TERMINATED TERMINATED
```

Setiap pengamatan diatur dengan `Thread.sleep(100)` singkat di `main`, yang memberi waktu kepada thread lain untuk mencapai keadaan yang ingin kita lihat. (Tepatnya, ini adalah asumsi pewaktuan, jenis asumsi yang diperingatkan bab ini; pada komputer yang sangat sibuk sebuah thread mungkin belum mencapai baris yang kita harapkan setelah 100 ms. Untuk peragaan, cara ini andal, dan tidak ada yang *rusak* bila thread itu terlambat: program hanya akan mencetak keadaan yang lain.) Thread `busy` menghitung di dalam perulangan dan berstatus `RUNNABLE`; syarat perulangannya, `!Thread.currentThread().isInterrupted()`, memungkinkan `busy.interrupt()` menghentikannya walaupun ia tidak pernah tidur. `waiter` memanggil `lock.wait()` dan berstatus `WAITING`. Untuk `BLOCKED`, `main` sendiri mengambil kunci dengan `synchronized (lock)` lalu memulai thread yang mencoba mengambil kunci yang sama; thread itu tidak dapat masuk dan berstatus `BLOCKED` sampai `main` meninggalkan bloknya. `lock.notifyAll()` membangunkan si penunggu, lalu kedua thread selesai.

Python tidak menyediakan enumerasi keadaan. Yang ditawarkannya adalah `is_alive()` (benar di antara `start()` dan akhir `run()`), `ident` (pengenal yang diberikan ketika thread dimulai, `None` sebelumnya), `threading.enumerate()` (daftar thread yang hidup), dan, untuk sebuah kunci, `locked()`. **Kode 14.8** mengamati saat-saat yang sama dengan alat-alat itu.

Kode 14.8 `thread_states.py`: apa yang dapat kita amati di Python tentang kehidupan sebuah thread: `is_alive()`, `ident`, `Lock.locked()`, dan `threading.enumerate()`.

```
"""thread_states.py: apa yang dapat kita amati di Python tentang kehidupan sebuah
thread."""
import threading
import time


def main():
    sleeper = threading.Thread(target=time.sleep, args=(0.3,), name="sleeper")
    print("created, not started : alive =", sleeper.is_alive(), " ident =", sleeper.ident)
    sleeper.start()
    time.sleep(0.1)
    print("inside sleep(0.3)    : alive =", sleeper.is_alive())
    sleeper.join()
    print("run() has returned   : alive =", sleeper.is_alive())

    cond = threading.Condition()

    def wait_for_notify():
        with cond:
            cond.wait()  # mengembalikan kunci dan menunggu notify

    waiter = threading.Thread(target=wait_for_notify, name="waiter")
    waiter.start()
    time.sleep(0.1)
    print("inside wait()        : alive =", waiter.is_alive())

    lock = threading.Lock()

    def enter():
        with lock:
            print("blocked thread finally got the lock")

    blocked = threading.Thread(target=enter, name="blocked")
    with lock:  # main memegang kunci, jadi "blocked" tidak dapat masuk
        blocked.start()
        time.sleep(0.1)
        print("waiting for a lock   : alive =", blocked.is_alive(), " lock held =",
lock.locked())
        with cond:
            cond.notify_all()
    waiter.join()
    blocked.join()
    print("both finished        : alive =", waiter.is_alive(), blocked.is_alive())
    print("threads left:", [t.name for t in threading.enumerate()])


if __name__ == "__main__":
    main()
```

```
created, not started : alive = False  ident = None
inside sleep(0.3)    : alive = True
run() has returned   : alive = False
inside wait()        : alive = True
waiting for a lock   : alive = True  lock held = True
blocked thread finally got the lock
both finished        : alive = False False
threads left: ['MainThread']
```

Thread Python tidak hidup sebelum `start()`, hidup selama ia tidur, menunggu, atau terblokir oleh kunci, dan tidak hidup lagi setelah `run()` kembali. Pada akhirnya hanya `MainThread` yang tersisa.

14.6 Global Interpreter Lock Python, dan Kapan Thread Membantu

Sebelum kita mencari-cari masalah, mari lihat apakah thread menepati janji kecepatannya. **Kode 14.9** mencacah bilangan prima di bawah tiga juta, pertama dalam satu thread, lalu dipecah di antara dua thread, satu untuk setiap separuh rentang.

Kode 14.9 `SpeedupDemo.java`: mencacah bilangan prima di bawah tiga juta dalam satu thread dan dalam dua thread, beserta waktunya.

```
public class SpeedupDemo {
    // Mencacah bilangan prima di [from, to) dengan pembagian coba: kerja CPU murni.
    static int countPrimes(int from, int to) {
        int count = 0;
        for (int n = Math.max(from, 2); n < to; n++) {
            boolean prime = true;
            for (int d = 2; (long) d * d <= n; d++) {
                if (n % d == 0) {
                    prime = false;
                    break;
                }
            }
            if (prime) {
                count++;
            }
        }
        return count;
    }

    public static void main(String[] args) throws InterruptedException {
        final int n = 3_000_000;
        countPrimes(0, n / 10);  // pemanasan, agar kedua pengukuran memakai kode
terkompilasi

        long t0 = System.nanoTime();
        int sequential = countPrimes(0, n);
        long t1 = System.nanoTime();

        int[] part = new int[2];
        Thread lower = new Thread(() -> part[0] = countPrimes(0, n / 2));
        Thread upper = new Thread(() -> part[1] = countPrimes(n / 2, n));
        long t2 = System.nanoTime();
        lower.start();
        upper.start();
        lower.join();  // join() juga menjamin kita melihat part[0]
        upper.join();
        long t3 = System.nanoTime();

        System.out.println("processors available : " +
Runtime.getRuntime().availableProcessors());
        System.out.println("primes below " + n + " : " + sequential + " and " + (part[0] +
part[1]));
        System.out.printf("one thread           : %4d ms%n", (t1 - t0) / 1_000_000);
        System.out.printf("two threads          : %4d ms%n", (t3 - t2) / 1_000_000);
    }
}
```

```
processors available : 2
primes below 3000000 : 216816 and 216816
one thread           :  623 ms
two threads          :  602 ms
```

Kedua versi menemukan 216,816 bilangan prima, sebagaimana seharusnya. Waktunya, bagaimanapun, adalah pelajaran tentang kejujuran. Pada komputer dengan dua inti yang bebas,

versi dua thread biasanya memerlukan sedikit lebih dari separuh waktu versi satu thread; tidak tepat separuh, karena separuh atas rentangnya berisi bilangan-bilangan yang lebih besar, yang lebih lama diuji, sehingga kedua thread mengerjakan jumlah pekerjaan yang tidak sama. Jalan yang tercetak di atas dibuat di mesin pembangun bersama yang kedua prosesornya juga sibuk dengan pekerjaan lain, dan di sana kedua thread nyaris tidak memperoleh apa-apa: 602 ms lawan 623 ms. Waktu program konkuren bergantung pada mesinnya dan pada apa lagi yang sedang dikerjakan mesin itu, dan Anda sebaiknya mengharapkan angka yang berbeda di komputer Anda sendiri. Yang tidak berubah adalah hasilnya, 216,816.

Seberapa cepat thread *dapat* membuat sebuah program? Misalkan sebagian f dari waktu jalan dapat dibagi di antara p inti dan sisanya, $1 - f$, tidak dapat. Maka *speedup* (percepatan) terbaik yang mungkin, yaitu waktu lama dibagi waktu baru, diberikan oleh **hukum Amdahl** [8]:

$$S(p) = \frac{1}{(1 - f) + \frac{f}{p}} \tag{14.1}$$

Misalnya, bila 90% pekerjaan dapat diparalelkan ($f = 0.9$) dan kita punya $p = 4$ inti, maka $S(4) = 1/(0.1 + 0.225) = 1/0.325 \approx 3.08$: empat inti memberi kira-kira tiga kali kecepatan, bukan empat. Dan berapa pun inti yang kita tambahkan, Persamaan (14.1) tidak pernah dapat melampaui $1/(1 - f) = 10$. Bagian yang berjalan satu thread demi satu thread akhirnya mendominasi. Ingatlah ini di Subbab 14.8: setiap kunci menciptakan sebagian kecil kode yang berjalan satu thread demi satu thread.

14.6.1 Global Interpreter Lock

Sekarang percobaan yang sama di Python. **Kode 14.10** mencacah bilangan prima di bawah 600,000 dalam satu thread, dalam dua thread, dan dalam dua *proses*, lalu melakukan hal yang sama sekali lain: menunggu empat kali selama setengah detik, pertama dalam satu thread dan kemudian dalam empat thread.

Kode 14.10 `gil_demo.py`: pekerjaan yang terikat CPU dalam satu thread, dua thread, dan dua proses, serta penantian dalam satu thread dan empat thread.

```
"""gil_demo.py: kapan thread Python membantu (menunggu) dan kapan tidak (menghitung)."""
import sys
import threading
import time
from concurrent.futures import ProcessPoolExecutor, ThreadPoolExecutor


def count_primes(bounds):
    """Mencacah bilangan prima di [lo, hi) dengan pembagian coba: kerja CPU murni."""
    lo, hi = bounds
    count = 0
    for n in range(max(lo, 2), hi):
        d = 2
        while d * d <= n:
            if n % d == 0:
                break
            d += 1
        else:
            count += 1
    return count


def wait_for_device(seconds):
    """Mewakili penantian terhadap disk, jaringan, atau mesin cuci."""
    time.sleep(seconds)
    return seconds


def timed(label, fn):
    t0 = time.perf_counter()
    result = fn()
    print(f"{label:<34}: {time.perf_counter() - t0:5.2f} s  -> {result}")


def main():
    gil = getattr(sys, "_is_gil_enabled", lambda: True)()  # Python 3.13+ dapat memberi
tahu
    print(f"Python {sys.version.split()[0]}, GIL enabled: {gil}")
    n = 600_000
    halves = [(0, n // 2), (n // 2, n)]
    timed("CPU work, one thread", lambda: count_primes((0, n)))
    with ThreadPoolExecutor(2) as pool:
        timed("CPU work, two threads", lambda: sum(pool.map(count_primes, halves)))
    with ProcessPoolExecutor(2) as pool:
        timed("CPU work, two processes", lambda: sum(pool.map(count_primes, halves)))
    waits = [0.5, 0.5, 0.5, 0.5]
    timed("four 0.5 s waits, one thread", lambda: sum(map(wait_for_device, waits)))
```

Kode 14.10 (Lanjutan) *gil_demo.py: pekerjaan yang terikat CPU dalam satu thread, dua thread, dan dua proses, serta penantian dalam satu thread dan empat thread.*

```
    with ThreadPoolExecutor(4) as pool:
        timed("four 0.5 s waits, four threads", lambda: sum(pool.map(wait_for_device,
waits)))

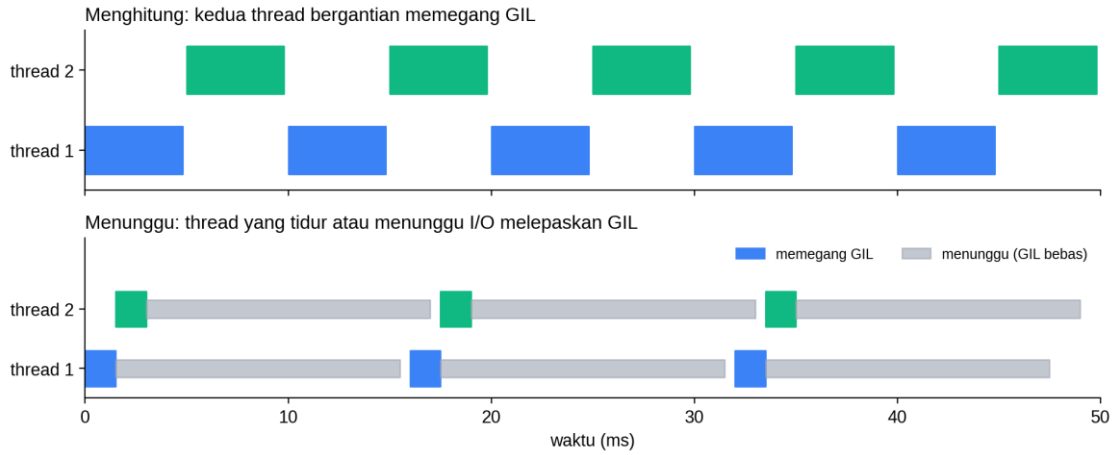

if __name__ == "__main__":
    main()
```

```
Python 3.11.15, GIL enabled: True
CPU work, one thread               :  1.48 s  -> 49098
CPU work, two threads              :  1.25 s  -> 49098
CPU work, two processes            :  1.24 s  -> 49098
four 0.5 s waits, one thread       :  2.00 s  -> 2.0
four 0.5 s waits, four threads     :  0.50 s  -> 2.0
```

Baris-baris penantian paling jelas: empat penantian setengah detik memakan 2.00 detik bila dilakukan berurutan dan 0.50 detik dalam empat thread. Baris-baris komputasi perlu penjelasan. Membagi pencacahan prima di antara dua *thread* tidak memangkas waktunya menjadi separuh dalam jalan mana pun yang kita lakukan; pada komputer multiinti yang tidak sibuk, baris dua thread biasanya kurang lebih sama dengan, atau sedikit *lebih lambat* daripada, baris satu thread. Penyebabnya punya nama.

> **DEFINISI. Global Interpreter Lock (GIL)** adalah sebuah kunci di dalam interpreter Python yang standar, CPython. Sebuah thread harus memegang GIL untuk mengeksekusi *bytecode* Python, dan hanya satu thread yang dapat memegangnya pada satu waktu. Interpreter membuat thread yang sedang berjalan melepas GIL secara berkala (secara bawaan setiap 5 ms) agar thread lain mendapat giliran, dan sebuah thread juga melepasnya setiap kali ia terblokir: di dalam `time.sleep`, saat membaca berkas atau soket, saat menunggu kunci.

Gambar 14.5 memperlihatkan akibatnya. Ketika dua thread *menghitung*, mereka bergantian memegang GIL, sehingga pada setiap saat hanya satu yang maju dan total waktunya sama dengan, atau sedikit lebih buruk daripada, satu thread yang mengerjakan semuanya. Ketika thread *menunggu*, thread yang menunggu tidak memegang GIL, sehingga thread lain berjalan leluasa, dan banyak penantian bertumpang tindih dengan sempurna.



Gambar 14.5 GIL dalam gambar. Atas: dua thread yang menghitung bergantian memegang GIL, sehingga keduanya tidak pernah menghitung pada saat yang sama. Bawah: thread yang lebih banyak menunggu melepas GIL selama menunggu, sehingga penantian mereka bertumpang tindih.

Jadi aturan untuk CPython adalah:

- **Pekerjaan yang terikat I/O** (*I/O-bound*: menunggu disk, jaringan, basis data, mesin yang sedang bekerja): **thread membantu**, sering kali sangat besar. Empat penantian kita turun dari 2.00 detik menjadi 0.50 detik.
- **Pekerjaan yang terikat CPU** (*CPU-bound*: aritmetika dalam Python murni): **thread tidak membantu**. Pakailah `ProcessPoolExecutor` atau modul `multiprocessing`: setiap proses memiliki interpreter dan GIL-nya sendiri, dan proses-proses sungguh menghitung secara paralel. Pada komputer dua inti yang tidak sibuk, baris proses di **Kode 14.10** mendekati separuh baris satu thread; pada mesin sibuk yang menghasilkan keluaran di atas, bahkan proses pun hanya memperoleh sedikit. Proses lebih mahal untuk dimulai daripada thread dan tidak dapat berbagi objek secara langsung (argumen dan hasil disalin di antara mereka), dan itulah sebabnya `count_primes` menerima tupel kecil dan mengembalikan satu bilangan.
- **Pekerjaan yang terikat CPU di dalam pustaka yang ditulis dalam C** (NumPy, kompresi, kriptografi): pustaka seperti itu sering melepas GIL selama menghitung, dan dalam hal itu thread membantu.

Dua peringatan. Pertama, **GIL tidak membuat kode Anda aman-thread.** GIL menjalankan satu instruksi *bytecode* pada satu waktu, bukan satu *metode*; Subbab 14.7 kehilangan pembaruan di Python meskipun ada GIL. Kedua, GIL sedang dalam perjalanan keluar. Python 3.13 memperkenalkan *build* eksperimental *free-threaded* tanpa GIL (PEP 703 [9]), dan Python 3.14 menjadikan *build* itu didukung resmi, walaupun opsional (PEP 779 [10]). Di sana `gil_demo.py` mencetak `GIL enabled: False` (fungsi `sys._is_gil_enabled()` tersedia sejak 3.13), dan baris dua thread menjadi lebih cepat seperti di Java. Kode yang tanpa sengaja bergantung pada GIL dapat rusak tanpanya.

14.7 Race Condition: Ketika Dua Thread Berbagi Pencacah

Inilah kelas yang begitu sederhana sehingga rasanya mustahil keliru: pencacah yang dapat naik dan turun.

```
public class Counter {
    private int c = 0;
    public void increment() { c++; }
    public void decrement() { c--; }
    public int get() { return c; }
}
```

Berikan satu pencacah kepada tiga anak. Setiap anak menaikkannya lalu segera menurunkannya, sejuta kali. Dalam urutan apa pun anak-anak itu berjalan, setiap kenaikan diikuti penurunannya sendiri, sehingga pada akhirnya pencacah harus bernilai 0. Dan bila anak-anak itu hanya menaikkan, tiga anak kali sejuta kenaikan harus menghasilkan tepat 3,000,000. **Kode 14.11** mencoba kedua percobaan, masing-masing lima putaran, dengan pencacah baru di setiap putaran.

Kode 14.11 `RaceDemo.java`: tiga anak berbagi satu pencacah tanpa kunci apa pun; lima putaran naik-dan-turun dan lima putaran naik saja.

```
import java.util.concurrent.CountDownLatch;

public class RaceDemo {
    static class Counter {
        // volatile: setiap c++ sungguh membaca dan menulis memori (Subbab 14.8).
        private volatile int c = 0;

        public void increment() {
            c++;
        }

        public void decrement() {
            c--;
        }

        public int get() {
            return c;
        }
    }

    static class Child implements Runnable {
        private final Counter counter;
        private final int times;
        private final boolean alsoDecrement;
        private final CountDownLatch go;

        Child(Counter counter, int times, boolean alsoDecrement, CountDownLatch go) {
            this.counter = counter;
            this.times = times;
            this.alsoDecrement = alsoDecrement;
            this.go = go;
        }

        @Override
        public void run() {
            try {
                go.await();  // semua anak mulai pada saat yang sama
            } catch (InterruptedException e) {
                return;
            }
            for (int i = 0; i < times; i++) {
                counter.increment();
                if (alsoDecrement) {
                    counter.decrement();
                }
            }
```

Kode 14.11 (Lanjutan) *RaceDemo.java: tiga anak berbagi satu pencacah tanpa kunci apa pun; lima putaran naik-dan-turun dan lima putaran naik saja.*

```
        }
    }

    static int race(int children, int times, boolean alsoDecrement) throws
InterruptedException {
        Counter counter = new Counter();
        CountDownLatch go = new CountDownLatch(1);
        Thread[] threads = new Thread[children];
        for (int k = 0; k < children; k++) {
            threads[k] = new Thread(new Child(counter, times, alsoDecrement, go));
            threads[k].start();
        }
        go.countDown();  // buka gerbangnya
        for (Thread t : threads) {
            t.join();
        }
        return counter.get();
    }

    public static void main(String[] args) throws InterruptedException {
        final int children = 3;
        final int times = 1_000_000;
        System.out.println("Experiment 1: each child increments and decrements " + times +
" times");
        for (int round = 1; round <= 5; round++) {
            System.out.println("  round " + round + ": counter should be 0 and actually is
"
                    + race(children, times, true));
        }
        int expected = children * times;
        System.out.println("Experiment 2: each child increments " + times + " times");
        for (int round = 1; round <= 5; round++) {
            int actual = race(children, times, false);
            System.out.println("  round " + round + ": should be " + expected + ", is " +
actual
                    + ", lost " + (expected - actual));
        }
    }
}
```

Satu kali jalan yang sungguhan mencetak:

```
Experiment 1: each child increments and decrements 1000000 times
  round 1: counter should be 0 and actually is 0
  round 2: counter should be 0 and actually is 0
  round 3: counter should be 0 and actually is 0
  round 4: counter should be 0 and actually is 399
  round 5: counter should be 0 and actually is 249
Experiment 2: each child increments 1000000 times
  round 1: should be 3000000, is 2423058, lost 576942
  round 2: should be 3000000, is 2924583, lost 75417
  round 3: should be 3000000, is 1020932, lost 1979068
  round 4: should be 3000000, is 3000000, lost 0
  round 5: should be 3000000, is 3000000, lost 0
```

Pencacah yang “harus 0” berakhir di 399 dan 249 pada dua dari lima putaran. Pencacah yang “harus 3,000,000” kurang 576,942 pada putaran 1 dan kurang hampir dua juta pada putaran 3: dua pertiga dari semua kenaikan di putaran itu lenyap. Tidak ada eksepsi yang dilempar, tidak ada peringatan yang tercetak, dan pada putaran 4 dan 5 jawabannya sempurna. Inilah **race condition** (kondisi balapan), dan kenyataan bahwa ia kadang memberi jawaban yang benar justru membuatnya begitu berbahaya.

> **DEFINISI. Race condition** adalah *bug* yang membuat hasil program bergantung pada pewaktuan relatif thread-threadnya, sehingga sebagian selang-seling memberi hasil yang salah. Jenis yang paling umum adalah **pembaruan yang hilang** (*lost update*): dua thread membaca nilai lama yang sama dari variabel bersama, masing-masing menghitung nilai baru darinya, dan operasi tulis kedua menimpa yang pertama.

Apa yang berubah-ubah di sini, dan apa yang tidak? Jumlah pembaruan yang hilang sangat berbeda antarputaran dan antarjalan, dan sebagian putaran tidak kehilangan apa pun. Kita menjalankan program itu sepuluh kali: pada tujuh dari sepuluh jalan paling sedikit satu putaran naik-saja kehilangan pembaruan, dan pada delapan dari sepuluh paling sedikit satu putaran naik-dan-turun berakhir tidak di 0; pada tiga jalan lainnya kelima putaran naik-saja tepat, dan satu jalan sempurna di kedua percobaan. Pada komputer dengan lebih banyak inti bebas, kehilangannya biasanya lebih besar dan lebih sering. Yang tidak pernah berubah adalah *arahnya*: pembaruan yang hilang hanya dapat membuat pencacah naik-saja terlalu kecil, tidak pernah terlalu besar, dan total yang benar hanya terjamin bila tidak ada dua pembaruan yang bertumpang tindih.

14.7.1 Mengapa `c++` bukan satu langkah

Masalahnya, `c++` bukan satu operasi. Tanyakan kepada pembongkar (*disassembler*) milik JDK, `javap -c`, apa yang dihasilkan kompiler untuk `increment()` di kelas bersarang `Counter` (README bab ini memberikan perintah persisnya):

```
public void increment();
  Code:
     0: aload_0
     1: dup
     2: getfield      #7                  // Field c:I
     5: iconst_1
     6: iadd
     7: putfield      #7                  // Field c:I
    10: return
```

Empat instruksi mengerjakan tugas itu. `getfield` **membaca** nilai `c` saat ini dari objek ke area kerja thread itu sendiri (*operand stack*-nya); `iconst_1` dan `iadd` **menambahkan** satu pada salinan pribadi itu; `putfield` **menulis** hasilnya kembali ke objek. Baca, ubah, tulis: tiga langkah, dan penjadwal dapat menjeda sebuah thread di antara dua langkah mana pun, sementara thread lain berjalan dan mengubah `c`.

Gambar 14.6 memperlambat tabrakan itu menjadi langkah-langkah tunggal, dengan dua anak, yang satu melakukan `c++` lalu `c--`, yang lain juga begitu.



Gambar 14.6 Pembaruan yang hilang, langkah demi langkah. Anak 1 membaca 0; Anak 2 membaca 0 dan menulis 1; Anak 1 menulis angka 1 miliknya sendiri di atasnya. Dua kenaikan sudah terjadi dan pencacah menunjukkan 1. Dua langkah kemudian sebuah penurunan juga hilang, dan pencacah berakhir di -1, bukan 0.

Dengan kata-kata, alur eksekusi yang benar adalah: Anak 1 menaikkan, Anak 1 menurunkan, Anak 2 menaikkan, dan seterusnya, setiap operasi tuntas sebelum yang berikutnya dimulai. Yang sungguh dapat terjadi adalah:

1. Anak 1 membaca pencacah: 0.
2. Anak 2 membaca pencacah: 0.
3. Anak 2 menulis 1 (sejauh ini benar: satu kenaikan selesai).
4. Anak 1 juga menulis 1, yang dihitung dari 0 yang dibacanya di langkah 1. Seharusnya 2: kenaikan Anak 2 hilang.
5. Anak 2 membaca 1.
6. Anak 2 menulis 0. Seharusnya 1.
7. Anak 1 membaca 0.
8. Anak 1 menulis -1. Seharusnya 0.

Setiap thread melakukan persis apa yang dikatakan kodenya; kesalahannya sepenuhnya terletak pada selang-selingnya.

Seberapa besar kemungkinan selang-seling yang buruk? Hitunglah. Bila satu thread mengeksekusi m langkah dan thread lain mengeksekusi n langkah, maka setiap eksekusi yang mungkin adalah urutan dari semua $m+n$ langkah yang mempertahankan urutan langkah masing-masing thread, dan banyaknya ada

$$\binom{m+n}{m} = \frac{(m+n)!}{m!\,n!} \tag{14.2}$$

urutan seperti itu. Untuk dua kenaikan yang masing-masing tiga langkah ($m = n = 3$), Persamaan (14.2) memberikan $\binom{6}{3} = 20$ selang-seling. Hanya 2 yang benar: semua langkah thread A sebelum semua langkah thread B, atau sebaliknya. Delapan belas sisanya kehilangan satu pembaruan. Dalam program sungguhan setiap thread mengeksekusi jutaan langkah, banyaknya selang-seling luar biasa besar, dan penjadwal memilih salah satunya untuk setiap jalan. Pengujian dapat memperlihatkan bahwa sebuah race ada; pengujian tidak pernah dapat memperlihatkan bahwa race itu *tidak* ada, karena jalan berikutnya mungkin memilih selang-seling yang belum pernah dicoba pengujian mana pun.

Untuk percobaan naik-saja kita dapat mengatakan dengan tepat apa yang hilang. Dengan T thread yang masing-masing melakukan N kenaikan, banyaknya pembaruan yang hilang dalam satu jalan adalah

$$L = T \cdot N - c_{\text{akhir}} \tag{14.3}$$

dengan c_{akhir} nilai pencacah ketika semua thread sudah selesai; **Kode 14.11** mencetak L sebagai `lost`. Untuk putaran 1 di atas, Persamaan (14.3) memberikan $L = 3 \cdot 1{,}000{,}000 - 2{,}423{,}058 = 576{,}942$.

Satu catatan lagi tentang **Kode 14.11**: field pencacahnya dideklarasikan `volatile`. Subbab 14.3.2 memperlihatkan bahwa tanpa `volatile` Java boleh menyimpan field di register; dalam program ini hal itu akan memungkinkan kompiler mengubah sejuta `c++` menjadi satu `c += 1000000` di akhir perulangan, yang kehilangan pembaruan dalam bongkahan besar atau, bila thread-thread itu kebetulan tidak bertumpang tindih, tidak sama sekali. `volatile` membuat setiap `c++` menjadi operasi baca dan operasi tulis memori yang sungguhan, sehingga peragaan ini memperlihatkan race dalam bentuk buku teksnya. Dan ia membuktikan satu hal yang mengejutkan banyak pemrogram: **`volatile` tidak membuat `c++` aman.** Ia menjamin setiap pembacaan melihat operasi tulis terakhir; ia tidak mencegah thread lain menulis di antara operasi baca dan operasi tulis kita.

14.7.2 Race yang sama di Python

Kode 14.12 mengulangi percobaan naik-saja di Python, pertama dengan `self.c += 1` biasa, lalu dengan tiga langkah yang sama ditulis terpisah dan sebuah jeda di antara operasi baca dan operasi tulis.

Kode 14.12 `race_demo.py`: pembaruan yang hilang pada pencacah bersama di Python, dengan `+=` biasa dan dengan baca-ubah-tulis yang sengaja memberi giliran.

```
"""race_demo.py: pembaruan yang hilang pada pencacah bersama."""
import threading
import time


class Counter:
    def __init__(self):
        self.c = 0

    def increment(self):
        self.c += 1  # baca, tambah satu, tulis kembali: tiga langkah

    def get(self):
        return self.c


class SlowCounter(Counter):
    """Tiga langkah yang sama, dengan jeda di antara pembacaan dan penulisan."""

    def increment(self):
        value = self.c        # 1. baca
        time.sleep(0)         # biarkan thread lain berjalan tepat di sini
        self.c = value + 1    # 2.-3. tambah satu dan tulis kembali


def race(counter, children, times):
    def child():
        for _ in range(times):
            counter.increment()

    threads = [threading.Thread(target=child) for _ in range(children)]
    for t in threads:
        t.start()
    for t in threads:
        t.join()
    return counter.get()
```

Kode 14.12 (Lanjutan) *race_demo.py: pembaruan yang hilang pada pencacah bersama di Python, dengan += biasa dan dengan baca-ubah-tulis yang sengaja memberi giliran.*

```
def main():
    children, times = 3, 200_000
    expected = children * times
    print(f"Experiment 1: {children} children, {times} plain increments each")
    for rnd in range(1, 4):
        actual = race(Counter(), children, times)
        print(f"  round {rnd}: should be {expected}, is {actual}, lost {expected -
actual}")
    times = 2_000
    expected = children * times
    print(f"Experiment 2: {children} children, {times} increments with a pause each")
    for rnd in range(1, 4):
        actual = race(SlowCounter(), children, times)
        print(f"  round {rnd}: should be {expected}, is {actual}, lost {expected -
actual}")


if __name__ == "__main__":
    main()
```

```
Experiment 1: 3 children, 200000 plain increments each
  round 1: should be 600000, is 600000, lost 0
  round 2: should be 600000, is 600000, lost 0
  round 3: should be 600000, is 600000, lost 0
Experiment 2: 3 children, 2000 increments with a pause each
  round 1: should be 6000, is 2001, lost 3999
  round 2: should be 6000, is 2000, lost 4000
  round 3: should be 6000, is 2008, lost 3992
```

Hasil jujurnya dahulu: di Percobaan 1, dengan `self.c += 1` biasa, CPython 3.11 **tidak kehilangan apa pun**, di ketiga putaran ini dan di setiap jalan lain yang kita lakukan. Apakah itu berarti `+=` atomik di Python? Lihatlah *bytecode*-nya, yang dicetak dengan `dis.dis(Counter.increment)`:

```
 10           0 RESUME                   0

 11           2 LOAD_FAST                0 (self)
              4 COPY                     1
              6 LOAD_ATTR                0 (c)
             16 LOAD_CONST               1 (1)
             18 BINARY_OP               13 (+=)
             22 SWAP                     2
             24 STORE_ATTR               0 (c)
             34 LOAD_CONST               0 (None)
             36 RETURN_VALUE
```

Bentuknya sama dengan Java: `LOAD_ATTR` membaca `c`, `BINARY_OP (+=)` menambahkan satu, `STORE_ATTR` menulis hasilnya kembali. Bahasa Python tidak menjanjikan bahwa langkah-langkah ini berjalan bersama. Kebetulan saja begitu di CPython 3.10 dan sesudahnya, karena interpreter hanya mempertimbangkan menyerahkan GIL kepada thread lain di titik-titik tertentu (kira-kira lompatan mundur dan pemanggilan), dan tak satu pun jatuh di dalam urutan pendek ini. Versi CPython yang lebih lama, implementasi Python yang lain, dan *build free-threaded* dari Subbab 14.6.1 dapat kehilangan pembaruan semacam itu, dan memang kehilangan [9]. *Bytecode* yang diperlihatkan juga sedikit berbeda antarversi (Python 3.12 dan 3.13 mencetak nama instruksi yang lain), tetapi bentuk baca, tambah, tulis selalu ada.

Percobaan 2 membuat bahaya itu terlihat sesuai permintaan. `SlowCounter.increment` melakukan tiga langkah yang sama, tetapi memanggil `time.sleep(0)` di antara operasi baca dan operasi tulis. `sleep(0)` tidur selama nol detik; satu-satunya akibatnya adalah melepas GIL dan membiarkan thread lain berjalan, persis hal yang selalu *boleh* dilakukan penjadwal di titik itu. Ia tidak menciptakan *bug* baru; ia hanya membuka jendela *bug* lama cukup lebar untuk dilihat. Hasilnya dramatis: dari 6,000 kenaikan, sekitar 4,000 hilang di setiap putaran. Dengan tiga thread yang bergiliran di jeda itu, hampir setiap pembacaan diikuti oleh dua thread lain yang membaca nilai yang sama sebelum ada yang menulis, sehingga dua dari setiap tiga kenaikan lenyap.

> **PERINGATAN:** "Waktu saya uji, jalan kok" tidak membuktikan apa pun tentang kode konkuren. Race yang kehilangan satu pembaruan dalam sejuta operasi akan lolos dari setiap pengujian yang Anda tulis di kelas, lalu merusak buku kas sungguhan sekali sehari di lingkungan produksi. Satu-satunya pertahanan adalah menemukan setiap data yang dapat diubah oleh lebih dari satu thread dan melindunginya dengan sengaja, seperti yang diperlihatkan subbab berikut.

14.8 Saling Pengecualian: `synchronized`, Kunci, dan Atomik

Obat bagi pembaruan yang hilang adalah membuat ketiga langkah itu *tak terbagi* sejauh menyangkut thread lain: selama satu thread berada di antara operasi baca dan operasi tulisnya,

tidak ada thread lain yang boleh menyentuh pencacah. Rentang kode yang harus berjalan tanpa gangguan disebut **bagian kritis** (*critical section*), dan aturan bahwa paling banyak satu thread boleh berada di dalam suatu bagian kritis pada satu waktu disebut **saling pengecualian** (*mutual exclusion*).

> **DEFINISI. Kunci** (*lock*, juga *mutex*, dari *mutual exclusion*) adalah objek yang paling banyak dapat dipegang satu thread pada satu waktu. Sebuah thread **memperoleh** (*acquire*) kunci sebelum memasuki bagian kritis dan **melepas** (*release*) kunci setelah keluar. Thread yang mencoba memperoleh kunci yang sedang dipegang thread lain akan menunggu (di Java keadaannya `BLOCKED` atau `WAITING`) sampai kunci itu dilepas.

Di Java, **setiap objek memiliki kunci bawaan**, yang disebut *intrinsic lock* atau *monitor* [11]. Kata kunci `synchronized` memakainya dalam dua bentuk.

Metode synchronized memperoleh kunci objek `this` ketika dipanggil dan melepasnya ketika kembali, sekalipun ia kembali dengan melempar eksepsi:

```
public synchronized void increment() {
    c++;
}
```

Blok synchronized memperoleh kunci objek mana pun yang Anda sebutkan, untuk pernyataan-pernyataan di dalam kurung kurawalnya:

```
synchronized (counter) {
    counter.increment();
}
```

Bentuk pertama membuat pencacah melindungi dirinya sendiri; bentuk kedua hanya berhasil bila *semua* pemanggil mengingatnya. Bagaimanapun juga, aturannya adalah: **setiap akses ke data bersama yang diubah oleh suatu thread harus memegang kunci yang sama**, termasuk para pembaca; itulah sebabnya `get()` di bawah juga synchronized.

Selain `synchronized`, paket `java.util.concurrent` menawarkan objek kunci eksplisit dan variabel atomik [6]. **Kode 14.13** menjalankan percobaan naik-saja dari **Kode 14.11** dengan lima obat yang berbeda.

Kode 14.13 `SafeCounters.java`: lima pencacah yang benar: metode synchronized, blok synchronized di pemanggil, `ReentrantLock`, `AtomicInteger`, dan `LongAdder`.

```
import java.util.concurrent.atomic.AtomicInteger;
import java.util.concurrent.atomic.LongAdder;
import java.util.concurrent.locks.ReentrantLock;

public class SafeCounters {
    interface Counter {
        void increment();

        long get();
    }

    // Obat 1: metode synchronized mengunci objek pencacah itu sendiri.
    static class SyncMethodCounter implements Counter {
        private int c = 0;

        public synchronized void increment() {
            c++;
        }

        public synchronized long get() {
            return c;
        }
    }

    // Obat 2: pemanggil mengunci objek dengan blok synchronized (lihat race()).
    static class PlainCounter implements Counter {
        private int c = 0;

        public void increment() {
            c++;
        }

        public long get() {
            return c;
        }
    }

    // Obat 3: kunci eksplisit, dilepas di finally.
    static class LockCounter implements Counter {
        private final ReentrantLock lock = new ReentrantLock();
        private int c = 0;
```

Kode 14.13 (Lanjutan) *SafeCounters.java: lima pencacah yang benar: metode synchronized, blok synchronized di pemanggil, ReentrantLock, AtomicInteger, dan LongAdder.*

```
        public void increment() {
            lock.lock();
            try {
                c++;
            } finally {
                lock.unlock();
            }
        }

        public long get() {
            lock.lock();
            try {
                return c;
            } finally {
                lock.unlock();
            }
        }
    }

    // Obat 4: variabel atomik; perangkat keras melakukan baca-tambah-tulis dalam satu
langkah.
    static class AtomicCounter implements Counter {
        private final AtomicInteger c = new AtomicInteger();

        public void increment() {
            c.incrementAndGet();
        }

        public long get() {
            return c.get();
        }
    }

    // Obat 5: LongAdder menyebar hitungan ke beberapa sel; terbaik bila banyak thread
menambah.
    static class AdderCounter implements Counter {
        private final LongAdder c = new LongAdder();

        public void increment() {
            c.increment();
        }

        public long get() {
            return c.sum();
        }
    }
```

Kode 14.13 (Lanjutan) *SafeCounters.java: lima pencacah yang benar: metode synchronized, blok synchronized di pemanggil, ReentrantLock, AtomicInteger, dan LongAdder.*

```
    static void race(String label, Counter counter, boolean callerLocks)
            throws InterruptedException {
        final int children = 3;
        final int times = 1_000_000;
        Runnable child = () -> {
            for (int i = 0; i < times; i++) {
                if (callerLocks) {
                    synchronized (counter) {
                        counter.increment();
                    }
                } else {
                    counter.increment();
                }
            }
        };
        Thread[] threads = new Thread[children];
        long t0 = System.nanoTime();
        for (int k = 0; k < children; k++) {
            threads[k] = new Thread(child);
            threads[k].start();
        }
        for (Thread t : threads) {
            t.join();
        }
        long ms = (System.nanoTime() - t0) / 1_000_000;
        System.out.printf("%-22s: should be %d, is %d  (%d ms)%n",
                label, children * times, counter.get(), ms);
    }

    public static void main(String[] args) throws InterruptedException {
        race("synchronized methods", new SyncMethodCounter(), false);
        race("synchronized block", new PlainCounter(), true);
        race("ReentrantLock", new LockCounter(), false);
        race("AtomicInteger", new AtomicCounter(), false);
        race("LongAdder", new AdderCounter(), false);
    }
}
```

```
synchronized methods  : should be 3000000, is 3000000  (52 ms)
synchronized block    : should be 3000000, is 3000000  (69 ms)
ReentrantLock         : should be 3000000, is 3000000  (140 ms)
AtomicInteger         : should be 3000000, is 3000000  (81 ms)
LongAdder             : should be 3000000, is 3000000  (76 ms)
```

Setiap pencacah kini mencapai tepat 3,000,000, pada setiap jalan. Waktu di dalam tanda kurung sangat berbeda antarjalan dan antarmesin dan tidak boleh dibaca sebagai tolok ukur; *runtime* Java mengoptimasi kunci yang tidak diperebutkan dengan sangat agresif, dan di mesin dua inti yang sibuk ini peringkatnya berganti dari jalan ke jalan. Total yang benar tidak berubah. Inilah yang dilakukan setiap obat.

- **Metode `synchronized`** (`SyncMethodCounter`) adalah obat paling sederhana dan pilihan bawaan yang tepat untuk kelas kecil yang metodenya pendek.
- **Blok `synchronized` di pemanggil** (`PlainCounter` bersama `synchronized (counter)` di `race`) melindungi kelas yang tidak ditulis untuk aman-thread. Ia benar di sini hanya karena *setiap* pemanggilan melewati blok itu.
- **`ReentrantLock`** (`LockCounter`) adalah objek kunci eksplisit dengan `lock()` dan `unlock()`. Polanya selalu sama: `lock()`, lalu `try`, lalu bagian kritis, lalu `finally { unlock(); }`, agar eksepsi tidak dapat membiarkan kunci terpegang selamanya. Kunci eksplisit dapat melakukan hal-hal yang tidak dapat dilakukan `synchronized`: `tryLock()` menyerah bila kuncinya sedang

dipakai, `tryLock(time, unit)` menyerah setelah batas waktu (kita akan memerlukannya untuk melawan deadlock di Subbab 14.12), dan `new ReentrantLock(true)` membuat kunci yang *adil* (*fair*), yang melayani thread-thread yang menunggu menurut urutan kedatangan (Subbab 14.13).

- **`AtomicInteger`** (`AtomicCounter`) sama sekali tidak memerlukan kunci. `incrementAndGet()` melakukan baca, tambah, dan tulis sebagai **satu instruksi perangkat keras yang tak terbagi** (perulangan *compare-and-swap*): prosesor sendiri menjamin tidak ada inti lain yang dapat mengubah nilainya di antara operasi baca dan operasi tulis. Kelas atomik tersedia untuk `int`, `long`, Boolean, dan referensi.
- **`LongAdder`** (`AdderCounter`) memelihara beberapa sel internal dan membiarkan thread yang berbeda menambah ke sel yang berbeda, lalu menjumlahkannya hanya ketika `sum()` dipanggil. Bila banyak thread sangat sering menaikkan, cara ini lebih cepat daripada satu `AtomicLong`, karena thread-thread itu tidak memperebutkan satu lokasi memori.

> **DEFINISI.** Sebuah operasi bersifat **atomik** bila tidak ada thread lain yang pernah dapat melihatnya setengah jadi: operasi itu belum terjadi sama sekali atau sudah terjadi seluruhnya. `c++` pada `int` tidak atomik; `incrementAndGet()` pada `AtomicInteger` atomik; dan setiap urutan pernyataan menjadi atomik terhadap thread lain yang memakai kunci yang sama.

14.8.1 Mengapa kunci juga membereskan visibilitas

Kunci melakukan dua pekerjaan, bukan satu. Selain menjauhkan thread lain, pelepasan kunci *menerbitkan* setiap operasi tulis yang dilakukan thread selama memegangnya, dan perolehan kunci yang sama sesudahnya *melihat* semuanya. Dalam kosakata Subbab 14.3.2, pelepasan sebuah kunci terjadi-sebelum setiap perolehan kunci yang sama sesudahnya [3], [5]. Itulah sebabnya `synchronized long get()` diperlukan: tanpanya, thread yang memanggil `get()` mungkin melihat nilai `c` yang basi, persis seperti pemutar di **Kode 14.3** melihat bendera yang basi. Jaminan yang sama menjelaskan mengapa `SpeedupDemo` dapat membaca `part[0]` setelah `lower.join()`: segala yang dilakukan sebuah thread terjadi-sebelum `join()` yang berhasil oleh thread lain atasnya.

Kunci Java juga bersifat **reentrant** (dapat dimasuki ulang): thread yang sudah memegang sebuah kunci boleh memperolehnya lagi tanpa memblokir dirinya sendiri, dan harus melepasnya sebanyak ia memperolehnya. Jadi metode synchronized boleh memanggil metode synchronized lain pada objek yang sama. Python membedakan kedua jenis itu secara eksplisit: `threading.Lock` *tidak* reentrant (thread yang memperolehnya dua kali akan menunggu dirinya sendiri selamanya), sedangkan `threading.RLock` reentrant.

14.8.2 Kunci di Python

Kode 14.14 melindungi pencacah lambat dari **Kode 14.12** dengan `threading.Lock`, sambil mempertahankan `sleep(0)` yang memancing itu di dalam bagian kritis.

Kode 14.14 `safe_counters.py`: pencacah baca-ubah-tulis yang lambat dibuat benar dengan `threading.Lock`, ditulis dengan `with` dan dengan `acquire`/`release` eksplisit.

```
"""safe_counters.py: pembaruan yang hilang di race_demo.py, diobati dengan Lock."""
import threading
import time


class LockedCounter:
    def __init__(self):
        self._c = 0
        self._lock = threading.Lock()

    def increment(self):
        with self._lock:          # hanya satu thread setiap saat yang melewati baris ini
            value = self._c
            time.sleep(0)         # jeda yang sama seperti SlowCounter: kini tidak
berbahaya
            self._c = value + 1

    def get(self):
        with self._lock:
            return self._c


class ExplicitLockCounter(LockedCounter):
    """Pernyataan with di atas adalah singkatan dari acquire ... try ... finally
release."""

    def increment(self):
        self._lock.acquire()
        try:
            value = self._c
            time.sleep(0)
            self._c = value + 1
        finally:
            self._lock.release()  # tetap dilepas walaupun badannya melempar eksepsi


def race(counter, children, times):
    def child():
        for _ in range(times):
            counter.increment()
```

Kode 14.14 (Lanjutan) *safe_counters.py: pencacah baca-ubah-tulis yang lambat dibuat benar dengan threading.Lock, ditulis dengan with dan dengan acquire/release eksplisit.*

```
    threads = [threading.Thread(target=child) for _ in range(children)]
    t0 = time.perf_counter()
    for t in threads:
        t.start()
    for t in threads:
        t.join()
    return counter.get(), time.perf_counter() - t0


def main():
    children, times = 3, 2_000
    for cls in (LockedCounter, ExplicitLockCounter):
        value, secs = race(cls(), children, times)
        print(f"{cls.__name__:<20}: should be {children * times}, is {value}"
              f"  ({secs:.2f} s)")


if __name__ == "__main__":
    main()
```

```
LockedCounter       : should be 6000, is 6000  (0.42 s)
ExplicitLockCounter : should be 6000, is 6000  (0.42 s)
```

Kedua pencacah mencapai tepat 6,000 pada setiap jalan. `sleep(0)` masih menyerahkan GIL kepada thread lain di tengah pembaruan, tetapi thread itu lalu mencoba memperoleh `self._lock`, mendapatinya sedang dipegang, dan menunggu; pembaruan itu tuntas sebelum ada yang dapat membaca `_c`. Pernyataan `with self._lock:` memperoleh kunci saat masuk dan melepasnya saat keluar, sekalipun badannya melempar eksepsi. `ExplicitLockCounter` menguraikan apa yang dilakukan `with`: `acquire()`, lalu `try` … `finally: release()`, persis pola `ReentrantLock` di Java. Utamakan `with`; ia tidak mungkin lupa melepas kunci.

Pustaka standar Python tidak memiliki kelas bilangan bulat atomik yang umum. Untuk pencacah, pakailah kunci seperti di atas, atau berikan setiap thread hitungan lokalnya sendiri dan jumlahkan semuanya setelah `join()`, yang sama sekali tidak memerlukan berbagi. **Tabel 14.2** menghimpun alat-alat sinkronisasi dalam bab ini.

Tabel 14.2 Alat-alat sinkronisasi Java dan Python yang dipakai dalam bab ini.

Kebutuhan	Java	Python
saling pengecualian	`synchronized`, `ReentrantLock`	`threading.Lock`, `threading.RLock`
mengunci, tetapi menyerah bila sedang dipakai	`tryLock()`, `tryLock(t, unit)`	`acquire(blocking=False)`, `acquire(timeout=t)`
satu variabel, tanpa kunci	`AtomicInteger`, `AtomicLong`, `LongAdder`	(tidak ada; pakai kunci)
selalu melihat nilai terbaru	field `volatile`	(atribut CPython berperilaku begitu; jangan diandalkan)
menunggu sampai suatu kondisi berlaku	`wait`/`notifyAll`, `Condition`	`threading.Condition`
sinyal sekali pakai	`CountDownLatch`	`threading.Event`
paling banyak *k* thread sekaligus	`Semaphore(k)`	`threading.Semaphore(k)`
mengoper data antarthread	`BlockingQueue`	`queue.Queue`

> **WAWASAN KUNCI:** Kunci terbaik adalah kunci yang tidak Anda perlukan. Data yang tidak pernah dibagi (variabel lokal, hasil per thread yang digabungkan setelah `join()`) tidak dapat balapan, dan data yang tidak pernah berubah (objek imutabel dari Bab 13) dapat dibagi oleh berapa pun thread tanpa kunci sama sekali. Kuncilah apa yang harus dibagi *dan* diubah, dan hanya itu.

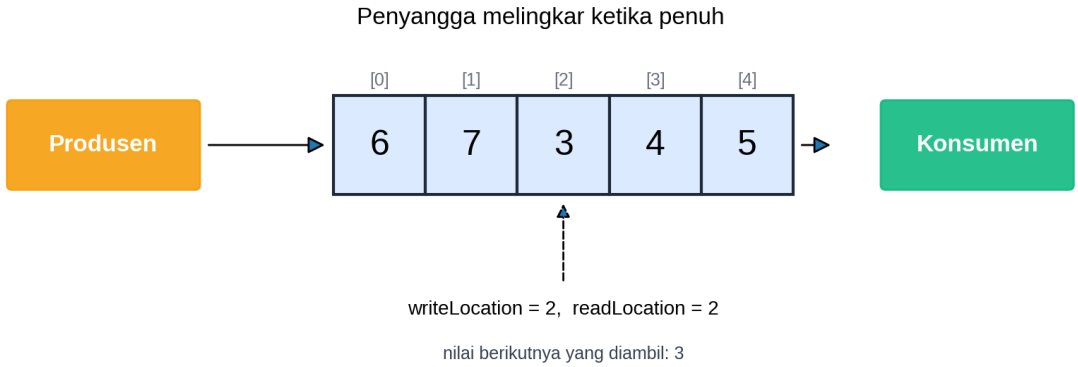
14.9 Menunggu Suatu Kondisi: `wait`, `notifyAll`, dan `Condition`

Kunci menjauhkan thread satu sama lain. Namun sering kali thread juga harus *saling menunggu*: mesin cuci yang tidak punya pekerjaan harus menunggu sampai ada pesanan datang; meja depan yang keranjangnya penuh harus menunggu sampai sebuah mesin mengambil pesanan. Inilah masalah klasik **produsen–konsumen** (*producer–consumer*): satu thread menghasilkan nilai, thread lain memakainya, dan di antara keduanya terdapat **penyangga** (*buffer*) yang ukurannya terbatas. Produsen harus menunggu selama penyangga penuh, dan konsumen harus menunggu selama penyangga kosong.

Penyangga kita memiliki lima sel dan dipakai secara *melingkar*: setelah menulis ke sel 4, produsen melanjutkan lagi di sel 0, asalkan konsumen sudah mengosongkannya. Dua indeks, `writeLocation` dan `readLocation`, menyatakan ke mana nilai berikutnya masuk dan dari mana nilai berikutnya diambil, dan masing-masing maju satu langkah, melingkar kembali, menurut

$$w \leftarrow (w + 1) \bmod n, \qquad r \leftarrow (r + 1) \bmod n \tag{14.4}$$

dengan $n = 5$ sel. Ketika kedua indeks menjadi sama, penyangga itu penuh sama sekali (bila produsen yang baru saja bergerak) atau kosong sama sekali (bila konsumen yang baru saja bergerak); dua bendera Boolean, `writeable` dan `readable`, mengingat yang mana. **Gambar 14.7** memperlihatkan keadaan penuh yang sebentar lagi akan kita jumpai dalam jalan yang sungguhan.



Gambar 14.7 Penyangga melingkar lima sel ketika penuh. Kedua indeks menunjuk ke sel 2: nilai berikutnya bagi konsumen adalah angka 3 yang tersimpan di sana, dan nilai berikutnya dari produsen akan masuk ke sana begitu angka 3 itu diambil.

Bagaimana sebuah thread seharusnya menunggu? Perulangan seperti `while (!writeable) { }` memang berjalan dalam arti tertentu, tetapi ia menghanguskan satu inti penuh tanpa berbuat apa-apa, dan, lebih buruk lagi, ia akan berputar sambil *memegang kunci*, sehingga konsumen tidak akan pernah dapat masuk untuk memberi ruang. Yang kita perlukan adalah cara untuk mengatakan: "lepaskan kunci, tidurlah sampai seseorang memberi tahu bahwa keadaan telah berubah, lalu ambil kembali kuncinya." Setiap objek Java menyediakan persis hal itu, bersama kunci bawaannya [11]:

- **`wait()`**, yang dipanggil sambil memegang kunci sebuah objek, melepas kunci itu dan menidurkan thread (keadaan `WAITING`) di *wait set* (himpunan tunggu) objek tersebut. Ketika thread itu dibangunkan, ia lebih dahulu memperoleh kembali kuncinya, dan baru kemudian `wait()` kembali.
- **`notify()`** membangunkan satu thread dari himpunan tunggu objek itu, bila ada; **`notifyAll()`** membangunkan semuanya. Pemanggilnya harus memegang kunci, dan thread yang dibangunkan belum dapat benar-benar melanjutkan sampai pemanggil melepas kunci itu.

Gabungan sebuah kunci dengan ruang tunggu ini disebut **monitor**, gagasan yang berasal dari Hoare [12]. **Kode 14.15** memakainya untuk penyangga melingkar kita, diadaptasi dari sebuah contoh buku teks klasik [7].

Kode 14.15 `ProCon.java`: seorang produsen dan seorang konsumen yang lebih lambat berbagi penyangga melingkar lima sel; `wait()` dan `notifyAll()` membuat masing-masing menunggu yang lain.

```
import java.util.Arrays;
import java.util.Random;

public class ProCon {
    // Penyangga melingkar lima sel yang dipakai bersama satu produsen dan satu konsumen.
    static class HoldIntegerSynchronized {
        private final int[] sharedInt = {-1, -1, -1, -1, -1};
        private boolean writeable = true;   // adakah sel yang kosong?
        private boolean readable = false;   // adakah sel yang terisi?
        private int readLocation = 0;
        private int writeLocation = 0;

        public synchronized void setSharedInt(int value) throws InterruptedException {
            while (!writeable) {            // while, bukan if: periksa ulang setiap kali
terbangun
                System.out.println("WAITING TO PRODUCE " + value);
                wait();                     // melepas kunci selama menunggu
            }
            sharedInt[writeLocation] = value;
            readable = true;
            System.out.printf("Produced %2d into cell %d", value, writeLocation);
            writeLocation = (writeLocation + 1) % sharedInt.length;
            System.out.printf("   write %d  read %d  buffer: %s%n",
                    writeLocation, readLocation, Arrays.toString(sharedInt));
            if (writeLocation == readLocation) {
                writeable = false;
                System.out.println("BUFFER FULL");
            }
            notifyAll();                    // bangunkan konsumen bila ia sedang menunggu
        }

        public synchronized int getSharedInt() throws InterruptedException {
            while (!readable) {
                System.out.println("WAITING TO CONSUME");
                wait();
            }
            writeable = true;
            int value = sharedInt[readLocation];
            System.out.printf("Consumed %2d from cell %d", value, readLocation);
            readLocation = (readLocation + 1) % sharedInt.length;
            System.out.printf("   write %d  read %d  buffer: %s%n",
                    writeLocation, readLocation, Arrays.toString(sharedInt));
            if (readLocation == writeLocation) {
                readable = false;
                System.out.println("BUFFER EMPTY");
            }
            notifyAll();                    // bangunkan produsen bila ia sedang menunggu
```

Kode 14.15 (Lanjutan) *ProCon.java: seorang produsen dan seorang konsumen yang lebih lambat berbagi penyangga melingkar lima sel; wait() dan notifyAll() membuat masing-masing menunggu yang lain.*

```
            return value;
        }
    }

    public static void main(String[] args) throws InterruptedException {
        HoldIntegerSynchronized buffer = new HoldIntegerSynchronized();
        Thread producer = new Thread(() -> {
            Random r = new Random(1);
            try {
                for (int i = 1; i <= 10; i++) {
                    Thread.sleep(r.nextInt(200));  // waktu untuk membuat nilai berikutnya
                    buffer.setSharedInt(i);
                }
            } catch (InterruptedException e) {
                return;
            }
            System.out.println("Producer finished producing values");
        });
        Thread consumer = new Thread(() -> {
            Random r = new Random(7);
            int sum = 0;
            try {
                for (int i = 1; i <= 10; i++) {
                    Thread.sleep(r.nextInt(700));  // konsumen lebih lambat
                    sum += buffer.getSharedInt();
                }
            } catch (InterruptedException e) {
                return;
            }
            System.out.println("Consumer retrieved values totaling: " + sum);
        });
        producer.start();
        consumer.start();
        producer.join();
        consumer.join();
    }
}
```

```
WAITING TO CONSUME
Produced  1 into cell 0   write 1  read 0  buffer: [1, -1, -1, -1, -1]
Consumed  1 from cell 0   write 1  read 1  buffer: [1, -1, -1, -1, -1]
BUFFER EMPTY
Produced  2 into cell 1   write 2  read 1  buffer: [1, 2, -1, -1, -1]
Produced  3 into cell 2   write 3  read 1  buffer: [1, 2, 3, -1, -1]
Consumed  2 from cell 1   write 3  read 2  buffer: [1, 2, 3, -1, -1]
Produced  4 into cell 3   write 4  read 2  buffer: [1, 2, 3, 4, -1]
Produced  5 into cell 4   write 0  read 2  buffer: [1, 2, 3, 4, 5]
Produced  6 into cell 0   write 1  read 2  buffer: [6, 2, 3, 4, 5]
Produced  7 into cell 1   write 2  read 2  buffer: [6, 7, 3, 4, 5]
BUFFER FULL
WAITING TO PRODUCE 8
Consumed  3 from cell 2   write 2  read 3  buffer: [6, 7, 3, 4, 5]
Produced  8 into cell 2   write 3  read 3  buffer: [6, 7, 8, 4, 5]
BUFFER FULL
Consumed  4 from cell 3   write 3  read 4  buffer: [6, 7, 8, 4, 5]
Produced  9 into cell 3   write 4  read 4  buffer: [6, 7, 8, 9, 5]
BUFFER FULL
Consumed  5 from cell 4   write 4  read 0  buffer: [6, 7, 8, 9, 5]
Produced 10 into cell 4   write 0  read 0  buffer: [6, 7, 8, 9, 10]
BUFFER FULL
Producer finished producing values
Consumed  6 from cell 0   write 0  read 1  buffer: [6, 7, 8, 9, 10]
Consumed  7 from cell 1   write 0  read 2  buffer: [6, 7, 8, 9, 10]
Consumed  8 from cell 2   write 0  read 3  buffer: [6, 7, 8, 9, 10]
Consumed  9 from cell 3   write 0  read 4  buffer: [6, 7, 8, 9, 10]
Consumed 10 from cell 4   write 0  read 0  buffer: [6, 7, 8, 9, 10]
BUFFER EMPTY
Consumer retrieved values totaling: 55
```

Ikutilah kisah yang dituturkan keluaran itu. Konsumen kebetulan siap lebih dahulu dan mendapati penyangga kosong, maka ia mencetak `WAITING TO CONSUME` dan menunggu. Nilai pertama dari produsen membangunkannya; ia mengambil 1, dan penyangga kosong lagi. Lalu produsen, yang lebih cepat, melaju: ia menulis 2 sampai 7 sementara konsumen hanya mengambil 2, melingkar dari sel 4 ke sel 0 dan 1 sebagaimana ditetapkan Persamaan (14.4), dan pada angka 7 indeks tulis menyusul indeks baca: `BUFFER FULL`. Ketika mencoba menulis 8, produsen tidak menemukan sel kosong, mencetak `WAITING TO PRODUCE 8`, lalu menunggu. Setiap kali konsumen mengambil sebuah nilai, `notifyAll()` miliknya membangunkan produsen, yang segera mengisi sel yang baru kosong, sehingga penyangga tetap penuh sampai produsen selesai di angka 10. Konsumen lalu menghabiskan lima nilai yang tersisa, penyangga menjadi kosong, dan total milik konsumen adalah $1 + 2 + \cdots + 10 = 55$.

Larik itu terus memperlihatkan nilai yang sudah dikonsumsi, seperti angka 3 di `[6, 7, 3, 4, 5]`: mengonsumsi hanya memindahkan `readLocation`, dan angka lama tetap ada sampai ditimpa. Itulah sebabnya bendera, bukan isi sel, yang menyatakan apakah penyangga penuh atau kosong.

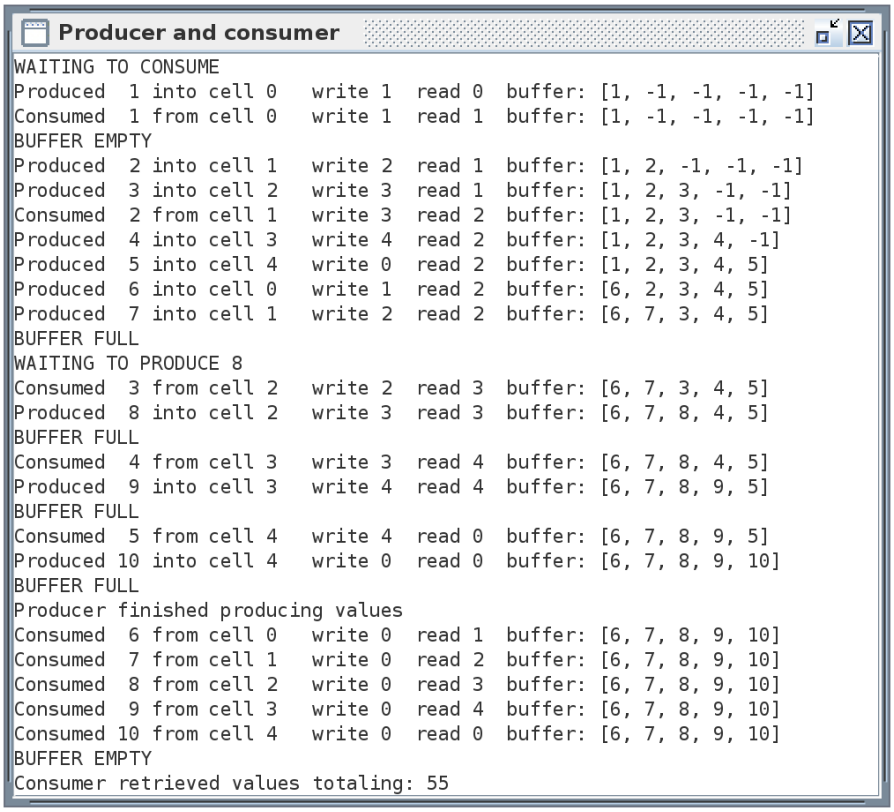
Apa yang berubah-ubah antarjalan? Benih 1 dan 7 menetapkan waktu tidur, sehingga kebanyakan jalan mencetak baris-baris yang sama; tetapi bila saat bangun produsen dan konsumen jatuh dalam selisih satu milidetik, dua baris dapat bertukar tempat dan sebuah baris `WAITING` dapat muncul atau hilang. Yang tidak pernah berubah: nilai-nilai dikonsumsi dalam urutan 1, 2, ..., 10, masing-masing tepat sekali, penyangga tidak pernah berisi lebih dari lima nilai, dan totalnya 55.

Sekarang rincian `setSharedInt` dan `getSharedInt`, yang merupakan jantung persoalannya.

- Kedua metode itu `synchronized`, sehingga keduanya memegang kunci penyangga selama memeriksa dan mengubah field-fieldnya. Bendera dan indeks adalah data bersama, dan aturan Subbab 14.8 berlaku sepenuhnya atasnya.

- `while (!writeable) { … wait(); }`: `while`, **bukan** `if`. Ketika thread yang menunggu terbangun, kondisi yang ditunggunya mungkin tidak lagi berlaku: thread lain mungkin sudah lebih dahulu masuk dan memakai sel kosong itu, dan Java bahkan mengizinkan *spurious wake-up* (bangun palsu), yaitu thread kembali dari `wait()` walaupun tidak ada yang memanggil `notify`. Satu-satunya pola yang aman adalah menguji ulang kondisinya setiap kali terbangun, dan itulah yang dilakukan perulangan `while` [2], [11].
- `wait()` **melepas kunci.** Ini mutlak: selama produsen menunggu di dalam `setSharedInt`, konsumen dapat memasuki `getSharedInt` pada objek yang sama, yang tidak mungkin dilakukannya bila produsen masih memegang kunci.
- `notifyAll()`, **bukan** `notify()`. Dengan tepat satu produsen dan satu konsumen, `notify()` akan berhasil. Dengan beberapa produsen dan konsumen, `notify()` mungkin membangunkan thread dari jenis yang salah (produsen, padahal hanya konsumen yang dapat maju), yang kembali menunggu sementara thread yang tepat tetap tidur: program macet. `notifyAll()` membangunkan semua dan membiarkan perulangan-perulangan `while` menentukan siapa yang boleh melanjutkan. Harganya beberapa kebangunan yang sia-sia; ia tidak pernah kehilangan satu pun.

Pertukaran yang sama dapat pula ditampilkan di jendela alih-alih di konsol. **Gambar 14.8** memasukkan baris-baris dari jalannya program ini ke area teks Swing: pesan-pesan yang sama, lingkaran balik yang sama, dan total 55 yang sama.



Gambar 14.8 Keluaran `ProCon.java` yang ditampilkan di area teks Swing: produsen menulis 1 sampai 10 ke lima sel melingkar, menunggu ketika penyangga penuh, dan total milik konsumen adalah 55.

Di Python, monitor adalah objek eksplisit, `threading.Condition`: sebuah kunci dan ruang tunggu sekaligus. `with cond:` memperoleh kuncinya, `cond.wait()` melepas kunci dan menunggu, dan `cond.notify_all()` membangunkan para penunggu. **Kode 14.16** adalah terjemahan baris demi baris.

Kode 14.16 `procon.py`: penyangga melingkar yang sama dengan `threading.Condition`.

```
"""procon.py: satu produsen, satu konsumen, dan penyangga melingkar lima sel."""
import random
import threading
import time


class HoldIntegerSynchronized:
    def __init__(self):
        self.shared_int = [-1, -1, -1, -1, -1]
        self.writeable = True    # adakah sel yang kosong?
        self.readable = False    # adakah sel yang terisi?
        self.read_location = 0
        self.write_location = 0
        self.cond = threading.Condition()  # sebuah kunci ditambah ruang tunggu

    def _state(self):
        return (f"   write {self.write_location}  read {self.read_location}"
                f"  buffer: {self.shared_int}")

    def set_shared_int(self, value):
        with self.cond:
            while not self.writeable:      # while, bukan if: periksa ulang setiap kali
terbangun
                print(f"WAITING TO PRODUCE {value}")
                self.cond.wait()           # melepas kunci selama menunggu
            self.shared_int[self.write_location] = value
            self.readable = True
            cell = self.write_location
            self.write_location = (self.write_location + 1) % len(self.shared_int)
            print(f"Produced {value:2d} into cell {cell}" + self._state())
            if self.write_location == self.read_location:
                self.writeable = False
                print("BUFFER FULL")
            self.cond.notify_all()         # bangunkan konsumen bila ia sedang menunggu

    def get_shared_int(self):
        with self.cond:
            while not self.readable:
                print("WAITING TO CONSUME")
                self.cond.wait()
            self.writeable = True
            value = self.shared_int[self.read_location]
            cell = self.read_location
            self.read_location = (self.read_location + 1) % len(self.shared_int)
            print(f"Consumed {value:2d} from cell {cell}" + self._state())
            if self.read_location == self.write_location:
                self.readable = False
```

Kode 14.16 (Lanjutan) *procon.py: penyangga melingkar yang sama dengan threading.Condition.*

```
                print("BUFFER EMPTY")
            self.cond.notify_all()          # bangunkan produsen bila ia sedang menunggu
            return value


def main():
    buffer = HoldIntegerSynchronized()

    def produce():
        r = random.Random(7)
        for i in range(1, 11):
            time.sleep(r.randrange(200) / 1000)  # waktu untuk membuat nilai berikutnya
            buffer.set_shared_int(i)
        print("Producer finished producing values")

    def consume():
        r = random.Random(31)
        total = 0
        for _ in range(10):
            time.sleep(r.randrange(700) / 1000)  # konsumen lebih lambat
            total += buffer.get_shared_int()
        print(f"Consumer retrieved values totaling: {total}")

    producer = threading.Thread(target=produce)
    consumer = threading.Thread(target=consume)
    producer.start()
    consumer.start()
    producer.join()
    consumer.join()


if __name__ == "__main__":
    main()
```

```
WAITING TO CONSUME
Produced  1 into cell 0   write 1  read 0  buffer: [1, -1, -1, -1, -1]
Consumed  1 from cell 0   write 1  read 1  buffer: [1, -1, -1, -1, -1]
BUFFER EMPTY
Produced  2 into cell 1   write 2  read 1  buffer: [1, 2, -1, -1, -1]
Produced  3 into cell 2   write 3  read 1  buffer: [1, 2, 3, -1, -1]
Produced  4 into cell 3   write 4  read 1  buffer: [1, 2, 3, 4, -1]
Produced  5 into cell 4   write 0  read 1  buffer: [1, 2, 3, 4, 5]
Produced  6 into cell 0   write 1  read 1  buffer: [6, 2, 3, 4, 5]
BUFFER FULL
WAITING TO PRODUCE 7
Consumed  2 from cell 1   write 1  read 2  buffer: [6, 2, 3, 4, 5]
Produced  7 into cell 1   write 2  read 2  buffer: [6, 7, 3, 4, 5]
BUFFER FULL
WAITING TO PRODUCE 8
Consumed  3 from cell 2   write 2  read 3  buffer: [6, 7, 3, 4, 5]
Produced  8 into cell 2   write 3  read 3  buffer: [6, 7, 8, 4, 5]
BUFFER FULL
WAITING TO PRODUCE 9
Consumed  4 from cell 3   write 3  read 4  buffer: [6, 7, 8, 4, 5]
Produced  9 into cell 3   write 4  read 4  buffer: [6, 7, 8, 9, 5]
BUFFER FULL
Consumed  5 from cell 4   write 4  read 0  buffer: [6, 7, 8, 9, 5]
Produced 10 into cell 4   write 0  read 0  buffer: [6, 7, 8, 9, 10]
BUFFER FULL
Producer finished producing values
Consumed  6 from cell 0   write 0  read 1  buffer: [6, 7, 8, 9, 10]
Consumed  7 from cell 1   write 0  read 2  buffer: [6, 7, 8, 9, 10]
Consumed  8 from cell 2   write 0  read 3  buffer: [6, 7, 8, 9, 10]
Consumed  9 from cell 3   write 0  read 4  buffer: [6, 7, 8, 9, 10]
Consumed 10 from cell 4   write 0  read 0  buffer: [6, 7, 8, 9, 10]
BUFFER EMPTY
Consumer retrieved values totaling: 55
```

Kisahnya sama, dengan waktu tidur yang berbeda dari pembangkit bilangan acak Python dan benih 7 dan 31: konsumen menunggu nilai pertama, produsen melaju dan memenuhi penyangga pada angka 6, lalu menunggu tiga kali, untuk 7, 8, dan 9. Nilai-nilainya, urutannya, dan total 55, seperti biasa, sama. Java juga menawarkan gaya eksplisit yang sama: sebuah `ReentrantLock` dapat menciptakan objek `Condition` dengan `lock.newCondition()`, yang `await()` dan `signalAll()`-nya berperan sebagai `wait()` dan `notifyAll()`, dan satu kunci boleh memiliki beberapa kondisi, misalnya satu untuk “tidak penuh” dan satu untuk “tidak kosong”.

14.10 Produsen dan Konsumen dengan Cara Mudah: Antrean Pemblokir

Menulis `wait` dan `notifyAll` dengan benar adalah latihan yang baik dan kebiasaan yang buruk. Pustaka standar kedua bahasa berisi penyangga yang sudah mengerjakan semuanya. Di Java, **`BlockingQueue`** memiliki metode `put(x)`, yang menunggu selama antrean penuh, dan metode `take()`, yang menunggu selama antrean kosong; `ArrayBlockingQueue` berkapasitas tetap dan `LinkedBlockingQueue` boleh tak terbatas [6]. **Kode 14.17** adalah produsen dan konsumen dari **Kode 14.15** sekali lagi, dipangkas sampai ke intinya.

Kode 14.17 `QueueDemo.java`: pertukaran produsen–konsumen dengan `ArrayBlockingQueue` berkapasitas lima dan sebuah pil racun.

```java
import java.util.concurrent.ArrayBlockingQueue;
import java.util.concurrent.BlockingQueue;

public class QueueDemo {
    public static void main(String[] args) throws InterruptedException {
        BlockingQueue<Integer> buffer = new ArrayBlockingQueue<>(5);  // lima sel
        final int done = -1;  // "pil racun": tidak ada nilai sungguhan yang negatif

        Thread producer = new Thread(() -> {
            try {
                for (int i = 1; i <= 10; i++) {
                    buffer.put(i);       // menunggu selama antrean penuh
                }
                buffer.put(done);
            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
            }
        });
        Thread consumer = new Thread(() -> {
            int sum = 0;
            int count = 0;
            try {
                while (true) {
                    int value = buffer.take();  // menunggu selama antrean kosong
                    if (value == done) {
                        break;
                    }
                    sum += value;
                    count++;
                }
            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
            }
            System.out.println("Consumer took " + count + " values totaling: " + sum);
        });
        producer.start();
        consumer.start();
        producer.join();
        consumer.join();
        System.out.println("queue left empty: " + buffer.isEmpty());
    }
}
```

```
Consumer took 10 values totaling: 55
queue left empty: true
```

Semua bendera, indeks, perulangan `while`, dan notifikasi dari **Kode 14.15** telah lenyap ke dalam antrean, dan keluarannya sama pada setiap jalan. Satu gagasan baru muncul: **pil racun** (*poison pill*). Bagaimana konsumen tahu bahwa tidak akan ada nilai lagi? Produsen memasukkan sebuah nilai khusus, di sini -1, yang tidak mungkin sama dengan nilai sungguhan mana pun, dan konsumen berhenti ketika mengambilnya. Dengan beberapa konsumen, produsen memasukkan satu pil per konsumen.

Blok `catch (InterruptedException e)` memanggil `Thread.currentThread().interrupt()` alih-alih diam-diam mengabaikan eksepsi itu. Menangkap `InterruptedException` menghapus bendera interupsi thread; memasangnya lagi menjaga informasi itu bagi kode yang lebih atas yang ingin tahu apakah thread itu diminta berhenti. Itulah sikap sopan setiap kali Anda menangkap interupsi yang tidak Anda tangani sepenuhnya sendiri [2].

Kode 14.18 memakai `queue.Queue` milik Python, yang `put` dan `get`-nya menunggu dengan cara yang sama ketika antrean penuh atau kosong.

Kode 14.18 `queue_demo.py`: pertukaran yang sama dengan `queue.Queue(maxsize=5)` dan `None` sebagai pil racun.

```
"""queue_demo.py: produsen dan konsumen lagi, dengan queue.Queue yang mengurus
penantian."""
import queue
import threading

DONE = None  # "pil racun": tidak ada nilai sungguhan yang None


def main():
    buffer = queue.Queue(maxsize=5)  # lima sel

    def produce():
        for i in range(1, 11):
            buffer.put(i)      # menunggu selama antrean penuh
        buffer.put(DONE)

    def consume():
        total = count = 0
        while True:
            value = buffer.get()  # menunggu selama antrean kosong
            if value is DONE:
                break
            total += value
            count += 1
        print(f"Consumer took {count} values totaling: {total}")

    producer = threading.Thread(target=produce)
    consumer = threading.Thread(target=consume)
    producer.start()
    consumer.start()
    producer.join()
    consumer.join()
    print("queue left empty:", buffer.empty())


if __name__ == "__main__":
    main()
```

```
Consumer took 10 values totaling: 55
queue left empty: True
```

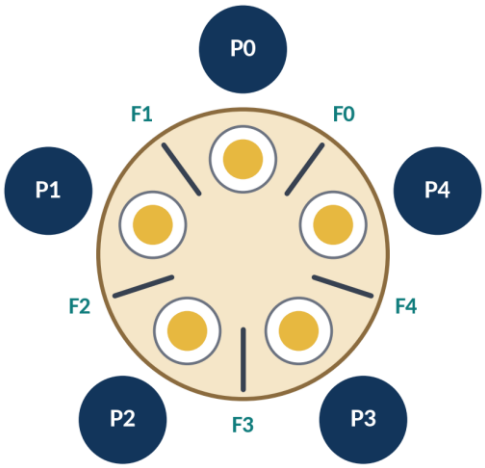
Di dalamnya, `queue.Queue` adalah sebuah list yang dilindungi `threading.Lock` dan dua objek `threading.Condition`, satu untuk "tidak kosong" dan satu untuk "tidak penuh": persis mesin dari **Kode 14.16**, yang ditulis sekali, dengan cermat, oleh para pengembang pustaka standar. Pakailah itu. Di kedua bahasa, antrean pemblokir adalah alat tunggal yang paling berguna untuk program konkuren, karena ia menggantikan *data bersama* (yang memerlukan kunci) dengan *pesan* (yang diserahkan dari satu thread ke thread lain dan sesudahnya menjadi milik penerimanya saja).

14.11 Deadlock

Kunci menyembuhkan race, tetapi membawa bahaya baru. Thread yang memegang satu kunci dan menunggu kunci kedua dapat menunggu selamanya bila kunci kedua itu dipegang oleh thread yang sedang menunggu kunci pertama. Tak satu pun dapat bergerak, dan tak satu pun akan pernah melepaskan. Inilah **deadlock** (kebuntuan).

14.11.1 Para filsuf yang bersantap

Ilustrasi yang paling terkenal diajukan oleh Edsger Dijkstra pada 1965 dan diberi bentuknya yang sekarang oleh Tony Hoare [13]. Lima filsuf duduk mengelilingi meja bundar, masing-masing di depan sepiring spageti. Di antara setiap pasangan tetangga tergeletak satu garpu, sehingga seluruhnya ada lima garpu, seperti diperlihatkan **Gambar 14.9**. Seorang filsuf bergantian berpikir dan makan. Untuk makan, seorang filsuf memerlukan *kedua* garpu, yang di kiri dan yang di kanan; sebuah garpu hanya dapat dipegang satu filsuf pada satu waktu, sehingga dua tetangga tidak dapat makan bersamaan. Selesai makan, filsuf meletakkan kedua garpu dan kembali berpikir. Spagetinya tidak pernah habis, dan para filsuf tidak pernah berhenti lapar. Masalahnya adalah merancang suatu disiplin perilaku, sebuah *algoritme konkuren*, yang menjamin tidak ada filsuf yang kelaparan, walaupun tidak ada filsuf yang tahu kapan filsuf lain ingin makan atau berpikir.



Gambar 14.9 Para filsuf yang bersantap: lima filsuf, lima piring spageti, dan satu garpu di antara setiap pasangan tetangga. Filsuf P_i memerlukan garpu F_i dan $F_{(i+1) \bmod 5}$, seperti dalam `Philosopher.java`.

Disiplin yang paling jelas adalah: "ambil garpu kiri; lalu ambil garpu kanan; makan; letakkan keduanya." Setiap filsuf adalah sebuah thread dan setiap garpu adalah sebuah kunci. **Kode 14.19**, diadaptasi dari program peraga yang terkenal [14], mendudukkan lima filsuf seperti itu dan menggambar meja setiap 100 ms. Dalam setiap gambar, `^` adalah garpu yang tergeletak di meja, `_O_` adalah filsuf yang tidak memegang garpu, `<O` filsuf yang hanya memegang garpu kiri, dan `<O>` filsuf yang memegang keduanya dan sedang makan. Garpu i digambar tepat di sebelah kiri filsuf i, dan garpu kanan filsuf i adalah garpu $i + 1$, yang digambar di sebelah kiri filsuf berikutnya (mejanya bundar, sehingga garpu kanan filsuf 5 adalah garpu 1, karakter pertama baris itu).

Kode 14.19 `Philosopher.java`: lima filsuf, masing-masing sebuah thread, masing-masing mengambil garpu kiri lalu garpu kanan; `main` menggambar meja sampai setiap filsuf memegang tepat satu garpu.

```
import java.util.Arrays;
import java.util.Random;

public class Philosopher extends Thread {
    static final int N = 5;                         // filsuf dan garpu
    static final int UNIT = 100;                    // satu satuan waktu, dalam milidetik
    static final int MAX_STEPS = 300;               // jaring pengaman: menyerah setelah 30
detik
    static final Object[] forks = new Object[N];   // garpu i terletak di sebelah kiri
filsuf i
    static final String[] forkCell = new String[N];
    static final String[] dinerCell = new String[N];
    static final int[] meals = new int[N];
    static final Random random = new Random();

    private final int me;

    Philosopher(int me) {
        this.me = me;
    }

    // Gambar meja dipakai bersama, jadi diubah dan dibaca di bawah satu kunci.
    static synchronized void show(int fork, String forkText, int diner, String dinerText) {
        forkCell[fork] = forkText;
        dinerCell[diner] = dinerText;
    }

    static synchronized void ate(int diner) {
        meals[diner]++;
    }

    static synchronized String mealsEaten() {
        return Arrays.toString(meals);
    }

    static synchronized String table() {
        StringBuilder sb = new StringBuilder();
        for (int i = 0; i < N; i++) {
            sb.append(forkCell[i]).append(' ').append(dinerCell[i]).append(' ');
        }
        return sb.toString();
    }
```

Kode 14.19 (Lanjutan) *Philosopher.java: lima filsuf, masing-masing sebuah thread, masing-masing mengambil garpu kiri lalu garpu kanan; main menggambar meja sampai setiap filsuf memegang tepat satu garpu.*

```
    @Override
    public void run() {
        int left = me;
        int right = (me + 1) % N;
        try {
            while (!isInterrupted()) {
                sleep(UNIT * random.nextInt(6));            // berpikir
                synchronized (forks[left]) {                // ambil garpu kiri
                    show(left, " ", me, "<O ");
                    sleep(UNIT);
                    synchronized (forks[right]) {           // ambil garpu kanan
                        show(right, " ", me, "<O>");
                        sleep(UNIT);                        // makan
                        ate(me);
                        show(right, "^", me, "<O ");        // letakkan kembali garpu
kanan
                    }
                    show(left, "^", me, "_O_");             // letakkan kembali garpu kiri
                }
            }
        } catch (InterruptedException e) {
            // diinterupsi saat berpikir atau makan: tinggalkan meja
        }
    }

    public static void main(String[] args) throws InterruptedException {
        for (int i = 0; i < N; i++) {
            forks[i] = new Object();
            forkCell[i] = "^";
            dinerCell[i] = "_O_";
        }
        String locked = "  <O  ".repeat(N);  // setiap filsuf memegang tepat satu garpu
        for (int i = 0; i < N; i++) {
            Thread t = new Philosopher(i);
            t.setDaemon(true);   // thread daemon tidak dapat membuat program tetap hidup
            t.start();
        }
        System.out.println("The diner table:");
        for (int step = 1; step <= MAX_STEPS; step++) {
            String now = table();
            System.out.printf("%s %3d%n", now, step);
            if (now.equals(locked)) {
                Thread.sleep(5 * UNIT);  // pastikan: tak seorang pun dapat bergerak lagi
                if (table().equals(locked)) {
                    System.out.println("The diner is locked.");
                    System.out.println("Meals eaten before the deadlock: " + mealsEaten());
                    return;
```

Kode 14.19 (Lanjutan) *Philosopher.java: lima filsuf, masing-masing sebuah thread, masing-masing mengambil garpu kiri lalu garpu kanan; main menggambar meja sampai setiap filsuf memegang tepat satu garpu.*

```
                }
            }
            Thread.sleep(UNIT);
        }
        System.out.println("No deadlock within " + MAX_STEPS + " steps this time.");
    }
}
```

Dalam satu jalan yang sungguhan, meja itu terkunci setelah 36 gambar:

```
The diner table:
^ _O_ ^ _O_   <O  ^ _O_ ^ _O_    1
^ _O_ ^ _O_   <O    <O    <O     2
  _O_ ^ _O_   <O    <O    <O>    3
  <O    <O    <O    <O>   _O_    4
  <O    <O    <O>   _O_ ^ _O_    5
  <O    <O>   _O_ ^ _O_ ^ _O_    6
  <O>   _O_   <O  ^ _O_ ^ _O_    7
  <O  ^ _O_   <O    <O    <O     8
  <O>   _O_   <O    <O    <O     9
  _O_ ^ _O_   <O    <O    <O>   10
  <O  ^ _O_   <O    <O>   _O_   11
  <O    <O    <O>   _O_ ^ _O_   12
  <O    <O>   _O_ ^ _O_ ^ _O_   13
  <O>   _O_   <O    <O  ^ _O_   14
^ _O_ ^ _O_   <O    <O    <O    15
  _O_ ^ _O_   <O    <O    <O>   16
  <O    <O    <O    <O>   _O_   17
  <O    <O    <O>   _O_ ^ _O_   18
  <O    <O>   _O_ ^ _O_ ^ _O_   19
  <O>   _O_   <O    <O  ^ _O_   20
^ _O_   <O    <O    <O>   _O_   21
  <O    <O    <O>   _O_   <O    22
  <O    <O>   _O_ ^ _O_   <O    23
  <O>   _O_ ^ _O_   <O    <O    24
  _O_ ^ _O_   <O    <O    <O>   25
  <O  ^ _O_   <O    <O>   _O_   26
  <O>   _O_   <O>   _O_ ^ _O_   27
^ _O_   <O  ^ _O_ ^ _O_ ^ _O_   28
^ _O_   <O>   _O_ ^ _O_   <O    29
  _O_ ^ _O_ ^ _O_   <O    <O>   30
^ _O_ ^ _O_ ^ _O_   <O>   _O_   31
  <O  ^ _O_ ^ _O_   <O  ^ _O_   32
  <O>   _O_   <O    <O    <O    33
  _O_ ^ _O_   <O    <O    <O>   34
  <O    <O    <O    <O>   _O_   35
  <O    <O    <O    <O    <O    36
The diner is locked.
Meals eaten before the deadlock: [7, 5, 5, 7, 6]
```

Untuk sementara semuanya baik: para filsuf mengambil garpu, sebagian makan (`<O>`), dan mereka meletakkan garpunya kembali. Lalu tibalah gambar 36. Setiap filsuf memegang garpu kiri, tidak ada garpu di meja, dan setiap filsuf menunggu garpu kanan yang dipegang tetangganya. `main` memeriksa setengah detik lagi bahwa tidak ada yang bergerak, lalu menyatakan meja itu terkunci; kelima thread filsuf adalah daemon, sehingga program dapat berakhir walaupun mereka tidak akan pernah berakhir. Masing-masing sempat makan beberapa kali lebih dahulu.

Yang berubah-ubah: hampir segala hal tentang garis waktunya. Para filsuf berpikir selama 0 sampai 5 satuan waktu secara acak, dengan pembangkit tanpa benih, sehingga setiap jalan berbeda; dalam jalan-jalan kita, deadlock datang setelah paling sedikit 24 gambar dan paling banyak 224. Yang tidak berubah: *pada akhirnya* kelima filsuf mengambil garpu kiri mereka dalam jendela waktu pendek yang sama, dan sejak saat itu tak seorang pun makan lagi. Program ini memiliki jaring pengaman, `MAX_STEPS = 300`, kalau-kalau sebuah jalan beruntung selama tiga puluh detik penuh; tidak ada jalan kita yang seberuntung itu.

Beberapa rincian kodenya:

- `synchronized (forks[left]) { … synchronized (forks[right]) { … } }` adalah disiplin "kiri, lalu kanan". Objek-objek garpu adalah `Object` biasa yang dipakai hanya karena kunci bawaannya.

- Gambar meja juga data bersama: para filsuf menulisnya dan `main` membacanya. Karena itu `show`, `ate`, `table`, dan `mealsEaten` adalah `static synchronized`, sehingga semuanya memakai satu kunci, yaitu kunci kelas `Philosopher`. Kunci gambar selalu diambil *di dalam* kunci-kunci garpu dan tidak pernah sebaliknya; Subbab 14.12 menjelaskan mengapa konsistensi itu penting.
- `sleep(UNIT)` sambil memegang garpu kiri adalah celah yang berbahaya: ia mewakili pekerjaan apa pun yang dilakukan di antara perolehan dua kunci, dan makin panjang celahnya, makin cepat deadlock datang.

Kode 14.20 adalah meja yang sama di Python, dengan objek `threading.Lock` sebagai garpu dan blok `with` untuk "ambil ... letakkan".

Kode 14.20 `philosopher.py`: para filsuf yang bersantap di Python; setiap garpu adalah sebuah `threading.Lock`.

```
"""philosopher.py: lima filsuf, lima garpu, dan (cepat atau lambat) sebuah deadlock."""
import random
import threading
import time

N = 5            # filsuf dan garpu
UNIT = 0.1       # satu satuan waktu, dalam detik
MAX_STEPS = 300  # jaring pengaman: menyerah setelah 30 detik
forks = [threading.Lock() for _ in range(N)]  # garpu i terletak di sebelah kiri filsuf i
fork_cell = ["^"] * N
diner_cell = ["_O_"] * N
meals = [0] * N
picture = threading.Lock()  # menjaga ketiga list di atas


def show(fork, fork_text, diner, diner_text):
    with picture:
        fork_cell[fork] = fork_text
        diner_cell[diner] = diner_text


def ate(diner):
    with picture:
        meals[diner] += 1


def table():
    with picture:
        return "".join(f"{fork_cell[i]} {diner_cell[i]} " for i in range(N))


def philosopher(me):
    left, right = me, (me + 1) % N
    while True:
        time.sleep(UNIT * random.randrange(6))    # berpikir
        with forks[left]:                         # ambil garpu kiri
            show(left, " ", me, "<O ")
            time.sleep(UNIT)
            with forks[right]:                    # ambil garpu kanan
                show(right, " ", me, "<O>")
                time.sleep(UNIT)                  # makan
                ate(me)
                show(right, "^", me, "<O ")       # letakkan kembali garpu kanan
            show(left, "^", me, "_O_")            # letakkan kembali garpu kiri
```

Kode 14.20 (Lanjutan) *philosopher.py: para filsuf yang bersantap di Python; setiap garpu adalah sebuah threading.Lock.*

```
def main():
    locked = "  <O  " * N  # setiap filsuf memegang tepat satu garpu
    for i in range(N):
        # thread daemon tidak dapat membuat program tetap hidup
        threading.Thread(target=philosopher, args=(i,), daemon=True).start()
    print("The diner table:")
    for step in range(1, MAX_STEPS + 1):
        now = table()
        print(f"{now} {step:3d}")
        if now == locked:
            time.sleep(5 * UNIT)  # pastikan: tak seorang pun dapat bergerak lagi
            if table() == locked:
                print("The diner is locked.")
                with picture:
                    print("Meals eaten before the deadlock:", meals)
                return
        time.sleep(UNIT)
    print(f"No deadlock within {MAX_STEPS} steps this time.")


if __name__ == "__main__":
    main()
```

```
The diner table:
^ _O_ ^ _O_ ^ _O_   <O  ^ _O_    1
^ _O_ ^ _O_ ^ _O_   <O    <O     2
  _O_ ^ _O_ ^ _O_   <O    <O>    3
^ _O_ ^ _O_ ^ _O_ ^ _O_ ^ _O_    4
  <O    <O    <O  ^ _O_ ^ _O_    5
  <O    <O    <O>   _O_   <O     6
  <O    <O>   _O_   <O    <O     7
  <O>   _O_ ^ _O_   <O    <O     8
  _O_ ^ _O_ ^ _O_   <O    <O>    9
^ _O_   <O  ^ _O_   <O>   _O_   10
^ _O_   <O    <O  ^ _O_   <O    11
  _O_   <O    <O>   _O_   <O>   12
^ _O_   <O>   _O_   <O  ^ _O_   13
  <O  ^ _O_ ^ _O_   <O>   _O_   14
  <O>   _O_   <O  ^ _O_   <O    15
  _O_   <O    <O>   _O_   <O>   16
^ _O_   <O>   _O_ ^ _O_ ^ _O_   17
  <O  ^ _O_   <O    <O  ^ _O_   18
  <O>   _O_   <O    <O>   _O_   19
^ _O_ ^ _O_   <O>   _O_ ^ _O_   20
^ _O_ ^ _O_ ^ _O_ ^ _O_ ^ _O_   21
^ _O_   <O  ^ _O_ ^ _O_   <O    22
  _O_   <O>   _O_   <O    <O>   23
^ _O_ ^ _O_   <O    <O>   _O_   24
  <O  ^ _O_   <O>   _O_   <O    25
  <O>   _O_ ^ _O_   <O    <O    26
  _O_ ^ _O_ ^ _O_   <O    <O>   27
^ _O_ ^ _O_ ^ _O_   <O>   _O_   28
^ _O_   <O  ^ _O_   <O    <O    29
  _O_   <O>   _O_   <O    <O>   30
^ _O_ ^ _O_   <O    <O>   _O_   31
  <O  ^ _O_   <O>   _O_ ^ _O_   32
  <O>   _O_   <O    <O  ^ _O_   33
^ _O_   <O    <O    <O    <O    34
  <O    <O    <O    <O    <O    35
The diner is locked.
Meals eaten before the deadlock: [5, 5, 6, 7, 7]
```

Jalan ini terkunci pada gambar 35. Jalan yang berbeda terkunci pada saat yang sangat berbeda, dari gambar kesepuluh sampai melewati gambar keduaratus empat puluh dalam percobaan kita, tetapi setiap jalan yang kita lakukan pada akhirnya terkunci. Thread Python yang macet di `with forks[right]:` menunggu kunci tanpa batas waktu, selamanya, persis seperti thread Java yang berstatus `BLOCKED`.

14.11.2 Dua petugas dan dua mesin

Para filsuf memperlihatkan deadlock yang datang karena nasib buruk. Binatu dapat memperlihatkan deadlock yang datang karena rancangan. Dua petugas ingin memindahkan muatan dari mesin cuci ke pengering, sehingga masing-masing memerlukan *kedua* mesin. Ani mengambil mesin cuci lebih dahulu lalu pengering; Budi, yang bekerja dari ujung lain binatu, mengambil pengering lebih dahulu lalu mesin cuci. **Kode 14.21** membiarkan mereka berbuat demikian, dan memakai `CountDownLatch` untuk memastikan masing-masing sudah mengambil mesin pertamanya sebelum salah satunya meraih mesin kedua, sehingga deadlock terjadi pada setiap jalan.

Kode 14.21 `DeadlockDemo.java`: dua thread mengambil dua kunci dalam urutan berlawanan; setelah satu detik, pendeteksi deadlock milik JVM melaporkan siklusnya.

```
import java.lang.management.ManagementFactory;
import java.lang.management.ThreadInfo;
import java.lang.management.ThreadMXBean;
import java.util.concurrent.CountDownLatch;

public class DeadlockDemo {
    static final Object washer = new Object();
    static final Object dryer = new Object();
    static final CountDownLatch bothHoldOne = new CountDownLatch(2);

    // Mengambil mesin pertama, menunggu sampai petugas lain juga memegang satu, lalu
menginginkan mesin kedua.
    static void clerk(String firstName, Object first, String secondName, Object second) {
        String me = Thread.currentThread().getName();
        synchronized (first) {
            System.out.println(me + " holds the " + firstName);
            bothHoldOne.countDown();
            try {
                bothHoldOne.await();  // kini kedua petugas masing-masing memegang satu
mesin
            } catch (InterruptedException e) {
                return;
            }
            System.out.println(me + " wants the " + secondName);
            synchronized (second) {
                System.out.println(me + " moved the load");  // tidak pernah tercetak
            }
        }
    }

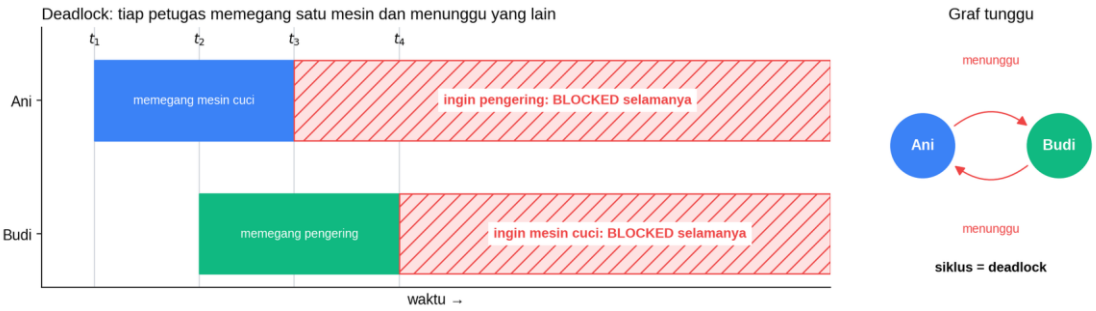
    public static void main(String[] args) throws InterruptedException {
        Thread ani = new Thread(() -> clerk("washer", washer, "dryer", dryer), "Ani");
        Thread budi = new Thread(() -> clerk("dryer", dryer, "washer", washer), "Budi");
        ani.setDaemon(true);   // agar program dapat berakhir walaupun mereka tidak akan
pernah berakhir
        budi.setDaemon(true);
        ani.start();
        budi.start();
        ani.join(1000);        // tunggu paling lama satu detik untuk setiap petugas
        budi.join(1000);
        System.out.println("after 1 s: Ani alive = " + ani.isAlive()
                + ", Budi alive = " + budi.isAlive());
```

Kode 14.21 (Lanjutan) *DeadlockDemo.java: dua thread mengambil dua kunci dalam urutan berlawanan; setelah satu detik, pendeteksi deadlock milik JVM melaporkan siklusnya.*

```
        ThreadMXBean jvm = ManagementFactory.getThreadMXBean();
        long[] ids = jvm.findDeadlockedThreads();  // JVM mencari siklus
        if (ids != null) {
            for (ThreadInfo info : jvm.getThreadInfo(ids)) {
                System.out.println("deadlocked: " + info.getThreadName() + " is " +
info.getThreadState()
                        + ", waiting for a lock held by " + info.getLockOwnerName());
            }
        }
    }
}
```

```
Ani holds the washer
Budi holds the dryer
Ani wants the dryer
Budi wants the washer
after 1 s: Ani alive = true, Budi alive = true
deadlocked: Ani is BLOCKED, waiting for a lock held by Budi
deadlocked: Budi is BLOCKED, waiting for a lock held by Ani
```

Kedua baris `holds` dapat muncul dalam urutan mana pun, demikian pula kedua baris `wants`; segala sesuatu sesudahnya sama pada setiap jalan. `CountDownLatch` adalah gerbang sekali pakai: ia mulai dengan suatu hitungan (di sini 2), `countDown()` menurunkannya, dan `await()` menunggu sampai hitungan itu mencapai nol. Setelah `bothHoldOne.await()` kembali di kedua thread, masing-masing memegang satu mesin dan menginginkan mesin yang lain. **Gambar 14.10** menggambar apa yang terjadi selanjutnya.



Gambar 14.10 Garis waktu deadlock pada **Kode 14.21**. Pada t_1 Ani mengambil mesin cuci, pada t_2 Budi mengambil pengering, pada t_3 dan t_4 masing-masing meminta mesin milik yang lain dan terblokir selamanya. Graf tunggu di sebelah kanan memiliki siklus, dan siklus dalam graf tunggu tepat merupakan sebuah deadlock.

`main` tidak dapat menunggu mereka selamanya, maka ia menunggu masing-masing dengan batas waktu satu detik lalu bertanya kepada JVM. `ManagementFactory.getThreadMXBean().findDeadlockedThreads()` membangun **graf tunggu** (*wait-for graph*) program yang sedang berjalan, dengan panah dari setiap thread yang terblokir ke thread pemilik kunci yang diinginkannya, dan mengembalikan thread-thread yang terletak pada sebuah siklus. Di sini: Ani menunggu Budi, dan Budi menunggu Ani. Dari luar program yang sedang berjalan, perintah JDK `jstack <pid>` mencetak *stack* setiap thread dan melaporkan `Found one Java-level deadlock:` beserta siklusnya.

> **DEFINISI.** Sekumpulan thread berada dalam **deadlock** bila masing-masing menunggu sumber daya yang dipegang thread lain dalam kumpulan itu, sehingga tak satu pun dapat pernah melanjutkan. Empat syarat diperlukan bagi sebuah deadlock, dan deadlock mungkin terjadi setiap kali keempatnya berlaku bersamaan [15]: (1) **saling pengecualian**: sebuah sumber daya hanya dapat dipegang satu thread; (2) **pegang dan tunggu** (*hold and wait*): sebuah thread memegang satu sumber daya sambil menunggu sumber daya lain; (3) **tanpa perampasan** (*no preemption*): sumber daya tidak dapat direbut dari pemegangnya, hanya dilepas olehnya; (4) **penantian melingkar** (*circular wait*): ada siklus thread, masing-masing menunggu sumber daya yang dipegang thread berikutnya.

Keempat *syarat Coffman* ini adalah kunci pencegahan: patahkan salah satunya dan deadlock menjadi mustahil. Pada contoh petugas, keempatnya berlaku. Sebuah mesin melayani satu petugas pada satu waktu (1); setiap petugas memegang satu mesin sambil menunggu yang lain (2); seorang petugas tidak dapat merebut mesin dari yang lain (3); dan Ani menunggu Budi yang menunggu Ani (4).

Kode 14.22 mementaskan adegan yang sama di Python. `threading.Barrier(2)` memerankan *latch*: `wait()` memblokir sampai dua thread telah tiba, lalu melepas keduanya.

Kode 14.22 `deadlock_demo.py`: kedua petugas di Python; setelah satu detik kedua thread masih hidup dan kedua kunci masih terpegang.

```
"""deadlock_demo.py: dua petugas, dua mesin, dua kunci yang diambil dalam urutan
berlawanan."""
import threading

washer = threading.Lock()
dryer = threading.Lock()
both_hold_one = threading.Barrier(2)  # memastikan terjadinya pewaktuan yang sial
print_lock = threading.Lock()


def say(text):
    with print_lock:  # print() tidak atomik: baris dari dua thread dapat bercampur
        print(text)


def clerk(first_name, first, second_name, second):
    me = threading.current_thread().name
    with first:
        say(f"{me} holds the {first_name}")
        both_hold_one.wait()          # kini kedua petugas masing-masing memegang satu
mesin
        say(f"{me} wants the {second_name}")
        with second:
            say(f"{me} moved the load")  # tidak pernah tercetak


def main():
    ani = threading.Thread(target=clerk, args=("washer", washer, "dryer", dryer),
                           name="Ani", daemon=True)
    budi = threading.Thread(target=clerk, args=("dryer", dryer, "washer", washer),
                            name="Budi", daemon=True)
    ani.start()
    budi.start()
    ani.join(1)   # tunggu paling lama satu detik untuk setiap petugas
    budi.join(1)
    print(f"after 1 s: Ani alive = {ani.is_alive()}, Budi alive = {budi.is_alive()}")
    print(f"washer locked = {washer.locked()}, dryer locked = {dryer.locked()}")
    # Thread daemon tidak membuat program tetap hidup, jadi program berakhir di sini.


if __name__ == "__main__":
    main()
```

```
Ani holds the washer
Budi holds the dryer
Budi wants the washer
Ani wants the dryer
after 1 s: Ani alive = True, Budi alive = True
washer locked = True, dryer locked = True
```

Python tidak memiliki pendeteksi deadlock bawaan, tetapi gejalanya tidak salah lagi: setelah batas waktu yang longgar kedua thread masih hidup dan kedua kunci masih terpegang. (`faulthandler.dump_traceback()` mencetak *stack* setiap thread, memperlihatkan di mana masing-masing macet.)

Listing ini juga memperagakan pelajaran konkurensi yang lebih kecil. Versi pertamanya memakai `print` biasa dari kedua thread, dan satu jalan sungguhan dari versi itu mencetak:

```
Ani holds the washerBudi holds the dryer

Ani wants the dryer
Budi wants the washer
```

`print` milik Python menulis teks dan akhir barisnya sebagai dua operasi terpisah, dan thread lain dapat mencetak di antaranya. Fungsi pembantu `say` membungkus `print` dengan sebuah kunci sehingga setiap baris keluar utuh. `System.out.println` milik Java disinkronkan secara internal dan tidak pernah mencampur dua baris dengan cara ini, walaupun baris-baris dari thread yang berbeda tentu masih dapat keluar dalam urutan apa pun.

14.12 Mencegah Deadlock

Deadlock memiliki empat syarat perlu, sehingga ada empat tempat untuk menyerangnya. Dalam praktik, dua di antaranya paling sering diserang: *penantian melingkar*, dengan pengurutan kunci, dan *pegang dan tunggu* atau *tanpa perampasan*, dengan menyerah lalu mencoba lagi.

Pengurutan kunci (*resource hierarchy* atau hierarki sumber daya dari Dijkstra). Beri nomor semua kunci, dan wajibkan setiap thread memperolehnya dalam urutan menaik. Secara formal, bila sebuah thread memegang kunci ℓ_i dan meminta kunci ℓ_j, maka

$$i < j \tag{14.5}$$

harus berlaku. Di bawah Persamaan (14.5) penantian melingkar tidak mungkin ada: sepanjang rantai thread yang menunggu, nomor kuncinya naik secara ketat, dan barisan yang naik secara ketat tidak pernah dapat kembali ke titik awalnya. Untuk para filsuf, garpu-garpu diberi nomor 0 sampai 4, dan setiap orang mengambil garpu bernomor lebih kecil lebih dahulu. Filsuf 0 sampai 3 tetap mengambil kiri lalu kanan, tetapi filsuf 4, yang garpunya 4 dan 0, mengambil garpu 0 lebih dahulu, yaitu garpu *kanan*. Satu perubahan itu memutus siklusnya. Untuk para petugas, keduanya akan mengambil mesin cuci lebih dahulu.

Mencoba, lalu mundur (mematahkan *pegang dan tunggu*). Peroleh kunci pertama; *coba* peroleh kunci kedua dengan batas waktu; bila gagal, lepaskan kunci pertama, tunggu sebentar selama waktu acak, lalu mulai lagi. Thread yang sudah mengembalikan kunci pertamanya tidak lagi memegang sambil menunggu, sehingga tidak ada siklus yang dapat bertahan. `ReentrantLock.tryLock(time, unit)` di Java dan `Lock.acquire(timeout=…)` di Python mendukung hal ini secara langsung. Jeda acak itu penting: tanpanya, dua thread yang gagal bersama akan mencoba lagi bersama dan dapat gagal bersama selamanya, yaitu livelock dari Subbab 14.13.

Seorang penengah (membatasi jumlah yang bersantap). Izinkan paling banyak $N - 1$ filsuf duduk di meja pada satu waktu, misalnya dengan **semafor** yang memegang $N - 1$ izin [16], [17]. Dengan empat filsuf dan lima garpu, paling sedikit satu filsuf yang duduk selalu dapat memperoleh kedua garpu. Pelayan yang membagikan kedua garpu sekaligus, atau tidak sama sekali, adalah bentuk lain dari penengah.

Penyelesaian Chandy dan Misra membiarkan para filsuf saling meminta garpu melalui pesan; cara ini berlaku untuk berapa pun filsuf dan sumber daya, tidak memerlukan koordinator pusat, dan menjamin tak seorang pun kelaparan [18].

Kode 14.23 menjajarkan disiplin naif itu dengan tiga obat tersebut. Setiap strategi mendudukkan lima filsuf baru dengan garpu baru, dan setiap filsuf mencoba makan tiga kali.

Kode 14.23 `DeadlockFixes.java`: filsuf kiri-lalu-kanan terkena deadlock; pengurutan kunci, `tryLock` dengan batas waktu, dan semafor empat kursi semuanya menuntaskan lima belas kali makan.

```
import java.util.concurrent.Semaphore;
import java.util.concurrent.ThreadLocalRandom;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.atomic.AtomicInteger;
import java.util.concurrent.locks.ReentrantLock;

public class DeadlockFixes {
    static final int N = 5;
    static final int MEALS = 3;  // setiap filsuf makan tiga kali, lalu pergi

    interface Diner {
        void dine(int me, ReentrantLock[] forks) throws InterruptedException;
    }

    static void pause(int maxMs) throws InterruptedException {
        Thread.sleep(ThreadLocalRandom.current().nextInt(maxMs + 1));
    }

    // Mendudukkan lima filsuf dengan garpu baru, membiarkan mereka makan, lalu melaporkan
hasilnya.
    static void run(String label, Diner diner) throws InterruptedException {
        ReentrantLock[] forks = new ReentrantLock[N];
        for (int i = 0; i < N; i++) {
            forks[i] = new ReentrantLock();
        }
        AtomicInteger meals = new AtomicInteger();
        Thread[] threads = new Thread[N];
        for (int i = 0; i < N; i++) {
            final int me = i;
            threads[i] = new Thread(() -> {
                try {
                    for (int m = 0; m < MEALS; m++) {
                        pause(20);              // berpikir
                        diner.dine(me, forks);  // ambil garpu, makan, letakkan kembali
garpu
                        meals.incrementAndGet();
                    }
                } catch (InterruptedException e) {
                    Thread.currentThread().interrupt();
                }
            });
            threads[i].setDaemon(true);         // daemon yang terkena deadlock tidak dapat
menghalangi program berakhir
            threads[i].start();
        }
        boolean finished = true;
        for (Thread t : threads) {
            t.join(2000);                       // deadlock tampak sebagai batas waktu yang
terlampaui
            finished &= !t.isAlive();
```

Kode 14.23 (Lanjutan) *DeadlockFixes.java: filsuf kiri-lalu-kanan terkena deadlock; pengurutan kunci, tryLock dengan batas waktu, dan semafor empat kursi semuanya menuntaskan lima belas kali makan.*

```
        }
        System.out.printf("%-24s: %2d of %d meals, finished = %b%n",
                label, meals.get(), N * MEALS, finished);
    }

    // Mengambil dua garpu dalam urutan yang diberikan, makan, lalu meletakkannya kembali.
    static void eatWith(ReentrantLock first, ReentrantLock second) throws
InterruptedException {
        first.lock();
        try {
            Thread.sleep(20);                   // celah berbahaya: memegang satu garpu
            second.lock();
            try {
                pause(10);                      // makan
            } finally {
                second.unlock();
            }
        } finally {
            first.unlock();
        }
    }

    public static void main(String[] args) throws InterruptedException {
        // Masalahnya: semua orang mengambil garpu kiri dahulu.
        run("0. left, then right", (me, forks) -> eatWith(forks[me], forks[(me + 1) % N]));

        // Perbaikan 1: pengurutan sumber daya. Selalu ambil garpu bernomor lebih kecil
dahulu.
        run("1. lock ordering", (me, forks) -> {
            int a = Math.min(me, (me + 1) % N);
            int b = Math.max(me, (me + 1) % N);
            eatWith(forks[a], forks[b]);
        });

        // Perbaikan 2: tryLock dengan batas waktu. Kembalikan garpu kiri bila garpu kanan
sedang dipakai.
        AtomicInteger retries = new AtomicInteger();
        run("2. tryLock with timeout", (me, forks) -> {
            ReentrantLock left = forks[me];
            ReentrantLock right = forks[(me + 1) % N];
            while (true) {
                left.lock();
                try {
                    pause(10);
                    if (right.tryLock(5, TimeUnit.MILLISECONDS)) {
                        try {
                            pause(10);          // makan
                            return;
                        } finally {
```

Kode 14.23 (Lanjutan) *DeadlockFixes.java: filsuf kiri-lalu-kanan terkena deadlock; pengurutan kunci, tryLock dengan batas waktu, dan semafor empat kursi semuanya menuntaskan lima belas kali makan.*

```
                            right.unlock();
                        }
                    }
                } finally {
                    left.unlock();              // bila gagal: letakkan juga garpu kiri
                }
                retries.incrementAndGet();
                pause(10);                      // mundur selama waktu acak, lalu coba lagi
            }
        });
        System.out.println("   (tryLock gave up and retried " + retries.get() + " times)");

        // Perbaikan 3: seorang penengah. Paling banyak N - 1 filsuf boleh duduk di meja.
        Semaphore seats = new Semaphore(N - 1);
        run("3. at most four seated", (me, forks) -> {
            seats.acquire();
            try {
                eatWith(forks[me], forks[(me + 1) % N]);
            } finally {
                seats.release();
            }
        });
    }
}
```

```
0. left, then right     :  0 of 15 meals, finished = false
1. lock ordering        : 15 of 15 meals, finished = true
2. tryLock with timeout : 15 of 15 meals, finished = true
   (tryLock gave up and retried 4 times)
3. at most four seated  : 15 of 15 meals, finished = true
```

Versi naif, `0. left, then right`, terkena deadlock sebelum sekali makan pun, dan thread-threadnya masih macet ketika program berakhir (mereka daemon, sehingga tidak membuat program tetap hidup). Celah 20 milidetik di antara kedua garpu membuat kelimanya mengambil garpu kiri hampir bersamaan; dengan celah yang lebih pendek deadlock lebih jarang terjadi, tetapi pada akhirnya selalu dapat terjadi. Ketiga obat menuntaskan kelima belas kali makan. Banyaknya percobaan ulang `tryLock` berbeda dari jalan ke jalan (dalam jalan-jalan kita, antara 2 dan 7); lima belas kali makan dan `finished = true` tidak.

Tabel 14.3 menghubungkan setiap obat dengan syarat yang dipatahkannya.

Tabel 14.3 Cara-cara mencegah deadlock, dan syarat Coffman yang dipatahkan masing-masing.

Strategi	Syarat Coffman yang dipatahkan	Java	Python
pengurutan kunci	penantian melingkar	peroleh dalam urutan global yang tetap	sama
coba dengan batas waktu, lalu mundur	pegang dan tunggu, tanpa perampasan	`tryLock(t, unit)`	`acquire(timeout=t)`
penengah (paling banyak $N-1$)	penantian melingkar (siklusnya tidak dapat tertutup)	`Semaphore(N - 1)`	`threading.Semaphore(N - 1)`
satu kunci untuk semuanya	pegang dan tunggu (hanya ada satu kunci)	satu objek `synchronized`	satu `Lock`
tanpa sumber daya bersama	saling pengecualian	pesan melalui `BlockingQueue`	pesan melalui `queue.Queue`

Kode 14.24 adalah percobaan yang sama di Python.

Kode 14.24 `deadlock_fixes.py`: filsuf-filsuf naif dan tiga obat di Python.

```
"""deadlock_fixes.py: filsuf-filsuf naif, dan tiga cara mencegah deadlock."""
import random
import threading
import time

N = 5
MEALS = 3  # setiap filsuf makan tiga kali, lalu pergi


def pause(max_ms):
    time.sleep(random.randint(0, max_ms) / 1000)


def run(label, dine):
    """Mendudukkan lima filsuf dengan garpu baru, membiarkan mereka makan, lalu melapor."""
    forks = [threading.Lock() for _ in range(N)]
    meals = [0]
    count_lock = threading.Lock()

    def philosopher(me):
        for _ in range(MEALS):
            pause(20)                # berpikir
            dine(me, forks)          # ambil garpu, makan, letakkan kembali garpu
            with count_lock:
                meals[0] += 1

    threads = [threading.Thread(target=philosopher, args=(i,), daemon=True)
               for i in range(N)]
    for t in threads:
        t.start()
    finished = True
    for t in threads:
        t.join(2)                    # deadlock tampak sebagai batas waktu yang terlampaui
        finished = finished and not t.is_alive()
    print(f"{label:<24}: {meals[0]:2d} of {N * MEALS} meals, finished = {finished}")


def eat_with(first, second):
    with first:
        time.sleep(0.02)             # celah berbahaya: memegang satu garpu
        with second:
            pause(10)                # makan
```

Kode 14.24 (Lanjutan) *deadlock_fixes.py: filsuf-filsuf naif dan tiga obat di Python.*

```
def naive(me, forks):
    eat_with(forks[me], forks[(me + 1) % N])


def ordered(me, forks):
    a, b = sorted((me, (me + 1) % N))  # garpu bernomor lebih kecil dahulu
    eat_with(forks[a], forks[b])


retries = [0]
retry_lock = threading.Lock()


def polite(me, forks):
    left, right = forks[me], forks[(me + 1) % N]
    while True:
        with left:
            pause(10)
            if right.acquire(timeout=0.005):  # tunggu paling lama 5 ms
                try:
                    pause(10)       # makan
                    return
                finally:
                    right.release()
        # bila gagal: blok with sudah meletakkan juga garpu kiri
        with retry_lock:            # += pada data bersama juga memerlukan kunci
            retries[0] += 1
        pause(10)                   # mundur selama waktu acak, lalu coba lagi


seats = threading.Semaphore(N - 1)  # paling banyak empat filsuf di meja


def seated(me, forks):
    with seats:
        eat_with(forks[me], forks[(me + 1) % N])


def main():
    run("0. left, then right", naive)
    run("1. lock ordering", ordered)
    run("2. acquire with timeout", polite)
    print(f"   (acquire gave up and retried {retries[0]} times)")
    run("3. at most four seated", seated)
```

Kode 14.24 (Lanjutan) *deadlock_fixes.py: filsuf-filsuf naif dan tiga obat di Python.*

```
if __name__ == "__main__":
    main()
```

```
0. left, then right     :  0 of 15 meals, finished = False
1. lock ordering        : 15 of 15 meals, finished = True
2. acquire with timeout : 15 of 15 meals, finished = True
   (acquire gave up and retried 6 times)
3. at most four seated  : 15 of 15 meals, finished = True
```

Hasilnya sesuai dengan Java. Dua hal kecil dalam kode Python patut diperhatikan. `sorted((me, (me + 1) % N))` adalah pengurutan kunci dalam satu baris. Dan pencacah percobaan ulang dinaikkan di bawah `retry_lock`-nya sendiri: `retries[0] += 1` persis baca-ubah-tulis dari Subbab 14.7, dan lima filsuf dapat mencoba ulang pada saat yang sama.

> **WAWASAN KUNCI:** Pengurutan kunci adalah kuda beban. Ia tidak memakan biaya saat program berjalan, tidak memerlukan batas waktu, dan dapat diperiksa dengan membaca kode: setiap kali sebuah metode memegang satu kunci dan meminta kunci lain, kunci kedua harus datang belakangan dalam urutan yang disepakati. Bila kunci-kunci itu milik objek dari kelas yang sama (dua dompet, dua mesin), beri setiap objek nomor unik dan selalu kunci nomor yang lebih kecil lebih dahulu. Soal Latihan 14.8 melakukan persis hal itu.

14.13 Livelock dan Starvation

Thread yang terkena deadlock tidak berbuat apa-apa. Ada dua cara lain bagi thread untuk gagal maju sambil tampak sibuk.

14.13.1 Livelock

Anda mungkin pernah mengalami livelock di sebuah lorong. Anda melangkah ke kiri untuk memberi jalan kepada seseorang; pada saat yang sama ia dengan sopan melangkah ke kanan *miliknya*, sehingga ia kembali berada di depan Anda. Anda berdua melangkah ke arah sebaliknya, lagi, dan lagi. Tak seorang pun terblokir, semua orang bergerak, dan tak seorang pun sampai ke mana-mana.

> **DEFINISI. Livelock** adalah keadaan ketika thread-thread terus mengubah keadaannya sebagai tanggapan terhadap satu sama lain, tanpa terblokir, tetapi tak satu pun maju.

Kode 14.25, diadaptasi dari sebuah contoh yang terkenal [19], membangun lorong itu dari dua pekerja yang terlalu sopan dan satu sumber daya bersama. Setiap pekerja ingin memakai sumber daya itu sekali. Seorang pekerja hanya memakainya bila ia pemiliknya; tetapi bila ia pemiliknya dan melihat bahwa pekerja *lain* masih aktif, ia dengan sopan menyerahkan sumber daya itu dan menunggu gilirannya. Bila kedua pekerja mulai dalam keadaan aktif, masing-masing menyerahkan sumber daya kepada yang lain selamanya. Agar peragaan ini terbatas, setiap pekerja juga memiliki batas `politeness` (kesopanan) dan bendera `stopped` yang dipasang `main` setelah 100 ms.

Kode 14.25 `Livelock.java`: dua pekerja yang selalu mengalah mengoper sebuah sumber daya bolak-balik tanpa pernah memakainya; membatasi kesopanan mereka menjadi satu kali penyerahan mengakhiri livelock itu.

```
public class Livelock {
    static class CommonResource {
        private Worker owner;

        CommonResource(Worker d) {
            owner = d;
        }

        synchronized Worker getOwner() {
            return owner;
        }

        synchronized void setOwner(Worker d) {
            owner = d;
        }
    }

    static class Worker {
        private final String name;
        private final int politeness;     // berapa kali pekerja ini mau mengalah
        private volatile boolean active;  // masih punya pekerjaan
        private volatile boolean stopped; // disuruh menyerah dari luar
        private int handovers = 0;
        private int done = 0;

        Worker(String name, boolean active, int politeness) {
            this.name = name;
            this.active = active;
            this.politeness = politeness;
        }

        String getName() {
            return name;
        }

        boolean isActive() {
            return active;
        }
```

Kode 14.25 (Lanjutan) *Livelock.java: dua pekerja yang selalu mengalah mengoper sebuah sumber daya bolak-balik tanpa pernah memakainya; membatasi kesopanan mereka menjadi satu kali penyerahan mengakhiri livelock itu.*

```
        synchronized void work(CommonResource commonResource, Worker otherWorker) {
            while (active && !stopped) {
                // Tunggu sampai sumber daya tersedia.
                if (commonResource.getOwner() != this) {
                    try {
                        wait(10);
                    } catch (InterruptedException e) {
                        return;
                    }
                    continue;
                }
                // Bila pekerja lain juga aktif, biarkan ia bekerja lebih dahulu.
                if (otherWorker.isActive() && handovers < politeness) {
                    System.out.println(getName() + " : handover the resource to the worker
"
                            + otherWorker.getName());
                    handovers++;
                    commonResource.setOwner(otherWorker);
                    continue;
                }
                // Sekarang pakai sumber daya bersama.
                System.out.println(getName() + " : working on the common resource");
                done++;
                active = false;
                commonResource.setOwner(otherWorker);
            }
        }

        synchronized String report() {
            return name + " handed over " + handovers + " times and did " + done + "
job(s)";
        }
    }

    static void scene(String title, int politeness) throws InterruptedException {
        System.out.println(title);
        Worker worker1 = new Worker("Worker 1", true, politeness);
        Worker worker2 = new Worker("Worker 2", true, politeness);
        CommonResource s = new CommonResource(worker1);
        Thread t1 = new Thread(() -> worker1.work(s, worker2));
        Thread t2 = new Thread(() -> worker2.work(s, worker1));
        t1.start();
        t2.start();
        Thread.sleep(100);            // beri mereka 100 ms
        worker1.stopped = true;       // lalu suruh keduanya menyerah
        worker2.stopped = true;
        t1.join();
        t2.join();
```

Kode 14.25 (Lanjutan) *Livelock.java: dua pekerja yang selalu mengalah mengoper sebuah sumber daya bolak-balik tanpa pernah memakainya; membatasi kesopanan mereka menjadi satu kali penyerahan mengakhiri livelock itu.*

```
        System.out.println("  " + worker1.report());
        System.out.println("  " + worker2.report());
    }

    public static void main(String[] args) throws InterruptedException {
        scene("Scene 1: both workers are always polite", Integer.MAX_VALUE);
        scene("Scene 2: each worker gives way at most once", 1);
    }
}
```

```
Scene 1: both workers are always polite
Worker 1 : handover the resource to the worker Worker 2
Worker 2 : handover the resource to the worker Worker 1
Worker 1 : handover the resource to the worker Worker 2
Worker 2 : handover the resource to the worker Worker 1
Worker 1 : handover the resource to the worker Worker 2
Worker 2 : handover the resource to the worker Worker 1
Worker 1 : handover the resource to the worker Worker 2
Worker 2 : handover the resource to the worker Worker 1
Worker 1 : handover the resource to the worker Worker 2
Worker 2 : handover the resource to the worker Worker 1
Worker 1 : handover the resource to the worker Worker 2
Worker 2 : handover the resource to the worker Worker 1
Worker 1 : handover the resource to the worker Worker 2
Worker 2 : handover the resource to the worker Worker 1
Worker 1 : handover the resource to the worker Worker 2
Worker 2 : handover the resource to the worker Worker 1
  Worker 1 handed over 8 times and did 0 job(s)
  Worker 2 handed over 8 times and did 0 job(s)
Scene 2: each worker gives way at most once
Worker 1 : handover the resource to the worker Worker 2
Worker 2 : handover the resource to the worker Worker 1
Worker 1 : working on the common resource
Worker 2 : working on the common resource
  Worker 1 handed over 1 times and did 1 job(s)
  Worker 2 handed over 1 times and did 1 job(s)
```

Pada Adegan 1, dengan kesopanan tanpa batas, sumber daya itu berpindah bolak-balik delapan kali untuk masing-masing dalam 100 ms, dan tidak satu pekerjaan pun selesai; banyaknya penyerahan berbeda antarjalan (kita melihat 16 sampai 19 baris), dan tanpa bendera `stopped` jumlahnya akan terus bertambah selama program berjalan. Perhatikan bahwa thread-thread itu tidak pernah `BLOCKED`: mereka sibuk, bangun setiap 10 ms, memeriksa, menyerahkan. Itulah perbedaannya dengan deadlock, dan itulah sebabnya livelock lebih sulit dikenali: alat pemantau melihat thread yang hidup dan memakan waktu prosesor. Pada Adegan 2 setiap pekerja mengalah paling banyak sekali, dan sesudahnya ia memakai sumber daya itu walaupun yang lain masih aktif; kedua pekerjaan selesai setelah dua penyerahan, dan adegan ini mencetak baris-baris yang sama pada setiap jalan. **Gambar 14.11** menggambar kedua adegan.



Gambar 14.11 Livelock dan obatnya. Atas: dua pekerja yang selalu sopan mengoper sumber daya bolak-balik dan tidak pernah bekerja. Bawah: setiap pekerja mengalah paling banyak sekali, sehingga setelah dua penyerahan keduanya bekerja.

Perhatikan bahwa `active` dan `stopped` adalah `volatile` (setiap pekerja membaca bendera `active` milik yang lain, dan `main` menulis kedua bendera `stopped`) dan bahwa `CommonResource` menjaga

`owner`-nya dengan metode synchronized; `wait(10)` hanyalah jeda sopan 10 ms yang melepas kunci milik pekerja itu sendiri. Program ini sama sekali tidak memiliki *data race*, dan tetap saja tidak maju: livelock adalah kegagalan *rancangan*, bukan kegagalan sinkronisasi.

Livelock diobati dengan mematahkan kesimetrisan: batasi kesopanannya, seperti Adegan 2; beri para pekerja prioritas yang berbeda (misalnya, hanya pekerja bernomor lebih besar yang mengalah); atau, seperti filsuf `tryLock` pada **Kode 14.23**, mundur selama waktu *acak*, sehingga dua thread yang pernah bertabrakan kecil kemungkinannya bertabrakan lagi. Obat yang paling sederhana sering kali adalah membiarkan satu thread mengerjakan semuanya secara berurutan: bila kedua pekerjaan memang harus memakai sumber daya yang sama satu demi satu, menjalankannya di dua thread tidak memberi keuntungan apa pun. Seperti halnya deadlock, tidak ada aturan umum yang mencegah livelock; yang menolong adalah kecermatan setiap kali sebuah thread mengubah keadaan bersama *sebagai tanggapan* terhadap apa yang dilakukan thread lain.

Kode 14.26 adalah lorong yang sama di Python.

Kode 14.26 `livelock.py`: dua pekerja yang terlalu sopan di Python.

```
"""livelock.py: dua pekerja yang terlalu sopan terus saling menyerahkan sumber daya."""
import sys
import threading
import time


class CommonResource:
    def __init__(self, owner):
        self._owner = owner
        self._lock = threading.Lock()

    def get_owner(self):
        with self._lock:
            return self._owner

    def set_owner(self, owner):
        with self._lock:
            self._owner = owner


class Worker:
    def __init__(self, name, active, politeness):
        self.name = name
        self.active = active          # masih punya pekerjaan
        self.politeness = politeness  # berapa kali pekerja ini mau mengalah
        self.stopped = False          # disuruh menyerah dari luar
        self.handovers = 0
        self.done = 0

    def work(self, common_resource, other_worker):
        while self.active and not self.stopped:
            # Tunggu sampai sumber daya tersedia.
            if common_resource.get_owner() is not self:
                time.sleep(0.01)
                continue
            # Bila pekerja lain juga aktif, biarkan ia bekerja lebih dahulu.
            if other_worker.active and self.handovers < self.politeness:
                print(f"{self.name} : handover the resource to the worker
{other_worker.name}")
                self.handovers += 1
                common_resource.set_owner(other_worker)
                continue
            # Sekarang pakai sumber daya bersama.
            print(f"{self.name} : working on the common resource")
            self.done += 1
            self.active = False
            common_resource.set_owner(other_worker)
```

Kode 14.26 (Lanjutan) *livelock.py: dua pekerja yang terlalu sopan di Python.*

```
    def report(self):
        return f"{self.name} handed over {self.handovers} times and did {self.done} job(s)"


def scene(title, politeness):
    print(title)
    worker1 = Worker("Worker 1", True, politeness)
    worker2 = Worker("Worker 2", True, politeness)
    s = CommonResource(worker1)
    t1 = threading.Thread(target=worker1.work, args=(s, worker2))
    t2 = threading.Thread(target=worker2.work, args=(s, worker1))
    t1.start()
    t2.start()
    time.sleep(0.1)          # beri mereka 100 ms
    worker1.stopped = True   # lalu suruh keduanya menyerah
    worker2.stopped = True
    t1.join()
    t2.join()
    print("  " + worker1.report())
    print("  " + worker2.report())


def main():
    scene("Scene 1: both workers are always polite", sys.maxsize)
    scene("Scene 2: each worker gives way at most once", 1)


if __name__ == "__main__":
    main()
```

```
Scene 1: both workers are always polite
Worker 1 : handover the resource to the worker Worker 2
Worker 2 : handover the resource to the worker Worker 1
Worker 1 : handover the resource to the worker Worker 2
Worker 2 : handover the resource to the worker Worker 1
Worker 1 : handover the resource to the worker Worker 2
Worker 2 : handover the resource to the worker Worker 1
Worker 1 : handover the resource to the worker Worker 2
Worker 2 : handover the resource to the worker Worker 1
Worker 1 : handover the resource to the worker Worker 2
Worker 2 : handover the resource to the worker Worker 1
Worker 1 : handover the resource to the worker Worker 2
Worker 2 : handover the resource to the worker Worker 1
Worker 1 : handover the resource to the worker Worker 2
Worker 2 : handover the resource to the worker Worker 1
Worker 1 : handover the resource to the worker Worker 2
Worker 2 : handover the resource to the worker Worker 1
Worker 1 : handover the resource to the worker Worker 2
Worker 2 : handover the resource to the worker Worker 1
Worker 1 : handover the resource to the worker Worker 2
Worker 2 : handover the resource to the worker Worker 1
  Worker 1 handed over 10 times and did 0 job(s)
  Worker 2 handed over 10 times and did 0 job(s)
Scene 2: each worker gives way at most once
Worker 1 : handover the resource to the worker Worker 2
Worker 2 : handover the resource to the worker Worker 1
Worker 1 : working on the common resource
Worker 2 : working on the common resource
  Worker 1 handed over 1 times and did 1 job(s)
  Worker 2 handed over 1 times and did 1 job(s)
```

Para pekerja Python memeriksa berulang kali dengan `time.sleep(0.01)` alih-alih `wait(10)`, dan pemiliknya dilindungi sebuah kunci di dalam `CommonResource`. Dalam jalan ini mereka

menyerahkan sumber daya sepuluh kali masing-masing dalam 100 ms sebelum `main` menghentikan mereka; adegan yang sudah diperbaiki lagi-lagi identik pada setiap jalan.

14.13.2 Starvation dan keadilan

Starvation (kelaparan) adalah nasib thread yang sebenarnya dapat maju tetapi tidak pernah mendapat kesempatan, karena thread lain selalu memperoleh sumber daya lebih dahulu. Seorang filsuf yang kedua tetangganya begitu rakus sehingga salah satunya selalu sedang makan akan kelaparan, walaupun tidak ada deadlock: meja secara keseluruhan tetap maju. Thread berprioritas rendah pada sistem yang sibuk dapat kelaparan; demikian pula thread yang hendak menulis dan menunggu kunci yang terus-menerus direbut oleh arus pembaca.

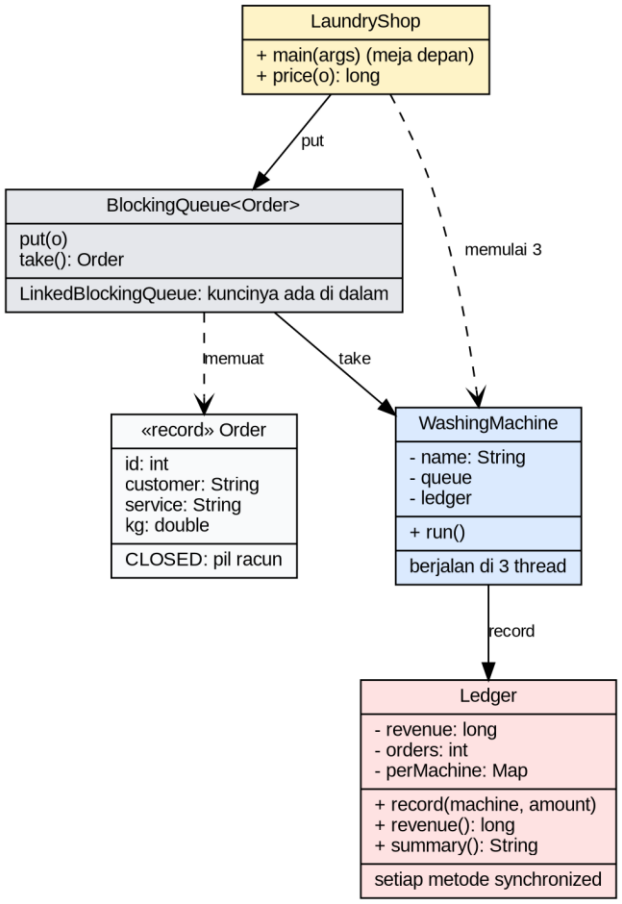
Pertahanan bakunya adalah **keadilan** (*fairness*): layani thread yang menunggu menurut urutan kedatangannya. `synchronized` di Java sama sekali tidak menjanjikan keadilan: ketika sebuah kunci dilepas, thread mana pun yang menunggu dapat memperolehnya, termasuk yang datang paling akhir. `new ReentrantLock(true)` dan `new Semaphore(n, true)` bersifat *adil*: thread yang paling lama menunggu mendapat giliran berikutnya. Kunci yang adil lebih lambat, jadi pakailah di tempat yang risiko kelaparannya nyata, bukan sebagai kebiasaan. `threading.Lock` milik Python juga tidak menjanjikan keadilan; sebaliknya, `queue.Queue` membagikan butir-butirnya dengan urutan masuk pertama, keluar pertama, satu alasan lagi untuk mengutamakan antrean. Penyelesaian Chandy–Misra dari Subbab 14.12 bebas kelaparan menurut konstruksinya [18].

14.14 Profitina Laundry: Mesin sebagai Thread Pekerja

Sekarang kita dapat membangun binatu pada **Gambar 14.1**. Kebutuhannya:

- Meja depan menerima pesanan hari itu. Setiap pesanan memiliki nomor, pelanggan, layanan, dan berat. Layanan dan harganya adalah yang dari Bab 7: **Wash & Fold** (`WF`) seharga Rp7,000 per kilogram dan **Wash, Dry & Iron** (`WDI`) seharga Rp10,000 per kilogram, keduanya dihitung paling sedikit 3 kg dan dibulatkan ke atas ke setengah kilogram berikutnya.
- Tiga mesin cuci, `W1`, `W2`, dan `W3`, bekerja bersamaan. Setiap mesin mengambil pesanan berikutnya yang menunggu begitu ia kosong, mencucinya (dalam simulasi kita, 20 ms per kilogram), dan mencatat harganya di kasir.
- Ada satu buku kas milik kasir. Pada akhir hari buku itu harus memuat pendapatan yang tepat, jumlah pesanan yang tepat, dan jumlah pesanan yang dicuci setiap mesin, bagaimanapun pekerjaan mesin-mesin itu berselang-seling.
- Pada jam tutup setiap mesin harus mematikan dirinya sendiri, dan program harus berakhir.

Setiap alat yang kita perlukan sudah tersedia di meja: **antrean pemblokir** untuk pesanan (Subbab 14.10), **thread pekerja** yang berputar pada `take()` (Subbab 14.4), **buku kas synchronized** untuk total bersama (Subbab 14.8), dan **pil racun** agar mesin-mesin tahu kapan berhenti. **Gambar 14.12** adalah rancangannya.



Gambar 14.12 Diagram kelas binatu konkuren. Meja depan (`main`) memasukkan record `Order` ke sebuah `BlockingQueue`; tiga objek `WashingMachine`, masing-masing dijalankan oleh thread-nya sendiri, mengambil pesanan darinya dan mencatat harganya di satu-satunya `Ledger`, yang seluruhnya tersinkronisasi.

Kode 14.27 adalah programnya.

Kode 14.27 `LaundryShop.java`: tiga thread mesin cuci mengambil pesanan dari `LinkedBlockingQueue` dan membayar ke `Ledger` yang tersinkronisasi; pil racun mematikan mereka.

```
import java.util.ArrayList;
import java.util.List;
import java.util.Locale;
import java.util.Map;
import java.util.TreeMap;
import java.util.concurrent.BlockingQueue;
import java.util.concurrent.LinkedBlockingQueue;

public class LaundryShop {
    record Order(int id, String customer, String service, double kg) { }

    static final Order CLOSED = new Order(0, "-", "-", 0);  // pil racun: "binatu sudah
tutup"
    static final int MS_PER_KG = 20;                         // waktu cuci simulasi

    // Aturan Bab 7: paling sedikit 3 kg, dibulatkan ke atas ke setengah kilogram
berikutnya.
    static long price(Order o) {
        long rate = o.service().equals("WF") ? 7000 : 10000;  // WF atau WDI, per kg
        double chargeable = Math.ceil(2 * Math.max(o.kg(), 3.0)) / 2;
        return Math.round(rate * chargeable);
    }

    // Kasir: satu buku kas bersama untuk semua mesin. Hanya satu mesin setiap saat yang
boleh menulis.
    static class Ledger {
        private long revenue = 0;
        private int orders = 0;
        private final Map<String, Integer> perMachine = new TreeMap<>();

        synchronized void record(String machine, long amount) {
            revenue += amount;
            orders++;
            perMachine.merge(machine, 1, Integer::sum);
        }

        synchronized long revenue() {
            return revenue;
        }

        synchronized String summary() {
            return orders + " orders, revenue Rp" + String.format(Locale.US, "%,d",
revenue)
                    + ", per machine " + perMachine;
        }
    }
```

Kode 14.27 (Lanjutan) *LaundryShop.java: tiga thread mesin cuci mengambil pesanan dari LinkedBlockingQueue dan membayar ke Ledger yang tersinkronisasi; pil racun mematikan mereka.*

```
    static class WashingMachine implements Runnable {
        private final String name;
        private final BlockingQueue<Order> queue;
        private final Ledger ledger;

        WashingMachine(String name, BlockingQueue<Order> queue, Ledger ledger) {
            this.name = name;
            this.queue = queue;
            this.ledger = ledger;
        }

        @Override
        public void run() {
            try {
                while (true) {
                    Order o = queue.take();          // tunggu pesanan berikutnya
                    if (o == CLOSED) {
                        break;
                    }
                    System.out.printf(Locale.US, "%s washes order %2d: %-7s %4.1f kg %s%n",
                            name, o.id(), o.customer(), o.kg(), o.service());
                    Thread.sleep((long) (o.kg() * MS_PER_KG));
                    ledger.record(name, price(o));   // bayar di satu-satunya kasir
                }
            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
            }
            System.out.println(name + " switched off");
        }
    }

    public static void main(String[] args) throws InterruptedException {
        List<Order> today = List.of(
                new Order(1, "Ayu", "WF", 4.2), new Order(2, "Bagus", "WDI", 6.0),
                new Order(3, "Citra", "WF", 2.5), new Order(4, "Dewi", "WF", 7.8),
                new Order(5, "Eko", "WDI", 3.4), new Order(6, "Fitri", "WF", 5.1),
                new Order(7, "Gilang", "WDI", 8.0), new Order(8, "Hana", "WF", 3.0),
                new Order(9, "Indra", "WF", 6.6), new Order(10, "Joko", "WDI", 4.4),
                new Order(11, "Kartika", "WF", 9.3), new Order(12, "Lestari", "WDI", 2.0));
```

Kode 14.27 (Lanjutan) *LaundryShop.java: tiga thread mesin cuci mengambil pesanan dari LinkedBlockingQueue dan membayar ke Ledger yang tersinkronisasi; pil racun mematikan mereka.*

```
        BlockingQueue<Order> queue = new LinkedBlockingQueue<>();
        Ledger ledger = new Ledger();
        List<Thread> machines = new ArrayList<>();
        for (String name : new String[] {"W1", "W2", "W3"}) {
            Thread t = new Thread(new WashingMachine(name, queue, ledger), name);
            machines.add(t);
            t.start();
        }
        long t0 = System.nanoTime();
        for (Order o : today) {
            queue.put(o);                  // meja depan menerima pesanan
        }
        for (int i = 0; i < machines.size(); i++) {
            queue.put(CLOSED);             // satu pil per mesin
        }
        for (Thread t : machines) {
            t.join(10_000);                // jangan pernah menunggu selamanya
        }
        long wallMs = (System.nanoTime() - t0) / 1_000_000;

        long expected = 0;
        double totalKg = 0;
        for (Order o : today) {
            expected += price(o);          // jumlah yang sama, dihitung oleh satu thread
            totalKg += o.kg();
        }
        System.out.println("Ledger: " + ledger.summary());
        System.out.println("Ledger matches the one-thread total: " + (ledger.revenue() ==
expected));
        System.out.printf(Locale.US, "Washing time %d ms in total, done in about %d ms%n",
                Math.round(totalKg * MS_PER_KG), wallMs);
    }
}
```

Satu jalan yang sungguhan mencetak:

```
W1 washes order  1: Ayu      4.2 kg WF
W3 washes order  3: Citra    2.5 kg WF
W2 washes order  2: Bagus    6.0 kg WDI
W3 washes order  4: Dewi     7.8 kg WF
W1 washes order  5: Eko      3.4 kg WDI
W2 washes order  6: Fitri    5.1 kg WF
W1 washes order  7: Gilang   8.0 kg WDI
W3 washes order  8: Hana     3.0 kg WF
W2 washes order  9: Indra    6.6 kg WF
W3 washes order 10: Joko     4.4 kg WDI
W1 washes order 11: Kartika  9.3 kg WF
W2 washes order 12: Lestari  2.0 kg WDI
W3 switched off
W2 switched off
W1 switched off
Ledger: 12 orders, revenue Rp533,500, per machine {W1=4, W2=4, W3=4}
Ledger matches the one-thread total: true
Washing time 1246 ms in total, done in about 532 ms
```

dan jalan lain, sesaat kemudian:

```
W1 washes order  2: Bagus    6.0 kg WDI
W3 washes order  1: Ayu      4.2 kg WF
W2 washes order  3: Citra    2.5 kg WF
W2 washes order  4: Dewi     7.8 kg WF
W3 washes order  5: Eko      3.4 kg WDI
W1 washes order  6: Fitri    5.1 kg WF
W3 washes order  7: Gilang   8.0 kg WDI
W2 washes order  8: Hana     3.0 kg WF
W1 washes order  9: Indra    6.6 kg WF
W2 washes order 10: Joko     4.4 kg WDI
W3 washes order 11: Kartika  9.3 kg WF
W2 washes order 12: Lestari  2.0 kg WDI
W1 switched off
W2 switched off
W3 switched off
Ledger: 12 orders, revenue Rp533,500, per machine {W1=3, W2=5, W3=4}
Ledger matches the one-thread total: true
Washing time 1246 ms in total, done in about 532 ms
```

Bandingkan kedua jalan itu baris demi baris, karena keduanya memperlihatkan dengan tepat apa yang boleh dan tidak boleh diubah oleh program konkuren dari satu jalan ke jalan berikutnya.

Yang berubah. Mesin mana yang mengambil pesanan mana: pada jalan pertama `W1` mengambil pesanan 1 dan `W3` pesanan 3; pada jalan kedua, `W1` mengambil pesanan 2 dan `W2` mengambil pesanan 3 sekaligus pesanan 4. Urutan baris-baris pertama, yang bergantung pada thread mesin mana yang kebetulan lebih dahulu mencapai `take()`. Sebaran pekerjaan: empat pesanan per mesin pada jalan pertama, tetapi 3, 5, dan 4 pada jalan kedua. Urutan mesin-mesin dimatikan. Dan waktu yang berlalu: 532 ms pada kedua jalan ini, tetapi antara 530 ms dan 561 ms pada enam jalan lainnya, dan berbeda lagi di komputer Anda.

Yang tidak berubah, dan tidak boleh berubah: kedua belas pesanan dicuci, masing-masing tepat sekali; pendapatannya Rp533,500; jumlah pesanan di buku kas 12; dan buku kas cocok dengan total yang dihitung satu thread dari daftar yang sama. Anda dapat memeriksa pendapatan itu dengan tangan menggunakan aturan Bab 7. Pesanan 1 adalah 4.2 kg Wash & Fold, dihitung sebagai 4.5 kg, $4.5 \times 7000 = 31{,}500$; pesanan 3 (2.5 kg) dan pesanan 8 (3.0 kg) sama-sama dihitung sebagai minimum 3 kg, masing-masing 21,000; pesanan 12 adalah 2.0 kg Wash, Dry & Iron, dihitung sebagai 3 kg, 30,000; dan seterusnya untuk kedua belas pesanan, yang berjumlah 533,500.

Sekarang keputusan-keputusan rancangannya, satu per satu.

- `record Order(...)` adalah record imutabel: begitu meja depan menciptakan sebuah pesanan, tidak ada thread yang dapat mengubahnya. Objek imutabel dapat dioper antarthread dengan bebas (Bab 13); antrean menyerahkan setiap pesanan kepada tepat satu mesin, dan tidak seorang pun perlu menguncinya.
- `CLOSED` adalah pil racun, sebuah objek `Order` biasa yang dikenali mesin-mesin melalui *identitasnya* (`o == CLOSED`), bukan isinya. `main` memasukkan satu pil per mesin setelah pesanan sungguhan terakhir. Karena antreannya masuk pertama, keluar pertama, setiap pesanan sungguhan diambil sebelum pil mana pun.
- `LinkedBlockingQueue` mengurus semua penantian. Mesin yang memanggil `take()` pada antrean kosong menunggu (`WAITING` pada **Gambar 14.4**) dan bangun begitu sebuah pesanan tiba. Tidak ada mesin yang dapat mengambil pesanan yang sudah diambil mesin lain: jaminan itu adalah tugas antrean, dan kunci internalnya yang menyediakannya.
- `Ledger` adalah satu-satunya kasir. Ketiga metodenya `synchronized`, sehingga mesin yang sedang mencatat pembayaran memiliki buku kas itu untuk dirinya sendiri, dan `summary()` serta

`revenue()` melihat setiap pembayaran yang dicatat sebelumnya. Hapus kata `synchronized` dari `record`, dan tiga mesin yang menambah `revenue` pada saat yang sama dapat kehilangan pembayaran persis seperti **Kode 14.11** kehilangan kenaikan; Soal Latihan 14.3 meminta Anda mencobanya.

- **Pencucian terjadi di luar kunci.** `Thread.sleep` (pencucian) berada di `run()`, bukan di dalam `record()`. Bila sebuah mesin memegang kunci buku kas selama mencuci, mesin lain tidak dapat membayar, dan ketiga mesin akan berjalan satu per satu. Jaga agar bagian kritis tetap pendek: kerjakan pekerjaannya dahulu, lalu kunci, perbarui, lepas.
- `t.join(10_000)` menunggu setiap mesin, tetapi tidak pernah selamanya. Bila suatu *bug* membuat sebuah mesin macet, program tetap berakhir, dan jumlah pesanan di buku kas akan mengungkap masalahnya.
- **Pemeriksaan** `ledger.revenue() == expected` membandingkan hasil konkuren dengan perhitungan berurutan atas hal yang sama, cara yang sederhana dan ampuh untuk menguji kode konkuren.

Berapa lama seharusnya satu hari berlangsung? Mesin-mesin itu mencuci 62.3 kg seluruhnya, yang dengan 20 ms per kilogram berarti 1,246 ms pencucian. Dengan m mesin dan waktu cuci $t_1, \ldots, t_k$, tidak ada jadwal yang dapat selesai lebih cepat daripada

$$T_{\text{selesai}} \geq \max\left(\frac{1}{m}\sum_{i=1}^{k} t_i\,,\ \max_i t_i\right) \tag{14.6}$$

karena ketiga mesin bersama-sama tidak dapat mencuci lebih cepat daripada tiga kali satu mesin, dan pesanan tunggal yang terpanjang tidak dapat dipecah. Di sini Persamaan (14.6) memberikan max(1246/3,186) ≈ 415 ms. Mesin-mesin kita memakai aturan sederhana "ambil pesanan berikutnya ketika kosong", dan menelusuri aturan itu dengan tangan memberikan 498 ms (pesanan 11 yang panjang, 9.3 kg, baru dimulai pada 312 ms, di mesin yang paling dahulu kosong); waktu terukur sekitar 530 ms menambahkan biaya memulai thread dan bangun dari `sleep`. Jadwal yang lebih baik ada: binatu yang mendahulukan pesanan terpanjang akan selesai lebih dekat ke batas itu. Itu pertanyaan penjadwalan, bukan konkurensi, tetapi ia memperlihatkan mengapa "tiga mesin" tidak pernah tepat berarti "tiga kali lebih cepat", sejalan dengan Persamaan (14.1).

Kode 14.28 adalah binatu itu dalam Python.

Kode 14.28 `laundry_shop.py`: binatu konkuren dalam Python: `WashingMachine` adalah subkelas `threading.Thread`, buku kas menjaga totalnya dengan sebuah `Lock`, dan `None` adalah pil racunnya.

```
"""laundry_shop.py: tiga mesin cuci mengambil pesanan dari satu antrean dan membayar ke
satu kasir."""
import math
import queue
import threading
import time
from dataclasses import dataclass


@dataclass(frozen=True)
class Order:
    id: int
    customer: str
    service: str
    kg: float


CLOSED = None      # pil racun: "binatu sudah tutup"
MS_PER_KG = 20     # waktu cuci simulasi
print_lock = threading.Lock()


def say(text):
    with print_lock:  # print() tidak atomik: baris dari dua thread dapat bercampur
        print(text)


def price(o):
    """Aturan Bab 7: paling sedikit 3 kg, dibulatkan ke atas ke setengah kilogram
berikutnya."""
    rate = 7000 if o.service == "WF" else 10000  # WF atau WDI, per kg
    chargeable = math.ceil(2 * max(o.kg, 3.0)) / 2
    return round(rate * chargeable)


class Ledger:
    """Kasir: satu buku kas bersama untuk semua mesin. Hanya satu mesin setiap saat yang
boleh menulis."""

    def __init__(self):
        self._lock = threading.Lock()
        self._revenue = 0
        self._orders = 0
        self._per_machine = {}
```

Kode 14.28 (Lanjutan) *laundry_shop.py: binatu konkuren dalam Python: WashingMachine adalah subkelas threading.Thread, buku kas menjaga totalnya dengan sebuah Lock, dan None adalah pil racunnya.*

```
    def record(self, machine, amount):
        with self._lock:
            self._revenue += amount
            self._orders += 1
            self._per_machine[machine] = self._per_machine.get(machine, 0) + 1

    def revenue(self):
        with self._lock:
            return self._revenue

    def summary(self):
        with self._lock:
            per = dict(sorted(self._per_machine.items()))
            return f"{self._orders} orders, revenue Rp{self._revenue:,}, per machine {per}"


class WashingMachine(threading.Thread):
    def __init__(self, name, orders, ledger):
        super().__init__(name=name)
        self.orders = orders
        self.ledger = ledger

    def run(self):
        while True:
            o = self.orders.get()           # tunggu pesanan berikutnya
            if o is CLOSED:
                break
            say(f"{self.name} washes order {o.id:2d}: {o.customer:<7} {o.kg:4.1f} kg
{o.service}")
            time.sleep(o.kg * MS_PER_KG / 1000)
            self.ledger.record(self.name, price(o))  # bayar di satu-satunya kasir
        say(f"{self.name} switched off")


def main():
    today = [
        Order(1, "Ayu", "WF", 4.2), Order(2, "Bagus", "WDI", 6.0),
        Order(3, "Citra", "WF", 2.5), Order(4, "Dewi", "WF", 7.8),
        Order(5, "Eko", "WDI", 3.4), Order(6, "Fitri", "WF", 5.1),
        Order(7, "Gilang", "WDI", 8.0), Order(8, "Hana", "WF", 3.0),
        Order(9, "Indra", "WF", 6.6), Order(10, "Joko", "WDI", 4.4),
        Order(11, "Kartika", "WF", 9.3), Order(12, "Lestari", "WDI", 2.0)]
```

Kode 14.28 (Lanjutan) *laundry_shop.py: binatu konkuren dalam Python: WashingMachine adalah subkelas threading.Thread, buku kas menjaga totalnya dengan sebuah Lock, dan None adalah pil racunnya.*

```
    orders = queue.Queue()
    ledger = Ledger()
    machines = [WashingMachine(name, orders, ledger) for name in ("W1", "W2", "W3")]
    for m in machines:
        m.start()
    t0 = time.perf_counter()
    for o in today:
        orders.put(o)                       # meja depan menerima pesanan
    for _ in machines:
        orders.put(CLOSED)                  # satu pil per mesin
    for m in machines:
        m.join(10)                          # jangan pernah menunggu selamanya
    wall_ms = (time.perf_counter() - t0) * 1000

    expected = sum(price(o) for o in today)  # jumlah yang sama, dihitung oleh satu thread
    total_kg = sum(o.kg for o in today)
    print("Ledger:", ledger.summary())
    print("Ledger matches the one-thread total:", ledger.revenue() == expected)
    print(f"Washing time {round(total_kg * MS_PER_KG)} ms in total, done in about
{wall_ms:.0f} ms")


if __name__ == "__main__":
    main()
```

```
W3 washes order  1: Ayu      4.2 kg WF
W2 washes order  2: Bagus    6.0 kg WDI
W1 washes order  3: Citra    2.5 kg WF
W1 washes order  4: Dewi     7.8 kg WF
W3 washes order  5: Eko      3.4 kg WDI
W2 washes order  6: Fitri    5.1 kg WF
W3 washes order  7: Gilang   8.0 kg WDI
W1 washes order  8: Hana     3.0 kg WF
W2 washes order  9: Indra    6.6 kg WF
W1 washes order 10: Joko     4.4 kg WDI
W3 washes order 11: Kartika  9.3 kg WF
W2 washes order 12: Lestari  2.0 kg WDI
W1 switched off
W2 switched off
W3 switched off
Ledger: 12 orders, revenue Rp533,500, per machine {'W1': 4, 'W2': 4, 'W3': 4}
Ledger matches the one-thread total: True
Washing time 1246 ms in total, done in about 499 ms
```

Rancangannya sama, kelas demi kelas. `@dataclass(frozen=True)` membuat `Order` imutabel, seperti yang dilakukan `record` di Java. `WashingMachine` adalah subkelas `threading.Thread`, yang cocok untuk pekerja berumur panjang dengan namanya sendiri dan referensinya sendiri ke antrean dan buku kas. Ketiga metode buku kas masing-masing memegang `self._lock`. Mesin-mesin mencetak melalui `say()`, karena alasan yang terlihat di Subbab 14.11.2. Dan karena mencuci adalah *menunggu* (`time.sleep`), GIL dilepas selama setiap pencucian, sehingga ketiga mesin Python bertumpang tindih sebaik mesin Java: sekitar 500 ms untuk 1,246 ms pencucian. Seandainya mesin-mesin itu *menghitung* dalam Python murni, Subbab 14.6 memberi tahu kita bahwa ketiga thread akan bergiliran, dan `ProcessPoolExecutor` akan menjadi alat yang tepat.

14.15 Profitina dalam Praktik

Profitina adalah platform visibilitas AI: ia memindai situs web sebuah merek dan mengukur seberapa siap situs itu dikutip oleh mesin penjawab AI, disiplin yang disebut *generative engine*

optimization (GEO). Aplikasi webnya ditulis dalam Python dengan FastAPI dan dilayani oleh Uvicorn, dan hampir setiap gagasan dalam bab ini muncul di dalamnya dalam salah satu bentuk.

Sebuah permintaan adalah pelanggan di meja depan. Selagi satu permintaan menunggu basis data, permintaan lain dilayani: konkurensi yang terikat I/O, yang menurut Subbab 14.6 ditangani dengan baik oleh thread Python (dan oleh tugas asinkron, kerabat dekatnya).

Penjadwal adalah produsen, pemindai adalah konsumen. Profitina memindai ulang setiap merek secara otomatis dengan irama yang ditetapkan oleh tingkat langganannya. Memutuskan *apa* yang sudah jatuh tempo dan benar-benar *melakukan* pemindaian dipisahkan oleh sebuah antrean, dalam bentuk **Kode 14.27**: penjadwal menghasilkan pekerjaan, para pekerja mengonsumsinya, dan situs yang lambat hanya menunda pekerja yang sedang memindainya.

Model bahasa lokal adalah satu-satunya kasir. Profitina menjalankan fitur AI-nya pada model Qwen2.5 lokal melalui Ollama, sehingga data pengguna tidak pernah meninggalkan server. Satu model hanya dapat menjawab sejumlah permintaan pada satu waktu, sehingga permintaan menunggu gilirannya dalam antrean yang tertib; dan ketika tidak ada model yang tersedia, aplikasi mengatakannya alih-alih berpura-pura.

Pencacah yang tidak boleh kehilangan pembaruan. Pembatas laju *login* mencacah percobaan yang gagal, dan koin utilitas PROFIT menyimpan saldo dengan batas perolehan harian: pencacah bersama yang diperbarui oleh banyak permintaan konkuren, persis keadaan **Kode 14.11**. Nilai-nilai seperti itu tinggal di basis data PostgreSQL, yang transaksi dan kunci barisnya memainkan peran `synchronized`. Pelajarannya berlaku persis: setiap nilai yang dapat diubah oleh lebih dari satu permintaan memerlukan satu penjaga.

Pengujian sebagai gerbang. Race tidak muncul sesuai permintaan, sehingga sistem konkuren bersandar pada disiplin: bagian kritis yang kecil, pesan yang imutabel, antrean alih-alih list bersama, dan rangkaian pengujian otomatis yang besar yang harus lulus sebelum setiap rilis. Seperti biasa di Profitina, perangkat lunak mengukur dan menjelaskan; pengguna yang memutuskan.

14.16 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya

1. Memanggil `run()` alih-alih `start()`. Kodenya berjalan, di thread saat ini, tanpa konkurensi sama sekali (**Kode 14.5**). *Hindari:* `start()` menciptakan thread; `run()` hanyalah apa yang akan dikerjakan thread itu. Di Python, berikan `target=f`, bukan `target=f()`.

2. Memperbarui data bersama tanpa kunci. `count++`, `total += price`, `list.add(x)` pada `ArrayList` bersama: masing-masing adalah baca-ubah-tulis yang kehilangan pembaruan di bawah konkurensi (**Kode 14.11**). *Hindari:* untuk setiap field yang dapat diubah lebih dari satu thread, sebutkan kunci yang menjaganya, dan pegang kunci itu untuk *setiap* operasi baca dan operasi tulis.

3. Menjaga operasi tulis tetapi tidak operasi baca. Pembaca tanpa kunci dapat melihat nilai yang basi atau setengah diperbarui. *Hindari:* jadikan getter juga `synchronized` (seperti di `Ledger`), atau pakai variabel `volatile`/atomik untuk nilai tunggal.

4. Mengira `volatile` membuat `++` atomik. Ia membuat setiap operasi baca dan setiap operasi tulis terlihat; ia tidak menyatukan keduanya (**Kode 14.11** memakai field volatile dan tetap kehilangan pembaruan). *Hindari:* pakai `AtomicInteger`, `LongAdder`, atau kunci untuk pencacah.

5. Mengira GIL membuat kode Python aman-thread. GIL melindungi interpreter, bukan invarian Anda; metode yang terdiri atas beberapa *bytecode* dapat diinterupsi di antara dua mana pun (**Kode 14.12**). *Hindari:* kunci data bersama di Python persis seperti di Java.

6. Memakai `if` alih-alih `while` di sekitar `wait()`. Thread yang terbangun dapat mendapati kondisinya kembali salah, atau terbangun secara palsu. *Hindari:* selalu `while (!condition) wait();` di Java, dan `while not condition: cond.wait()` (atau `cond.wait_for(predicate)`) di Python.

7. Memanggil `wait`/`notify` tanpa memegang kunci. Java melempar `IllegalMonitorStateException`; Python melempar `RuntimeError: cannot wait on un-acquired lock`. *Hindari:* panggil keduanya hanya di dalam `synchronized (obj)` pada `obj` yang sama, atau di dalam `with cond:`.

8. Memperoleh kunci dalam urutan yang berbeda-beda. Dua thread yang mengambil dua kunci dalam urutan berlawanan cepat atau lambat akan terkena deadlock (**Kode 14.21**). *Hindari:* tetapkan urutan global (Persamaan (14.5)) dan dokumentasikan; kunci menurut nomor unik bila kunci-kunci itu milik objek dari satu kelas.

9. Memegang kunci sambil mengerjakan pekerjaan yang lambat. Mencuci, tidur, membaca berkas, atau memanggil layanan jaringan di dalam bagian kritis membuat setiap thread lain menunggu, dan memanggil kode yang tidak dikenal (sebuah *listener*, metode yang dapat di-*override*) sambil memegang kunci mengundang deadlock. *Hindari:* hitung dahulu, lalu kunci, perbarui, dan lepas.

10. Menelan `InterruptedException`. `catch` yang kosong membuat thread mustahil dihentikan. *Hindari:* akhiri thread itu, atau paling tidak pulihkan benderanya dengan `Thread.currentThread().interrupt()`.

11. Menunggu selamanya. `join()`, `take()`, dan `acquire()` tanpa batas menggantung seluruh program bila ada yang salah. *Hindari:* dalam peragaan dan dalam kode produksi yang harus tetap tanggap, pakai versi dengan batas waktu, dan perlakukan batas waktu yang terlampaui sebagai tanda untuk menyelidiki.

12. Mencetak dari beberapa thread Python tanpa kunci. Baris-baris dapat menyatu (Subbab 14.11.2). *Hindari:* pakai kunci cetak, modul `logging` (yang mengunci secara internal), atau kirim pesan melalui antrean ke satu thread pencetak.

13. Memercayai pengujian konkurensi yang lulus. Race dapat muncul sekali dalam sejuta jalan (Persamaan (14.2)). *Hindari:* utamakan rancangan (imutabilitas, antrean, data yang terkurung di satu thread) yang membuat race mustahil, bukan sekadar jarang.

14.17 Ringkasan Bab

- **Thread** adalah urutan instruksi mandiri di dalam sebuah proses; thread memiliki *stack* sendiri tetapi berbagi *heap*. **Konkurensi** berarti beberapa tugas sedang berlangsung; **paralelisme** berarti beberapa tugas dieksekusi pada detik yang sama di inti yang berbeda. **Penjadwal** menentukan selang-selingnya, berbeda pada setiap jalan.
- Ciptakan thread dengan subkelas `Thread`, sebuah `Runnable` atau lambda, atau sebuah `ExecutorService` (Java), dan dengan subkelas `threading.Thread`, fungsi target, atau `ThreadPoolExecutor` (Python). Panggil `start()`, jangan `run()`; tunggu dengan `join()`. Hentikan thread secara kooperatif: `interrupt()` di Java, `threading.Event` di Python, atau bendera `volatile`.

- Thread Java berada dalam keadaan `NEW`, `RUNNABLE`, `BLOCKED`, `WAITING`, `TIMED_WAITING`, atau `TERMINATED` (**Gambar 14.4**); Python menyediakan `is_alive()`.
- **GIL** milik CPython membuat hanya satu thread yang mengeksekusi *bytecode* Python pada satu waktu: thread membantu pekerjaan yang **terikat I/O**, bukan pekerjaan **terikat CPU** dalam Python murni, yang untuknya dipakai proses. Percepatan dibatasi oleh **hukum Amdahl** (Persamaan (14.1)). Python *free-threaded* (eksperimental di 3.13, didukung di 3.14) menghapus GIL.
- **Race condition** membuat hasil bergantung pada pewaktuan; kasus klasiknya adalah **pembaruan yang hilang** pada baca-ubah-tulis seperti `c++` atau `+=` (**Gambar 14.6**). Dari $\binom{m+n}{m}$ selang-seling (Persamaan (14.2)) mungkin hanya sedikit yang benar, sehingga pengujian tidak dapat membuktikan bahwa race tidak ada.
- **Saling pengecualian** mengobati race: metode dan blok `synchronized`, `ReentrantLock`, `threading.Lock`; variabel tunggal dapat memakai **atomik** (`AtomicInteger`, `LongAdder`). Kunci juga menjamin **visibilitas**; `volatile` hanya menjamin visibilitas.
- Thread menunggu kondisi dengan `wait`/`notifyAll` atau `Condition`, selalu di dalam perulangan `while`; **monitor** adalah kunci ditambah ruang tunggu. `BlockingQueue` dan `queue.Queue` mengimplementasikan pola **produsen–konsumen** untuk Anda; **pil racun** mengakhiri para konsumen.
- **Deadlock** memerlukan saling pengecualian, pegang dan tunggu, tanpa perampasan, dan penantian melingkar; siklus dalam **graf tunggu** adalah deadlock. Cegahlah dengan **pengurutan kunci** (Persamaan (14.5)), **coba dengan batas waktu dan mundur secara acak**, **penengah** seperti semafor, atau dengan tidak berbagi (**Tabel 14.3**).
- **Livelock** adalah kesibukan tanpa kemajuan, diobati dengan mematahkan kesimetrisan (batas, prioritas, mundur secara acak). **Starvation** adalah thread yang tidak pernah mendapat giliran, diobati dengan **keadilan**.
- Di binatu, tiga thread mesin mengambil pesanan dari satu antrean dan membayar ke satu buku kas yang tersinkronisasi; mesin mana mencuci apa berubah antarjalan, pendapatan Rp533,500 tidak pernah berubah.

14.18 Soal Latihan

Soal Latihan 14.1 (thread atau bukan?). Untuk setiap tugas, tentukan apakah menjalankannya dalam beberapa thread akan membuatnya lebih cepat di Java, dan di CPython dengan GIL, beserta alasannya: (a) mengunduh 50 halaman web; (b) mencacah bilangan prima di bawah sepuluh juta dalam kode murni; (c) memampatkan 50 berkas dengan pustaka kompresi yang ditulis dalam C; (d) menjaga jendela Tkinter tetap tanggap selagi sebuah laporan dihitung; (e) menjumlahkan daftar berisi dua belas harga.

Soal Latihan 14.2 (mencacah selang-seling). Dua thread masing-masing mengeksekusi `c++` sekali, sebagai tiga langkah baca, tambah, tulis. (a) Pakai Persamaan (14.2) untuk mencacah selang-seling yang mungkin dan daftarlah yang menyisakan $c = 2$. (b) Tiga thread masing-masing mengeksekusi `c++` sekali. Ada berapa selang-seling, dan berapa yang benar? (c) Apa artinya bagi pengujian?

Soal Latihan 14.3 (rusakkan kasirnya). Hapus kata `synchronized` dari `Ledger.record` di **Kode 14.27** dan jalankan program itu dua puluh kali. Laporkan apa yang Anda lihat, jelaskan, dan jelaskan mengapa program itu tetap salah. Sebutkan dua hal yang dapat keliru.

Soal Latihan 14.4 (keadaan). Apa yang dicetak setiap `println`, dan mengapa?

```
Object lock = new Object();
Thread t = new Thread(() -> {
    synchronized (lock) { }
});
System.out.println(t.getState());          // (1)
synchronized (lock) {
    t.start();
    Thread.sleep(200);
    System.out.println(t.getState());      // (2)
}
t.join();
System.out.println(t.getState());          // (3)
```

Soal Latihan 14.5 (`if` alih-alih `while`). Misalkan `getSharedInt` di **Kode 14.15** memakai `if (!readable) wait();` alih-alih `while`, dan ada *dua* thread konsumen. Uraikan, langkah demi langkah, sebuah selang-seling yang membuat seorang konsumen membaca nilai yang tidak ada.

Soal Latihan 14.6 (Amdahl). Sebuah pembuat laporan menghabiskan 80% waktunya untuk pekerjaan yang terbagi sempurna ke beberapa inti. Pakai Persamaan (14.1) untuk menghitung percepatan terbaik pada 2, 4, dan 8 inti, serta limitnya bila banyaknya inti bertambah tanpa batas. Berapa inti yang memberikan 90% dari limit itu?

Soal Latihan 14.7 (mengapa pengurutan berhasil). Buktikan bahwa filsuf-filsuf dengan pengurutan kunci pada **Kode 14.23** tidak mungkin terkena deadlock, dengan memakai Persamaan (14.5). Lalu tunjukkan bahwa cukup *satu* filsuf yang mengambil garpu kanan lebih dahulu.

Soal Latihan 14.8 (dua dompet). Profitina Laundry menjual dompet prabayar, dan pelanggan boleh saling mentransfer uang. `transfer(from, to, amount)` harus mengunci kedua dompet. Tulislah dalam Java dan Python sehingga satu thread yang berulang kali mentransfer dari Ayu ke Budi dan thread lain dari Budi ke Ayu tidak pernah dapat terkena deadlock, dan tunjukkan bahwa tidak ada uang yang tercipta atau musnah.

Soal Latihan 14.9 (kilat lebih dahulu). Perluas binatu itu sehingga pesanan kilat dicuci sebelum pesanan biasa, dan pesanan sejenis menurut urutan kedatangan. Pakai `PriorityBlockingQueue` di Java dan `queue.PriorityQueue` di Python, dengan satu mesin, dan tunjukkan urutan pencucian lima pesanan (pesanan 2 dan 4 kilat).

Soal Latihan 14.10 (livelock di dalam obatnya). Pada filsuf `tryLock` di **Kode 14.23**, filsuf yang gagal memperoleh garpu kanan meletakkan garpu kiri dan menunggu selama waktu *acak* sebelum mencoba lagi. Uraikan apa yang dapat terjadi bila setiap filsuf menunggu tepat 10 ms, dan mengapa keacakan memperbaikinya.

Soal Latihan 14.11 (deadlock atau bukan?). Untuk setiap keadaan, tentukan apakah deadlock mungkin terjadi, dan sebutkan syarat Coffman yang tidak terpenuhi bila tidak mungkin. (a) Dua thread, masing-masing memakai satu kunci, tidak pernah memegang dua sekaligus. (b) Thread 1 memanggil metode synchronized `a.f()`, yang memanggil metode synchronized `b.g()`; thread 2 memanggil `b.h()`, yang memanggil `a.k()`; keempat metode synchronized. (c) Satu thread Python mengeksekusi `with lock:` dua kali, bersarang, pada `threading.Lock` yang sama. (d) Sama, tetapi dengan `synchronized (obj)` Java yang bersarang dua kali pada objek yang sama. (e) Thread A memanggil `b.join()` dan thread B memanggil `a.join()`.

14.19 Penyelesaian Soal Latihan

Penyelesaian 14.1. (a) Lebih cepat di keduanya: mengunduh berarti menunggu jaringan, dan thread yang menunggu (di Python, thread yang telah melepas GIL) membiarkan thread lain mengunduh. (b) Lebih cepat di Java dengan beberapa inti; *tidak* lebih cepat di CPython dengan GIL, karena hanya satu thread yang mengeksekusi *bytecode* Python pada satu waktu; pakailah proses di Python. (c) Lebih cepat di keduanya: pustaka seperti itu mengerjakan tugasnya dalam C dan lazimnya melepas GIL selama memampatkan. (d) Ya di keduanya, tetapi demi ketanggapan, bukan kecepatan: laporan berjalan di thread pekerja sementara thread milik jendela terus menangani peristiwa (dan hanya thread itu yang boleh menyentuh widget, Bab 10). (e) Tidak: dua belas penjumlahan memakan waktu jauh lebih singkat daripada menciptakan satu thread.

Penyelesaian 14.2. (a) Dengan $m = n = 3$, Persamaan (14.2) memberikan $\binom{6}{3} = 20$ selang-seling. Dengan menuliskan langkah-langkah A sebagai $a_1a_2a_3$ dan langkah-langkah B sebagai $b_1b_2b_3$, hasilnya 2 hanya bila operasi baca satu thread terjadi setelah operasi tulis thread lain, yaitu untuk $a_1a_2a_3b_1b_2b_3$ dan $b_1b_2b_3a_1a_2a_3$: 2 benar, 18 kehilangan satu pembaruan dan menyisakan $c = 1$. (b) Tiga barisan tiga langkah dapat digabung dengan $\frac{9!}{3!\,3!\,3!} = 1680$ cara; hanya $3! = 6$ cara, yang membuat ketiga kenaikan berjalan satu demi satu tanpa bertumpang tindih, yang benar. (c) Hampir semua selang-seling salah, tetapi penjadwal biasanya memilih yang tidak berbahaya karena setiap thread berjalan selama satu irisan waktu utuh sekaligus; pengujian hanya melihat selang-seling buruk yang langka itu karena kebetulan. Pengujian dapat mengungkap race; pengujian tidak dapat membuktikan ketiadaannya.

Penyelesaian 14.3. Dalam dua puluh jalan program yang sudah diubah itu, kedua puluhnya mencetak `Ledger matches the one-thread total: true`. Itu bukan bukti kebenaran. Setiap mesin memanggil `record` hanya empat kali, berselang sekitar 100 ms, dan satu pemanggilan memakan waktu jauh di bawah satu mikrodetik, sehingga peluang dua pemanggilan bertumpang tindih sangat kecil, sama seperti race yang kehilangan satu pembaruan dalam sejuta operasi lolos dari setiap pengujian singkat. Program itu salah karena dua mesin *dapat* memanggil `record` pada saat yang sama. Dua hal lalu dapat keliru: (1) `revenue += amount` dan `orders++` adalah baca-ubah-tulis dan dapat kehilangan satu pembayaran atau satu pesanan, persis seperti pada **Kode 14.11**; (2) `perMachine.merge` mengubah sebuah `TreeMap`, yang sama sekali tidak aman-thread; dua perubahan serentak dapat merusak pohon internalnya, menghilangkan entri, atau melempar `ConcurrentModificationException` di `summary()`. Selain itu, tanpa `synchronized` pada `record`, `summary()` yang synchronized tidak lagi menjamin bahwa ia melihat semua operasi tulis. Untuk membuat *bug* itu terlihat, buat mesin-mesin mencatat banyak pembayaran berturut-turut dengan cepat, misalnya seribu per pesanan tanpa tidur di antaranya.

Penyelesaian 14.4. (1) `NEW`: thread-nya sudah ada tetapi belum dimulai. (2) `BLOCKED`: ia sudah dimulai dan sedang mencoba memasuki `synchronized (lock)`, yang dipegang `main` selama 200 ms penuh. (3) `TERMINATED`: begitu `main` meninggalkan bloknya, `t` masuk, mendapati badan yang kosong, lalu berakhir; `join()` menunggu hal itu.

Penyelesaian 14.5. Misalkan penyangga kosong dan konsumen C1 dan C2 sama-sama menunggu di dalam `getSharedInt` (masing-masing melewati `if`-nya, mendapati `readable` bernilai false, lalu memanggil `wait()`). Produsen menyimpan satu nilai, memasang `readable = true`, dan memanggil `notifyAll()`, membangunkan keduanya. C1 memperoleh kembali kunci lebih dahulu, mengambil nilai itu, mendapati penyangga kosong dan memasang `readable = false`, lalu kembali. C2 kini memperoleh kembali kunci. Dengan `if`, ia *tidak* menguji `readable` lagi: ia melanjutkan setelah `wait()`, "membaca" sel di `readLocation`, yang berisi nilai lama yang sudah

dikonsumsi, dan memindahkan `readLocation` melewati `writeLocation` milik produsen. Sejak itu indeks-indeksnya tidak konsisten dan penyangga mengembalikan nilai basi selamanya. Dengan `while`, C2 akan menguji `readable`, mendapatinya false, dan menunggu lagi.

Penyelesaian 14.6. Dengan $f = 0.8$: $S(2) = 1/(0.2 + 0.4) = 1.67$; $S(4) = 1/(0.2 + 0.2) = 2.5$; $S(8) = 1/(0.2 + 0.1) = 3.33$; dan $S(p) \to 1/0.2 = 5$ bila $p \to \infty$. Sembilan puluh persen dari limit itu adalah 4.5: $1/(0.2 + 0.8/p) = 4.5$ memberikan $0.2 + 0.8/p = 0.2222$, sehingga $0.8/p = 0.0222$ dan $p = 36$. Tiga puluh enam inti mencapai 4.5; setengah satuan percepatan terakhir akan memerlukan inti yang tak terhingga banyaknya.

Penyelesaian 14.7. Beri nomor garpu-garpu 0 sampai 4. Dengan pengurutan, filsuf yang memegang garpu i hanya meminta garpu j dengan $i < j$ (Persamaan (14.5)). Andaikan ada deadlock: siklus filsuf $P_1, \ldots, P_k$, masing-masing memegang satu garpu dan menunggu garpu yang dipegang filsuf berikutnya, dengan P_k menunggu garpu milik P_1. Misalkan x_t garpu yang dipegang P_t. P_t menunggu x_{t+1} sambil memegang x_t, sehingga $x_t < x_{t+1}$ untuk setiap t, dan P_k menunggu x_1 sambil memegang x_k, sehingga $x_k < x_1$. Maka $x_1 < x_2 < \cdots < x_k < x_1$, dan itu mustahil. Jadi tidak ada deadlock. Pada versi berurutan di **Kode 14.23**, filsuf 0 sampai 3 mengambil $\min(i, i + 1) = i$ lebih dahulu, yaitu garpu kiri mereka; hanya filsuf 4, dengan garpu 4 dan 0, yang mengambil garpu 0 lebih dahulu, yaitu garpu kanan. Jadi aturan "semua kiri-dahulu kecuali satu kanan-dahulu" tepat merupakan pengurutan kunci, dan satu pengecualian sudah cukup untuk memutus siklusnya.

Penyelesaian 14.8. **Kode 14.29** memberi setiap dompet `id` yang unik dan selalu mengunci dompet ber-id lebih kecil lebih dahulu, ke arah mana pun uang mengalir, sehingga kedua arah mengambil kunci dalam urutan yang sama dan Persamaan (14.5) berlaku.

Kode 14.29 `WalletTransfer.java`: transfer ke dua arah antara dua dompet, dibuat bebas deadlock dengan mengunci id yang lebih kecil lebih dahulu.

```
import java.util.concurrent.locks.ReentrantLock;

public class WalletTransfer {
    static class Wallet {
        final int id;                    // nomor unik menetapkan urutan penguncian
        final ReentrantLock lock = new ReentrantLock();
        private long balance;

        Wallet(int id, long balance) {
            this.id = id;
            this.balance = balance;
        }

        long balance() {
            lock.lock();
            try {
                return balance;
            } finally {
                lock.unlock();
            }
        }
    }

    // Selalu kunci dompet ber-id lebih kecil dahulu, ke arah mana pun uang mengalir.
    static void transfer(Wallet from, Wallet to, long amount) {
        Wallet first = from.id < to.id ? from : to;
        Wallet second = from.id < to.id ? to : from;
        first.lock.lock();
        try {
            second.lock.lock();
            try {
                if (from.balance >= amount) {
                    from.balance -= amount;
                    to.balance += amount;
                }
            } finally {
                second.lock.unlock();
            }
        } finally {
            first.lock.unlock();
        }
    }
```

Kode 14.29 (Lanjutan) *WalletTransfer.java: transfer ke dua arah antara dua dompet, dibuat bebas deadlock dengan mengunci id yang lebih kecil lebih dahulu.*

```
    public static void main(String[] args) throws InterruptedException {
        Wallet ayu = new Wallet(1, 500_000);
        Wallet budi = new Wallet(2, 500_000);
        final int times = 100_000;
        Thread one = new Thread(() -> {
            for (int i = 0; i < times; i++) {
                transfer(ayu, budi, 7_000);
            }
        });
        Thread two = new Thread(() -> {
            for (int i = 0; i < times; i++) {
                transfer(budi, ayu, 7_000);  // arah sebaliknya
            }
        });
        one.setDaemon(true);
        two.setDaemon(true);
        one.start();
        two.start();
        one.join(10_000);
        two.join(10_000);
        boolean finished = !one.isAlive() && !two.isAlive();
        System.out.println("finished without deadlock: " + finished);
        System.out.println("money in the system: Rp" + (ayu.balance() + budi.balance()));
    }
}
```

```
finished without deadlock: true
money in the system: Rp1000000
```

Setiap thread melakukan 100,000 transfer sebesar Rp7,000. Pemeriksaan saldo `from.balance >= amount` berada di dalam kedua kunci, sehingga sebuah dompet tidak pernah bersaldo negatif, dan totalnya tetap tepat Rp1,000,000: setiap rupiah yang meninggalkan satu dompet tiba di dompet lain di dalam bagian kritis yang sama. Bila Anda mengganti `first`/`second` dengan `from`/`to`, program itu terkena deadlock: pada empat dari lima jalan versi itu yang kita lakukan, kedua thread macet dan `finished without deadlock: false` muncul setelah dua kali `join` sepuluh detik. **Kode 14.30** adalah versi Python-nya; `with first.lock, second.lock:` memperoleh kedua kunci dalam urutan yang tertulis.

Kode 14.30 `wallet_transfer.py`: transfer bebas deadlock yang sama dalam Python.

```python
"""wallet_transfer.py: transfer ke dua arah tanpa deadlock, dengan pengurutan kunci."""
import threading


class Wallet:
    def __init__(self, wallet_id, balance):
        self.id = wallet_id              # nomor unik menetapkan urutan penguncian
        self.lock = threading.Lock()
        self._balance = balance

    def balance(self):
        with self.lock:
            return self._balance


def transfer(source, target, amount):
    # Selalu kunci dompet ber-id lebih kecil dahulu, ke arah mana pun uang mengalir.
    first, second = sorted((source, target), key=lambda w: w.id)
    with first.lock, second.lock:
        if source._balance >= amount:
            source._balance -= amount
            target._balance += amount


def main():
    ayu = Wallet(1, 500_000)
    budi = Wallet(2, 500_000)
    times = 100_000

    def ayu_pays():
        for _ in range(times):
            transfer(ayu, budi, 7_000)

    def budi_pays():
        for _ in range(times):
            transfer(budi, ayu, 7_000)   # arah sebaliknya
```

Kode 14.30 (Lanjutan) *wallet_transfer.py: transfer bebas deadlock yang sama dalam Python.*

```python
    one = threading.Thread(target=ayu_pays, daemon=True)
    two = threading.Thread(target=budi_pays, daemon=True)
    one.start()
    two.start()
    one.join(10)
    two.join(10)
    finished = not one.is_alive() and not two.is_alive()
    print("finished without deadlock:", finished)
    print(f"money in the system: Rp{ayu.balance() + budi.balance()}")


if __name__ == "__main__":
    main()
```

```
finished without deadlock: True
money in the system: Rp1000000
```

Penyelesaian 14.9. Antrean prioritas menyerahkan elemen *terkecil* lebih dahulu. **Kode 14.31** membuat `Order` dapat dibandingkan sehingga pesanan kilat lebih kecil daripada pesanan biasa dan, di antara yang setara, id yang lebih awal lebih kecil. Pil racun memiliki id terbesar yang mungkin dan bukan pesanan kilat, sehingga terurut paling akhir.

Kode 14.31 `ExpressQueue.java`: `PriorityBlockingQueue` membiarkan pesanan kilat menyalip pesanan biasa.

```java
import java.util.concurrent.PriorityBlockingQueue;

public class ExpressQueue {
    record Order(int id, String customer, boolean express) implements Comparable<Order> {
        @Override
        public int compareTo(Order other) {
            if (express != other.express) {
                return express ? -1 : 1;             // pesanan kilat lebih dahulu
            }
            return Integer.compare(id, other.id);    // lalu yang datang lebih dahulu
dilayani lebih dahulu
        }
    }

    static final Order CLOSED = new Order(Integer.MAX_VALUE, "-", false);  // terurut
paling akhir

    public static void main(String[] args) throws InterruptedException {
        PriorityBlockingQueue<Order> queue = new PriorityBlockingQueue<>();
        queue.put(new Order(1, "Ayu", false));
        queue.put(new Order(2, "Bagus", true));
        queue.put(new Order(3, "Citra", false));
        queue.put(new Order(4, "Dewi", true));
        queue.put(new Order(5, "Eko", false));
        queue.put(CLOSED);
        Thread machine = new Thread(() -> {
            try {
                while (true) {
                    Order o = queue.take();
                    if (o == CLOSED) {
                        break;
                    }
                    System.out.println("W1 washes order " + o.id() + " (" + o.customer()
                            + (o.express() ? ", express)" : ")"));
                }
            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
            }
        }, "W1");
        machine.start();
        machine.join();
    }
}
```

```
W1 washes order 2 (Bagus, express)
W1 washes order 4 (Dewi, express)
W1 washes order 1 (Ayu)
W1 washes order 3 (Citra)
W1 washes order 5 (Eko)
```

Karena semua pesanan sudah berada di antrean sebelum mesin mulai, keluarannya sama pada setiap jalan. Bila pesanan tiba *selagi* mesin bekerja, pesanan kilat hanya dapat menyalip pesanan yang masih menunggu, tidak pernah yang sudah berada di dalam mesin. **Kode 14.32** memakai tupel, yang dibandingkan Python elemen demi elemen: `(0, 2, "Bagus")` datang sebelum `(1, 1, "Ayu")` karena $0 < 1$.

Kode 14.32 `express_queue.py`: pesanan kilat lebih dahulu dengan `queue.PriorityQueue` dan tupel.

```
"""express_queue.py: antrean prioritas membiarkan pesanan kilat menyalip."""
import queue
import threading


def main():
    orders = queue.PriorityQueue()
    # (0 = kilat, 1 = biasa; lalu id pesanan) mengurutkan kilat dahulu, lalu menurut
kedatangan.
    for order_id, customer, express in [(1, "Ayu", False), (2, "Bagus", True),
                                        (3, "Citra", False), (4, "Dewi", True),
                                        (5, "Eko", False)]:
        orders.put((0 if express else 1, order_id, customer))
    orders.put((2, 0, None))  # pil penutup terurut paling akhir

    def machine():
        while True:
            rank, order_id, customer = orders.get()
            if customer is None:
                break
            print(f"W1 washes order {order_id} ({customer}" + (", express)" if rank == 0
else ")"))

    t = threading.Thread(target=machine, name="W1")
    t.start()
    t.join()


if __name__ == "__main__":
    main()
```

```
W1 washes order 2 (Bagus, express)
W1 washes order 4 (Dewi, express)
W1 washes order 1 (Ayu)
W1 washes order 3 (Citra)
W1 washes order 5 (Eko)
```

Penyelesaian 14.10. Misalkan kelima filsuf mengambil garpu kiri mereka pada saat yang sama. Masing-masing mencoba selama 5 ms memperoleh garpu kanan, gagal, meletakkan garpu kiri, lalu menunggu tepat 10 ms. Mereka semua bangun pada saat yang sama, mengambil garpu kiri bersama-sama lagi, gagal bersama-sama lagi, dan seterusnya: sebuah livelock. Tak seorang pun terblokir, garpu-garpu terus berpindah, dan tak seorang pun makan. Dengan mundur selama waktu *acak*, para filsuf bangun pada saat yang berbeda; setelah tabrakan, yang pertama bangun biasanya mendapati kedua garpu bebas, makan, dan kesimetrisan pun patah. Protokol jaringan seperti Ethernet memakai gagasan yang sama, yang disebut *randomized exponential back-off*.

Penyelesaian 14.11. (a) Tidak mungkin: tidak ada thread yang memegang satu kunci sambil menunggu kunci lain; syarat *pegang dan tunggu* tidak terpenuhi. (b) Mungkin: thread 1 memegang `a` dan menginginkan `b` sementara thread 2 memegang `b` dan menginginkan `a`, sebuah penantian melingkar dengan keempat syarat terpenuhi. Bahayanya, persarangan itu tersembunyi di dalam pemanggilan metode; itulah sebabnya memanggil metode synchronized milik objek lain sambil memegang kunci itu berisiko. (c) Deadlock satu thread dengan dirinya sendiri: `threading.Lock` tidak reentrant, sehingga `with lock:` yang di dalam menunggu selamanya kunci yang dipegang thread-nya sendiri. Pakailah `threading.RLock`. (d) Tidak ada deadlock: kunci bawaan Java bersifat reentrant, sehingga `synchronized` yang di dalam langsung berhasil. (e) Deadlock: setiap thread menunggu yang lain berakhir, dan tidak satu pun dapat berakhir; `join` berperilaku seperti kunci yang “dipegang” sebuah thread sampai thread itu berakhir. `join` dengan batas waktu akan mengungkapkannya.

Daftar Pustaka

[1] A. Silberschatz, P. B. Galvin, and G. Gagne, *Operating System Concepts*, 10th ed. Hoboken, NJ, USA: Wiley, 2018.

[2] B. Goetz, T. Peierls, J. Bloch, J. Bowbeer, D. Holmes, and D. Lea, *Java Concurrency in Practice*. Upper Saddle River, NJ, USA: Addison-Wesley, 2006.

[3] J. Gosling, B. Joy, G. Steele, G. Bracha, A. Buckley, D. Smith, and G. Bierman, *The Java Language Specification, Java SE 21 Edition*, ch. 17, "Threads and Locks." Oracle America, 2023. [Online]. Available: https://docs.oracle.com/javase/specs/jls/se21/html/jls-17.html

[4] Python Software Foundation, *The Python Standard Library*, modules `threading`, `queue` and `concurrent.futures`, version 3.14. [Online]. Available: https://docs.python.org/3/library/threading.html

[5] J. Manson, W. Pugh, and S. V. Adve, "The Java memory model," in *Proc. 32nd ACM SIGPLAN-SIGACT Symp. Principles of Programming Languages (POPL '05)*, Long Beach, CA, USA, 2005, pp. 378–391.

[6] Oracle, "Package java.util.concurrent," *Java SE 21 API Specification*, and "Lesson: Concurrency," *The Java Tutorials*. [Online]. Available: https://docs.oracle.com/en/java/javase/21/docs/api/java.base/java/util/concurrent/package-summary.html

[7] H. M. Deitel and P. J. Deitel, *Java: How to Program*, 4th ed. Upper Saddle River, NJ, USA: Prentice Hall, 2002.

[8] G. M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," in *Proc. AFIPS Spring Joint Computer Conf.*, Atlantic City, NJ, USA, 1967, pp. 483–485.

[9] S. Gross, "PEP 703 – Making the Global Interpreter Lock optional in CPython," Python Enhancement Proposal 703, 2023. [Online]. Available: https://peps.python.org/pep-0703/

[10] T. Wouters, M. Page, and S. Gross, "PEP 779 – Criteria for supported status for free-threaded Python," Python Enhancement Proposal 779, 2025. [Online]. Available: https://peps.python.org/pep-0779/

[11] Oracle, "Synchronization" and "Guarded blocks," *The Java Tutorials*. [Online]. Available: https://docs.oracle.com/javase/tutorial/essential/concurrency/sync.html

[12] C. A. R. Hoare, "Monitors: an operating system structuring concept," *Communications of the ACM*, vol. 17, no. 10, pp. 549–557, 1974.

[13] E. W. Dijkstra, "Hierarchical ordering of sequential processes," *Acta Informatica*, vol. 1, no. 2, pp. 115–138, 1971.

[14] H. Yang, "Deadlock sample program," *Java Tutorials: Herong's Tutorial Examples*. [Online]. Available: https://www.herongyang.com/

[15] E. G. Coffman, M. J. Elphick, and A. Shoshani, "System deadlocks," *ACM Computing Surveys*, vol. 3, no. 2, pp. 67–78, 1971.

[16] W. Stallings, *Operating Systems: Internals and Design Principles*, 9th ed. Hoboken, NJ, USA: Pearson, 2018.

[17] E. W. Dijkstra, "Cooperating sequential processes," Technological University, Eindhoven, The Netherlands, Tech. Rep. EWD-123, 1965; reprinted in F. Genuys, Ed., *Programming Languages*. London, U.K.: Academic Press, 1968, pp. 43–112.

[18] K. M. Chandy and J. Misra, "The drinking philosophers problem," *ACM Transactions on Programming Languages and Systems*, vol. 6, no. 4, pp. 632–646, 1984.

[19] LogicBig, "Java — Thread livelock." [Online]. Available: https://www.logicbig.com/

Bab 15

Socket, Jaringan, dan Proyek Tim

Pada akhir bab ini Anda akan sanggup menjelaskan, dengan kata-kata sederhana dan sebuah gambar, bagaimana sebuah pesan berpindah dari satu program ke program lain melintasi jaringan: apa itu alamat IP, apa itu port, apa yang dijanjikan TCP dan UDP, dan mengapa setiap percakapan memiliki klien dan server. Anda akan membuka *socket* di Java dengan `ServerSocket` dan `Socket` dan di Python dengan modul `socket`; menulis server dan klien *echo* berbasis baris lalu menjalankannya di dua terminal; merancang protokol di atas kertas sebagai mesin keadaan sebelum menulis satu baris pun; melayani banyak klien sekaligus dengan kumpulan thread (*thread pool*) dari Bab 14; dan membangun layanan status pesanan kecil berbasis JSON di atas TCP untuk Profitina Laundry, yang klien Java-nya dapat bercakap dengan server Python-nya dan klien Python-nya dapat bercakap dengan server Java-nya, sebagaimana kita buktikan dengan menjalankan keduanya. Anda akan berbicara HTTP dari kedua bahasa kepada server yang dijalankan dan dihentikan oleh program Anda sendiri, melindungi data pribadi yang melewati layanan Anda sesuai semangat undang-undang pelindungan data pribadi Indonesia, dan menguji kode jaringan secara otomatis pada port efemeral. Paruh kedua bab ini menjadikan Anda anggota sebuah tim: cara memilih proyek, membaginya menjadi bagian-bagian dengan antarmuka yang jelas, bekerja bersama dengan Git, menulis README, menguji bagian milik rekan, bertahan ketika server atau klien mati, dan mempresentasikan hasilnya. Sebuah contoh ringkasan proyek (*project brief*) yang lengkap dan sebuah daftar periksa menutup bab ini.

15.1 Di Mana Kita Berhenti

Setiap program dalam buku ini sejauh ini hidup sendirian. Ia berjalan di satu komputer, membaca papan ketik atau berkas, dan mencetak ke layar. Bab 11 memberi program kita jendela dan Bab 14 memberinya beberapa thread, tetapi program tersibuk di Bab 14 pun tidak pernah berbicara dengan program lain.

Perangkat lunak yang sesungguhnya gemar bergaul. Laptop kasir di Profitina Laundry bertanya kepada sebuah server apakah pesanan ORD-0713 sudah siap; ponsel kurir memberi tahu server yang sama bahwa sebuah kantong cucian telah diantar; dasbor pemilik menanyakan pendapatan hari ini. Masing-masing adalah program terpisah, sering ditulis oleh orang yang berbeda dalam bahasa yang berbeda di mesin yang berbeda, tetapi semuanya bekerja sama seolah-olah satu program. Perekat di antara mereka adalah jaringan, dan pegangan pemrogram pada jaringan adalah **socket**.

Bab ini memiliki dua paruh yang saling melengkapi. Paruh pertama mengajarkan jaringan dari nol: apa yang terjadi di antara dua komputer, bagaimana Java dan Python membuka koneksi, bagaimana menyepakati kata-kata yang akan dipakai kedua pihak, dan bagaimana server menjaga banyak percakapan berlangsung sekaligus. Paruh kedua mengajari Anda membangun sistem semacam itu *bersama orang lain*, karena aplikasi klien–server adalah proyek tim yang klasik: satu orang menulis server, yang lain klien, yang ketiga antarmuka grafis, dan mereka hanya bertemu di antarmuka yang telah disepakati sebelumnya. Pelajaran bersamanya: entah pihak-pihak itu program atau manusia, mereka berhasil bila menuliskan, sebelum mulai, apa persisnya yang akan dikatakan setiap pihak dan apa yang boleh diharapkannya.

> **WAWASAN KUNCI:** Koneksi jaringan hanyalah dua aliran (*stream*), satu untuk setiap arah, persis seperti aliran berkas di Bab 5. Semua kesulitan jaringan berasal dari tiga fakta: pihak lain adalah program terpisah yang tidak Anda kendalikan, ia bisa lenyap kapan saja, dan kedua pihak harus sepakat tentang makna setiap bita. Objek membantu ketiganya: objek dapat menyembunyikan aliran di balik metode yang ramah, eksepsi dapat melaporkan lenyapnya pihak lain, dan objek protokol dapat menyimpan kesepakatan itu.

15.2 Jaringan dari Nol

Sebelum menulis satu baris kode, kita memerlukan sedikit kosakata, yang kita bangun dengan analogi yang cocok dengan studi kasus: sebuah usaha binatu di dalam gedung besar.

15.2.1 Host dan alamat IP

Komputer yang terhubung ke jaringan disebut **host**: laptop Anda, server di ruang belakang binatu, ponsel Anda. Untuk mengirim apa pun ke sebuah host, jaringan memerlukan alamatnya, sebagaimana kurir memerlukan alamat sebuah gedung.

> **DEFINISI. Alamat IP** (*Internet Protocol address*) adalah bilangan yang mengenali sambungan sebuah host ke jaringan. Alamat **IPv4** panjangnya 32 bit dan ditulis sebagai empat bilangan desimal 0 sampai 255 yang dipisahkan titik, misalnya `192.168.1.10`. Alamat **IPv6** panjangnya 128 bit dan ditulis sebagai delapan kelompok digit heksadesimal yang dipisahkan titik dua, misalnya `2001:db8::10` (deretan kelompok nol boleh disingkat menjadi `::`).

Bilangan 32 bit hanya dapat bernilai $2^{32} \approx 4.3$ miliar, terlalu sedikit untuk planet yang penuh ponsel; IPv6 adalah jawaban jangka panjangnya [1], [2]. Ada tiga jenis alamat yang penting dalam bab ini:

- `127.0.0.1` (IPv4) dan `::1` (IPv6) adalah alamat **loopback**, "komputer ini sendiri". Pesan yang dikirim ke sana tidak pernah meninggalkan mesin. Setiap contoh dalam bab ini memakai loopback, sehingga berjalan tanpa Wi-Fi dan tanpa memperlihatkan program Anda kepada orang asing.
- `localhost` adalah **nama host** yang diterjemahkan komputer menjadi alamat loopback. Menerjemahkan nama menjadi alamat adalah tugas **Domain Name System** (DNS): untuk `app.profitina.com`, DNS menjawab dengan alamat servernya.
- Alamat seperti `192.168.x.y` dan `10.x.y.z` bersifat **privat**: dipakai di dalam jaringan rumah atau kantor dan tidak dapat dijangkau langsung dari Internet. Server binatu dan laptop kasir akan memiliki alamat seperti ini.

15.2.2 Port

Alamat mengantarkan pesan ke gedung yang benar, tetapi sebuah gedung memiliki banyak kantor, dan sebuah komputer menjalankan banyak program. Sistem operasi perlu tahu untuk program mana sebuah pesan ditujukan. Itulah gunanya port.

> **DEFINISI. Port** adalah bilangan 16 bit yang dipakai sistem operasi untuk mengantarkan data jaringan ke program yang tepat di sebuah host. Program server meminta sebuah port kepada sistem operasi dan *mendengarkan* (*listen*) padanya; klien menyebut port itu ketika menyambung.

Karena port berukuran 16 bit, banyaknya port yang mungkin pada satu host, untuk satu protokol, adalah

$$2^{16} = 65\,536, \qquad \text{port } 0{,}1, \ldots, 65\,535. \tag{15.1}$$

Persamaan (15.1) adalah seluruh persediaannya, yang dibagi menjadi tiga rentang [3]. Port 0–1023 adalah port **terkenal** (*well-known*): 80 untuk HTTP, 443 untuk HTTPS, 22 untuk SSH, 53 untuk DNS; di Linux dan macOS pengguna biasa tidak boleh mendengarkan padanya. Port 1024–49151 adalah port **terdaftar** yang dipakai aplikasi, misalnya 5432 untuk PostgreSQL; kita memakai 5050, 5051, dan 5052 untuk ketiga server kita. Port 49152–65535 adalah port **efemeral** (berumur pendek) yang dibagikan sistem operasi secara otomatis, misalnya kepada sisi klien setiap koneksi. Ketika sebuah server meminta port 0, sistem memilihkan port efemeral yang bebas, persis yang diinginkan pengujian otomatis: dua uji tidak akan pernah memperebutkan nomor yang sama.

Satu percakapan antara dua program dikenali oleh empat bilangan sekaligus, yaitu **empat-serangkai** (*four-tuple*):

$$\text{koneksi} = (\,IP_c,\ p_c,\ IP_s,\ p_s\,) \tag{15.2}$$

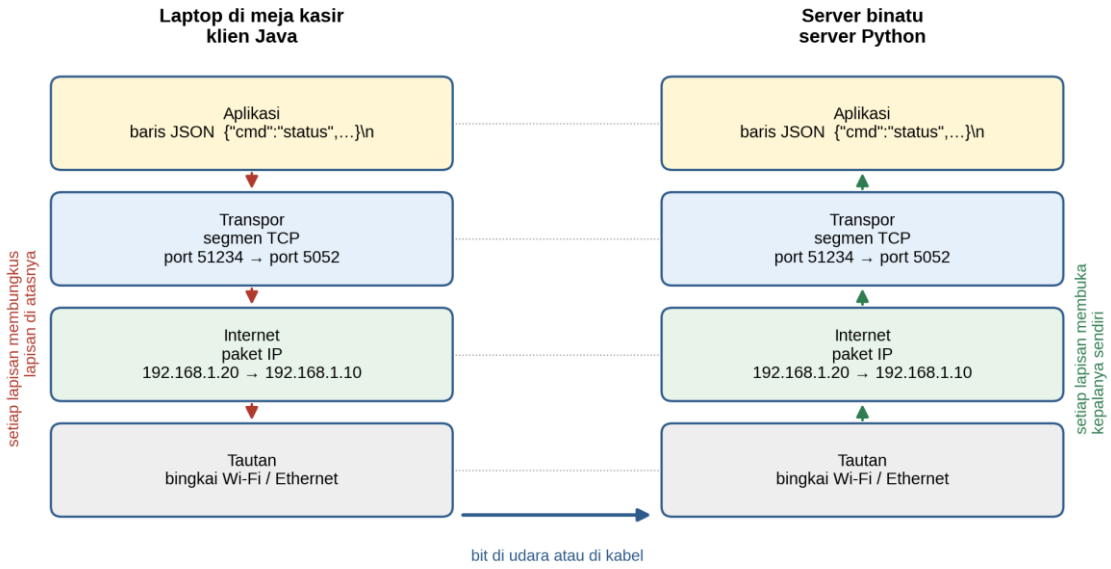
dengan IP_c dan p_c alamat dan port klien, serta IP_s dan p_s milik server. Persamaan (15.2) menjelaskan bagaimana server yang mendengarkan pada satu port dapat berbicara dengan ribuan klien sekaligus: dua tab peramban di satu laptop yang berbicara dengan server yang sama berbagi tiga dari empat bilangan itu, tetapi setiap tab mendapat port efemeral p_c sendiri, sehingga percakapannya tidak pernah tercampur.

15.2.3 Lapisan: bagaimana sebuah pesan berpindah

Mengirim pesan melintasi jaringan terdiri atas beberapa pekerjaan yang sangat berbeda: mengubah bit menjadi gelombang radio, mencari rute, memastikan tidak ada yang hilang, menentukan makna bita. Para insinyur memisahkannya dengan menumpuknya dalam **lapisan** (*layer*), yang masing-masing hanya memakai layanan lapisan di bawahnya. Internet memakai empat lapisan [1]:

1. **Lapisan tautan** (*link layer*) memindahkan satu bingkai (*frame*) bit melintasi satu lompatan fisik: dari satu radio Wi-Fi ke titik akses, dari satu kabel Ethernet ke *switch*.
2. **Lapisan internet** (IP) memindahkan sebuah *paket* dari host asal ke host tujuan melalui banyak lompatan, dengan memakai alamat IP. Ia berusaha sebaik-baiknya tetapi tidak menjanjikan apa pun: paket bisa hilang, terduplikasi, atau tiba tidak berurutan.
3. **Lapisan transpor** memindahkan data dari satu *program* ke program lain, dengan memakai port. Di sinilah TCP dan UDP berada, dan di sinilah pilihan antara keandalan dan kecepatan dibuat.
4. **Lapisan aplikasi** adalah tempat program Anda hidup. Ia menentukan makna bita: satu baris teks, satu objek JSON, satu permintaan HTTP.

Gambar 15.1 mengikuti satu permintaan status dari klien Java milik kasir ke server Python milik binatu. Dalam perjalanan turun, setiap lapisan membungkus apa yang diterimanya dengan amplopnya sendiri (sebuah *header*): TCP menambahkan port, IP menambahkan alamat, lapisan tautan menambahkan alamat perangkat keras lompatan berikutnya. Di ujung lain, setiap lapisan membuka amplopnya dan meneruskan isinya ke atas. Garis bertitik memperlihatkan ilusi paling berguna dalam dunia jaringan: setiap lapisan seakan-akan berbicara langsung dengan kembarannya di host lain. Program kita duduk di puncak; ia menulis sebaris JSON ke dalam socket dan memercayakan pengantarannya kepada lapisan-lapisan di bawahnya.



Gambar 15.1 Satu permintaan yang berjalan melalui empat lapisan Internet. Lapisan-lapisan klien membungkus baris JSON dalam segmen TCP, paket IP, dan bingkai lapisan tautan; lapisan-lapisan server membukanya dalam urutan terbalik. Setiap lapisan tampak berbicara langsung dengan kembarannya di host lain.

15.2.4 TCP dan UDP

Lapisan transpor menawarkan dua layanan, dan memilih di antara keduanya adalah keputusan rancangan pertama setiap program jaringan.

> **DEFINISI. TCP** (*Transmission Control Protocol*) menyediakan *koneksi*: *aliran bita* dua arah yang andal dan berurutan antara dua program. **UDP** (*User Datagram Protocol*) menyediakan *datagram*: pesan-pesan yang berdiri sendiri, masing-masing diantar utuh atau tidak sama sekali, tanpa janji pengantaran, tanpa janji urutan, dan tanpa koneksi.

TCP ibarat panggilan telepon. Sebelum ada yang berbicara, kedua pihak menyiapkan panggilan dengan **jabat tangan tiga langkah** (*three-way handshake*: SYN, SYN-ACK, ACK). Sejak itu, semua yang ditulis satu pihak tiba di pihak lain, berurutan, tepat sekali; paket yang hilang dikirim ulang, dan penerima yang lambat memperlambat pengirim. Harganya adalah waktu penyiapan, sedikit lalu lintas tambahan, dan satu hal halus yang akan kita temui di Subbab 15.4.4: TCP mengantarkan *aliran* bita, sehingga batas-batas antarpesan Anda *tidak* dipertahankan [4].

UDP ibarat berkirim kartu pos: tidak ada panggilan yang disiapkan, setiap kartu tiba utuh atau tidak sama sekali, mungkin tidak berurutan. UDP murah, cepat, dan menjaga batas pesan, sehingga dipakai ketika pesan yang terlambat memang tidak berguna lagi: suara dan video langsung, gim, pencarian DNS, dan detak "saya masih hidup" yang dikirim mesin setiap beberapa detik [5]. **Tabel 15.1** merangkumnya.

Tabel 15.1 Perbandingan TCP dan UDP.

Pertanyaan	TCP	UDP
Koneksi disiapkan dulu?	Ya, jabat tangan tiga langkah	Tidak
Pengantaran dijamin?	Ya; data yang hilang dikirim ulang	Tidak
Urutan dijamin?	Ya	Tidak
Batas pesan dijaga?	Tidak: aliran bita	Ya: setiap datagram utuh
Kecepatan dan beban	Lebih lambat di awal, lalu lintas lebih banyak	Minimal
Kelas Java	`ServerSocket`, `Socket`	`DatagramSocket`, `DatagramPacket`
Python	`socket.SOCK_STREAM`	`socket.SOCK_DGRAM`
Kegunaan umum	Web, surel, transfer berkas, layanan pesanan kita	Suara, video, gim, DNS, detak

Sisa bab ini memakai TCP, karena binatu tidak boleh kehilangan pesan "pesanan ORD-0713 sudah diambil". Namun UDP begitu sederhana sehingga menjadi program pertama yang sempurna.

15.2.5 Program jaringan pertama: datagram untuk diri sendiri

Kode 15.1 mengirim tiga datagram melalui loopback. Socket *penerima* diikat ke port 0, sehingga sistem operasi memilihkan port yang bebas; socket *pengirim* menembakkan tiga pesan status pendek kepadanya, lalu penerima mengumpulkannya.

Kode 15.1 `UdpHello.java`: tiga datagram UDP dikirim melalui loopback dari satu socket ke socket lain.

```
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.nio.charset.StandardCharsets;

public class UdpHello {
    public static void main(String[] args) throws Exception {
        InetAddress loopback = InetAddress.getLoopbackAddress();  // 127.0.0.1
        try (DatagramSocket receiver = new DatagramSocket(0, loopback);  // port 0: OS
memilihkan satu
             DatagramSocket sender = new DatagramSocket()) {
            receiver.setSoTimeout(2000);  // jangan pernah menunggu datagram lebih dari 2 s
            int port = receiver.getLocalPort();
            System.out.println("receiver bound to " + loopback.getHostAddress()
                    + ", port chosen by the OS: " + (port > 0));

            String[] messages = {"machine W1: washing", "machine W2: idle", "machine D1:
drying"};
            for (String message : messages) {
                byte[] data = message.getBytes(StandardCharsets.UTF_8);
                sender.send(new DatagramPacket(data, data.length, loopback, port));
                System.out.println("sent     " + data.length + " bytes: " + message);
            }

            byte[] buffer = new byte[1500];
            for (int i = 0; i < messages.length; i++) {
                DatagramPacket packet = new DatagramPacket(buffer, buffer.length);
                receiver.receive(packet);  // satu panggilan, satu datagram utuh
                String text = new String(packet.getData(), 0, packet.getLength(),
                        StandardCharsets.UTF_8);
                System.out.println("received " + packet.getLength() + " bytes from "
                        + packet.getAddress().getHostAddress() + ": " + text);
            }
        }
    }
}
```

Kompilasi dan jalankan dengan `javac UdpHello.java` dan `java UdpHello`:

```
receiver bound to 127.0.0.1, port chosen by the OS: true
sent     19 bytes: machine W1: washing
sent     16 bytes: machine W2: idle
sent     18 bytes: machine D1: drying
received 19 bytes from 127.0.0.1: machine W1: washing
received 16 bytes from 127.0.0.1: machine W2: idle
received 18 bytes from 127.0.0.1: machine D1: drying
```

Mari membacanya dari atas.

- `InetAddress.getLoopbackAddress()` mengembalikan alamat loopback sebagai `InetAddress`, representasi alamat IP di Java.
- `new DatagramSocket(0, loopback)` membuat socket UDP dan *mengikatnya* (*bind*) ke port 0 pada loopback: "port bebas mana saja, silakan". `getLocalPort()` memberi tahu port yang kita dapat; kita hanya mencetak `port > 0`, karena nomornya berubah setiap kali dijalankan.
- `new DatagramSocket()` membuat socket pengirim pada suatu port efemeral. Kedua socket berada di kepala `try (...)`, sehingga otomatis ditutup (Bab 5): socket adalah sumber daya sistem operasi, seperti berkas yang terbuka.
- `setSoTimeout(2000)` membatasi `receive` dua detik; tanpanya, satu datagram yang hilang akan membuat `receive` menunggu selamanya.

- `getBytes(StandardCharsets.UTF_8)` mengubah teks menjadi bita. Jaringan membawa bita, bukan karakter, dan buku ini selalu memakai **UTF-8**, yang dapat menyatakan huruf semua bahasa [6].
- `new DatagramPacket(data, data.length, loopback, port)` adalah datagram beserta tujuannya; `send` menyerahkannya kepada sistem. Di sisi lain, `receive(packet)` terblokir sampai satu datagram tiba lalu mengisi paket kosong; `getLength()` dan `getAddress()` memberi tahu berapa banyak yang tiba dan dari siapa.

Keluarannya memperlihatkan ciri khas UDP: tiga panggilan `send` menghasilkan tiga hasil `receive` yang panjangnya tepat 19, 16, dan 18 bita. Batas-batasnya selamat. Pada loopback tidak ada yang hilang atau tertukar urutannya; melintasi jaringan sungguhan, keduanya tidak dijanjikan.

Kode 15.2 adalah program yang sama dalam Python.

Kode 15.2 `udp_hello.py`: tiga datagram yang sama dalam Python.

```
"""udp_hello.py: tiga datagram dikirim lewat UDP ke komputer kita sendiri."""
import socket

LOOPBACK = "127.0.0.1"


def main():
    with socket.socket(socket.AF_INET, socket.SOCK_DGRAM) as receiver, \
         socket.socket(socket.AF_INET, socket.SOCK_DGRAM) as sender:
        receiver.bind((LOOPBACK, 0))  # port 0: OS memilihkan satu
        receiver.settimeout(2.0)      # jangan pernah menunggu datagram lebih dari 2 s
        port = receiver.getsockname()[1]
        print(f"receiver bound to {LOOPBACK}, port chosen by the OS: {str(port >
0).lower()}")

        messages = ["machine W1: washing", "machine W2: idle", "machine D1: drying"]
        for message in messages:
            data = message.encode("utf-8")
            sender.sendto(data, (LOOPBACK, port))
            print(f"sent     {len(data)} bytes: {message}")

        for _ in messages:
            data, (host, _port) = receiver.recvfrom(1500)  # satu panggilan, satu datagram
utuh
            print(f"received {len(data)} bytes from {host}: {data.decode('utf-8')}")


if __name__ == "__main__":
    main()
```

Jalankan dengan `python3 udp_hello.py` (di Windows, `python udp_hello.py` atau `py udp_hello.py`). Hasilnya persis sama dengan program Java:

```
receiver bound to 127.0.0.1, port chosen by the OS: true
sent     19 bytes: machine W1: washing
sent     16 bytes: machine W2: idle
sent     18 bytes: machine D1: drying
received 19 bytes from 127.0.0.1: machine W1: washing
received 16 bytes from 127.0.0.1: machine W2: idle
received 18 bytes from 127.0.0.1: machine D1: drying
```

Modul `socket` Python adalah pembungkus tipis bagi antarmuka *Berkeley sockets* milik sistem operasi, yang telah menjadi bagian Unix sejak 1983 [7], [8]. Java menyembunyikan antarmuka itu di balik satu kelas untuk setiap tugas; Python memperlihatkannya hampir tanpa perubahan:

- `socket.socket(socket.AF_INET, socket.SOCK_DGRAM)` membuat socket untuk alamat IPv4 (`AF_INET`) dan datagram (`SOCK_DGRAM`), yaitu UDP.
- `bind((LOOPBACK, 0))` mengikatnya; Python menulis alamat sebagai *tuple* `(host, port)`, dan `getsockname()[1]` adalah port yang didapat.
- `sendto` mengirim satu datagram dan `recvfrom(1500)` menerima satu datagram beserta alamat pengirimnya; `with` menutup kedua socket; `str(port > 0).lower()` mencetak `true` agar kedua keluaran cocok baris demi baris.

15.2.6 Klien dan server

Hampir setiap program jaringan memainkan salah satu dari dua peran.

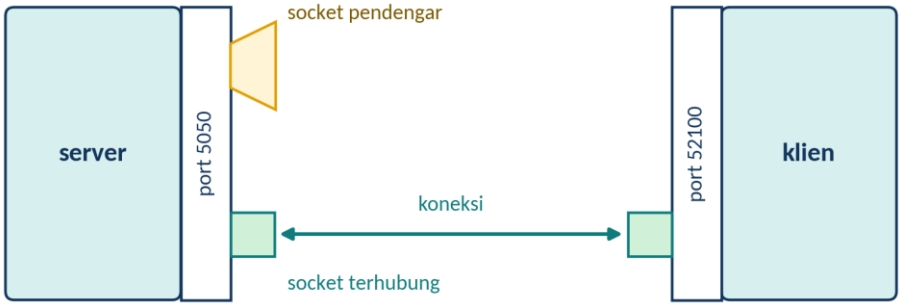
> **DEFINISI. Server** adalah program yang berjalan lebih dahulu, mengikat socket ke port yang diketahui, dan menunggu dihubungi. **Klien** adalah program yang mengetahui host dan port server lalu *memulai* kontak. Perannya menyangkut siapa yang memulai percakapan, bukan besarnya komputer: laptop dapat menjalankan server, dan mesin besar dapat menjadi klien.

Biasanya server berjalan di komputer tertentu dan memiliki socket yang terikat ke port tertentu; ia menunggu, mendengarkan permintaan koneksi dari klien. Klien mengetahui nama host dan port server, lalu berusaha menemuinya di sana; ia juga mengikat diri ke port lokalnya sendiri, yang biasanya diberikan oleh sistem [9]. **Gambar 15.2** memperlihatkan saat itu.



Gambar 15.2 Klien mengajukan permintaan koneksi ke port tempat server mendengarkan (di sini port 5050; port milik klien sendiri, misalnya 52100, adalah port efemeral yang dipilih sistemnya).

Bila semuanya lancar, server *menerima* (*accept*) koneksi itu dan mendapat socket **baru**, terikat ke port lokal yang sama, yang ujung jauhnya adalah alamat dan port klien. Server memerlukan socket baru agar tetap dapat mendengarkan pada socket semula sembari melayani klien yang baru datang. Di sisi klien, sebuah socket tercipta, dan kedua program kini berkomunikasi dengan menulis ke dan membaca dari socket masing-masing (**Gambar 15.3**).



Gambar 15.3 Setelah koneksi diterima, klien dan server terhubung melalui socket baru pada port 5050 milik server; socket pendengar milik server tetap terbuka untuk klien lain.

Pembedaan ini adalah gagasan terpenting dalam menulis server: sebuah server memiliki satu socket **pendengar** (*listening socket*) sepanjang hidupnya, yang tidak pernah membawa data dan hanya menghasilkan socket **terhubung** (*connected socket*), satu per klien, masing-masing dikenali oleh empat-serangkainya sendiri (Persamaan (15.2)).

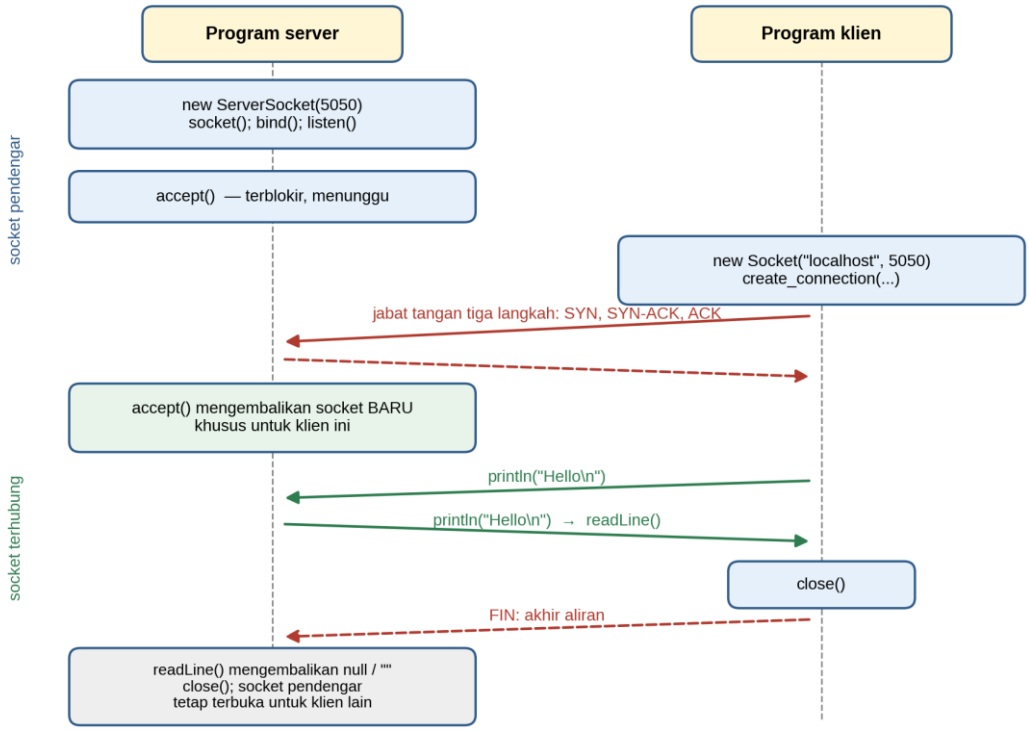
15.3 Socket di Java dan Python

> **DEFINISI. Socket** adalah satu titik ujung dari tautan komunikasi dua arah antara dua program yang berjalan di jaringan. Socket terikat ke sebuah nomor port agar lapisan transpor dapat mengenali program yang menjadi tujuan data [9].

Paket `java.net` di Java menyediakan `Socket`, satu sisi koneksi dua arah, yang dibangun di atas implementasi yang bergantung pada platform sehingga program Java dapat berkomunikasi secara tidak bergantung pada platform, dan `ServerSocket`, socket pendengar yang dipakai server untuk menerima koneksi. Untuk mengambil halaman web, kelas HTTP tingkat tinggi di Subbab 15.8 lebih nyaman; kelas-kelas itu memakai socket di bawahnya. `socket.socket` di Python memainkan kedua peran, bergantung pada metode yang Anda panggil. **Tabel 15.2** menjajarkan kedua pustaka, dalam urutan pemanggilan oleh program, dan **Gambar 15.4** memperlihatkan panggilan-panggilan itu pada garis waktu.

Tabel 15.2 Panggilan socket TCP di Java dan Python, dalam urutan pemakaiannya oleh program.

Langkah	Java	Python
Server: buat, ikat, dengarkan	`new ServerSocket(port, backlog, addr)`	`s = socket.socket()`; `s.bind((host, port))`; `s.listen(backlog)`
Server: menunggu klien	`Socket c = server.accept()`	`c, addr = s.accept()`
Klien: menyambung	`new Socket(host, port)`	`socket.create_connection((host, port))`
Membaca baris teks	`new BufferedReader(new InputStreamReader(c.getInputStream(), UTF_8))`	`c.makefile("r", encoding="utf-8")`
Menulis baris teks	`new PrintWriter(new OutputStreamWriter(c.getOutputStream(), UTF_8), true)`	`c.makefile("w", encoding="utf-8")` lalu `flush()`
Bita mentah	`getInputStream().read(...)`, `getOutputStream().write(...)`	`c.recv(n)`, `c.sendall(data)`
Membatasi penantian	`c.setSoTimeout(ms)`	`c.settimeout(seconds)`
Menutup	`c.close()` atau try-with-resources	`c.close()` atau `with`
Pihak lain menutup	`readLine()` mengembalikan `null`	`readline()` mengembalikan `""`
Server tidak ada	`ConnectException`	`ConnectionRefusedError`



Gambar 15.4 Riwayat hidup satu koneksi TCP. Server membuat socket pendengar dan terblokir di `accept()`; penyambungan klien memicu jabat tangan tiga langkah; `accept()` mengembalikan socket terhubung yang baru; baris-baris mengalir dua arah; `close()` klien mengirim FIN, yang dilihat server sebagai akhir aliran masukannya.

Tiga hal perlu dikomentari sebelum kita memakainya.

Aliran di atas socket. Socket terhubung memberi bita mentah: `InputStream` dan `OutputStream` di Java, `recv` dan `sendall` di Python. Seperti di Bab 5, kita membungkusnya: di Java, `InputStreamReader` ditambah `BufferedReader` memberi `readLine()`, dan `OutputStreamWriter` ditambah `PrintWriter` memberi `println`; di Python, `makefile` mengembalikan objek serupa berkas dengan `readline`, `write`, dan iterasi.

Selalu sebutkan pengodeannya. Bila tidak, kedua bahasa memakai pengodean bawaan komputer, yang dapat berbeda antara klien Windows dan server Linux, dan keduanya akan berselisih tentang setiap huruf di luar huruf Latin dasar.

Pembilasan (*flushing*). Penulis menampung apa yang Anda tulis dalam penyangga (*buffer*), yang efisien untuk berkas tetapi fatal untuk percakapan: pihak lain menunggu baris yang masih tertahan di penyangga Anda. `PrintWriter(writer, true)` di Java menyalakan *autoflush* pada setiap `println`; di Python kita memanggil `writer.flush()`. Lupa membilas adalah penyebab paling umum program jaringan pertama "macet".

15.4 Server dan Klien *Echo* Berbasis Baris

Pasangan TCP pertama kita adalah protokol berguna yang paling sederhana: klien mengirim sebaris teks dan server mengirimkannya kembali, sebuah *echo* (gema). Inilah "Hello, World" dunia jaringan, dan server *echo* sungguh dipakai untuk memeriksa jalur jaringan dan mengukur waktu pulang-pergi.

15.4.1 Server

Kode 15.3 adalah server Java. Ia mengambil nomor port dari baris perintah, menunggu satu klien, memantulkan setiap baris yang dikirim klien itu, dan berhenti ketika klien menutup sambungan.

Kode 15.3 `EchoServer.java`: terima satu klien dan kirim balik setiap barisnya.

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.net.InetAddress;
import java.net.ServerSocket;
import java.net.Socket;
import java.nio.charset.StandardCharsets;

public class EchoServer {
    public static void main(String[] args) {
        if (args.length != 1) {
            System.err.println("Usage: java EchoServer <port>");
            System.exit(1);
        }
        int port = Integer.parseInt(args[0]);
        InetAddress loopback = InetAddress.getLoopbackAddress();

        // backlog 50: paling banyak 50 permintaan koneksi boleh menunggu dalam antrean
        try (ServerSocket serverSocket = new ServerSocket(port, 50, loopback)) {
            System.out.println("EchoServer listening on " + loopback.getHostAddress()
                    + " port " + serverSocket.getLocalPort());
            try (Socket clientSocket = serverSocket.accept();  // terblokir sampai ada
klien datang
                 BufferedReader in = new BufferedReader(new InputStreamReader(
                         clientSocket.getInputStream(), StandardCharsets.UTF_8));
                 PrintWriter out = new PrintWriter(new OutputStreamWriter(
                         clientSocket.getOutputStream(), StandardCharsets.UTF_8), true)) {
                System.out.println("Client connected from "
                        + clientSocket.getInetAddress().getHostAddress());
                int lines = 0;
                String line;
                while ((line = in.readLine()) != null) {  // null: klien sudah menutup
                    out.println(line);                    // autoflush langsung
mengirimkannya
                    lines++;
                }
                System.out.println("Client closed the connection after " + lines + "
lines");
            }
        } catch (IOException e) {
            System.out.println("EchoServer error: " + e.getMessage());
        }
        System.out.println("EchoServer stopped");
    }
}
```

Baris demi baris:

- `if (args.length != 1)` mewajibkan tepat satu argumen, yaitu port, dan bila tidak, mencetak pesan cara pakai lalu keluar dengan status 1.
- `new ServerSocket(port, 50, loopback)` membuat socket pendengar hanya pada loopback, sehingga hanya program di komputer ini yang dapat menyambung, bawaan yang aman selama pengembangan (Subbab 15.9 menjelaskan cara membukanya). **Backlog**, yaitu 50, adalah banyaknya permintaan koneksi yang boleh diantrekan sistem selagi program sibuk.

- `serverSocket.accept()` *terblokir*: thread berhenti di sini sampai ada klien yang menyambung, lalu `accept` mengembalikan `Socket` baru untuk klien itu, socket terhubung pada **Gambar 15.3**.
- Kepala `try` membungkus aliran socket dalam `BufferedReader` UTF-8 dan `PrintWriter` yang ber-*autoflush*; try-with-resources menutup ketiganya.
- `while ((line = in.readLine()) != null)` adalah jantung server: `readLine` terblokir sampai satu baris utuh tiba dan mengembalikan `null` ketika klien menutup ujungnya. `out.println(line)` segera mengirim baris itu kembali.
- Setiap `IOException`, misalnya port yang sudah dipakai, dilaporkan dalam satu baris.

Kode 15.4 adalah server Python.

Kode 15.4 `echo_server.py`: server *echo* dalam Python.

```
"""echo_server.py: terima satu klien dan kirim balik setiap barisnya."""
import socket
import sys

LOOPBACK = "127.0.0.1"


def main():
    if len(sys.argv) != 2:
        print("Usage: python3 echo_server.py <port>", file=sys.stderr)
        sys.exit(1)
    port = int(sys.argv[1])

    try:
        with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as server_socket:
            if sys.platform != "win32":  # pakai ulang port segera setelah dijalankan ulang
                server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
            server_socket.bind((LOOPBACK, port))
            server_socket.listen(50)  # paling banyak 50 permintaan koneksi boleh menunggu
dalam antrean
            print(f"EchoServer listening on {LOOPBACK} port
{server_socket.getsockname()[1]}",
                  flush=True)
            client_socket, (client_host, _) = server_socket.accept()  # terblokir sampai
ada klien datang
            with client_socket, \
                 client_socket.makefile("r", encoding="utf-8", newline="\n") as reader, \
                 client_socket.makefile("w", encoding="utf-8", newline="\n") as writer:
                print(f"Client connected from {client_host}", flush=True)
                lines = 0
                for line in reader:           # perulangan berakhir saat klien menutup
                    writer.write(line)
                    writer.flush()            # kirim seketika
                    lines += 1
                print(f"Client closed the connection after {lines} lines")
    except OSError as e:
        print(f"EchoServer error: {e}")
    print("EchoServer stopped")


if __name__ == "__main__":
    main()
```

Panggilan-panggilannya mengikuti kolom Python pada **Tabel 15.2**: `SOCK_STREAM` berarti aliran, yaitu TCP; `bind`, `listen(50)`, dan `accept()` melakukan apa yang dikatakan namanya. Dua baris perlu dikomentari.

- `setsockopt(SOL_SOCKET, SO_REUSEADDR, 1)` memungkinkan server mengikat lagi segera setelah dijalankan ulang. Pihak yang lebih dahulu menutup koneksi TCP menyimpan catatan

tentangnya selama satu atau dua menit (keadaan TIME_WAIT), dan tanpa opsi ini Linux dan macOS menolak server baru mengikat port itu sampai catatannya kedaluwarsa. `ServerSocket` di Java memasang opsi ini secara bawaan; di Windows opsi ini justru membuat dua server dapat berbagi satu port, sehingga kita memasangnya hanya di sistem lain.

- `for line in reader:` membaca baris-baris klien sampai klien menutup koneksi; setiap baris tetap membawa `\n`-nya, sehingga `writer.write(line)` mengembalikannya tanpa perubahan, dan `flush()` mengirimkannya.

Panggilan `print` di awal memakai `flush=True`: ketika keluaran diarahkan ke pipa, bukan ke terminal, Python menampungnya, dan pesan "listening" dapat tertahan tanpa terlihat selagi sebuah uji menunggunya.

15.4.2 Klien

Kode 15.5 adalah kliennya. Ia menyambung, lalu berulang: baca sebaris dari papan ketik, kirim, baca gemanya, cetak. Ia berhenti setelah baris `bye` atau di akhir masukan.

Kode 15.5 `EchoClient.java`: kirim baris yang diketik pengguna dan cetak apa yang kembali.

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.net.ConnectException;
import java.net.Socket;
import java.net.UnknownHostException;
import java.nio.charset.StandardCharsets;

public class EchoClient {
    public static void main(String[] args) {
        if (args.length != 2) {
            System.err.println("Usage: java EchoClient <host> <port>");
            System.exit(1);
        }
        String host = args[0];
        int port = Integer.parseInt(args[1]);

        try (Socket socket = new Socket(host, port);  // 1. buka socket
             PrintWriter out = new PrintWriter(new OutputStreamWriter(      // 2. buka
aliran
                     socket.getOutputStream(), StandardCharsets.UTF_8), true);
             BufferedReader in = new BufferedReader(new InputStreamReader(
                     socket.getInputStream(), StandardCharsets.UTF_8));
             BufferedReader keyboard = new BufferedReader(new InputStreamReader(
                     System.in, StandardCharsets.UTF_8))) {
            System.out.println("Connected to " + host + " port " + port
                    + ". Type lines; 'bye' or end of input quits.");
            String userLine;
            while ((userLine = keyboard.readLine()) != null) {     // 3. bercakap
                out.println(userLine);
                String reply = in.readLine();
                if (reply == null) {
                    System.out.println("The server closed the connection.");
                    break;
                }
                System.out.println("echo: " + reply);
                if (userLine.equals("bye")) {
                    break;
                }
            }
        } catch (UnknownHostException e) {                 // 4.-5. try-with-resources
            System.out.println("Unknown host: " + host);   //       menutup aliran dan
socket
        } catch (ConnectException e) {
            System.out.println("Cannot connect to " + host + " port " + port
                    + ": is the server running?");
```

Kode 15.5 (Lanjutan) *EchoClient.java: kirim baris yang diketik pengguna dan cetak apa yang kembali.*

```
        } catch (IOException e) {
            System.out.println("I/O error: " + e.getMessage());
        }
    }
}
```

Komentar bernomor menandai lima langkah yang diikuti setiap klien, apa pun protokolnya [9]:

1. Buka socket: `new Socket(host, port)`. Konstruktor inilah yang menyambung: mencari nama host, melakukan jabat tangan tiga langkah, dan baru kembali ketika koneksi sudah ada. Bila tidak ada yang mendengarkan pada port itu, ia melempar `ConnectException`.
2. Buka aliran masukan dan aliran keluaran pada socket.
3. Baca dari dan tulis ke aliran sesuai protokol server.

4. Tutup aliran.

5. Tutup socket.

Hanya langkah 3 yang berbeda dari klien ke klien; dengan try-with-resources, langkah 4 dan 5 terjadi otomatis, bahkan ketika eksepsi menyela langkah 3. Ketiga klausa `catch` mengubah tiga cara sebuah koneksi dapat gagal menjadi kalimat yang dipahami pengguna: host yang tidak dikenal, server yang tidak berjalan (`ConnectException`, "connection refused"), dan galat masukan atau keluaran lainnya.

Kode 15.6 adalah klien Python.

Kode 15.6 `echo_client.py`: klien *echo* dalam Python.

```
"""echo_client.py: kirim baris yang diketik pengguna dan cetak apa yang kembali."""
import socket
import sys


def main():
    if len(sys.argv) != 3:
        print("Usage: python3 echo_client.py <host> <port>", file=sys.stderr)
        sys.exit(1)
    host, port = sys.argv[1], int(sys.argv[2])

    try:
        with socket.create_connection((host, port)) as sock, \
             sock.makefile("w", encoding="utf-8", newline="\n") as writer, \
             sock.makefile("r", encoding="utf-8", newline="\n") as reader:
            print(f"Connected to {host} port {port}. Type lines; 'bye' or end of input
quits.")
            for user_line in sys.stdin:
                user_line = user_line.rstrip("\r\n")
                writer.write(user_line + "\n")
                writer.flush()
                reply = reader.readline()
                if reply == "":
                    print("The server closed the connection.")
                    break
                print("echo: " + reply.rstrip("\n"))
                if user_line == "bye":
                    break
    except socket.gaierror:
        print(f"Unknown host: {host}")
    except ConnectionRefusedError:
        print(f"Cannot connect to {host} port {port}: is the server running?")
    except OSError as e:
        print(f"I/O error: {e}")


if __name__ == "__main__":
    main()
```

`socket.create_connection((host, port))` adalah versi Python yang praktis untuk langkah 1: ia mencari nama host, mencoba setiap alamat hasil nama itu (`localhost` dapat menjadi `::1` maupun `127.0.0.1`), dan mengembalikan socket yang terhubung. Eksepsinya berpadanan seperti pada **Tabel 15.2**: `socket.gaierror` untuk nama yang tidak dikenal (*get address info error*) dan `ConnectionRefusedError` ketika tidak ada server yang mendengarkan.

15.4.3 Menjalankan pasangan ini di dua terminal

Program klien–server adalah dua program yang berjalan bersamaan. Buka dua terminal (di VS Code, **Terminal → New Terminal** dua kali), masuk ke folder `java` bab ini, kompilasi sekali dengan `javac *.java`, lalu jalankan server lebih dahulu:

```
Terminal 1:  java EchoServer 5050
Terminal 2:  java EchoClient localhost 5050
```

Lalu ketikkan baris-baris di terminal 2. Agar hasilnya dapat diulang, kami memberi klien berkas `EchoClient.input.txt`, yang terletak di samping programnya:

```
Hello, Profitina Laundry!
Is order ORD-0713 ready?
Terima kasih
bye
```

dengan `java EchoClient localhost 5050 < EchoClient.input.txt`. Terminal 1, sang server, mencetak:

```
EchoServer listening on 127.0.0.1 port 5050
Client connected from 127.0.0.1
Client closed the connection after 4 lines
EchoServer stopped
```

dan terminal 2, sang klien, mencetak:

```
Connected to localhost port 5050. Type lines; 'bye' or end of input quits.
echo: Hello, Profitina Laundry!
echo: Is order ORD-0713 ready?
echo: Terima kasih
echo: bye
```

Ketika Anda mengetik sendiri, setiap baris diikuti gemanya. Pasangan Python (`python3 echo_server.py 5050` dan `python3 echo_client.py localhost 5050 < echo_client.input.txt`) mencetak baris yang sama. Karena kedua pasangan berbicara dengan protokol yang sama, teks UTF-8 dengan satu baris per pesan, klien Java dapat bekerja dengan server Python dan sebaliknya; kami mencobanya, dan keluarannya persis keluaran klien di atas, baris demi baris. Inilah cicipan pertama *interoperabilitas*: protokol, bukan bahasa, yang menentukan siapa dapat berbicara dengan siapa.

Bila tidak ada server yang berjalan, kedua klien mengatakannya dengan kalimat sederhana:

```
Cannot connect to localhost port 5050: is the server running?
```

15.4.4 Baris, pembilasan, dan pembingkaian

TCP mengantarkan aliran bita. Bila klien menulis `Hello\n` lalu `bye\n`, server dapat menerima `Hello\nbye\n` dalam satu potong, atau `Hel`, `lo\nb`, `ye\n` dalam tiga potong. TCP menjamin bita dan urutannya, tidak pernah potongannya, sehingga anggapan "satu `recv` mengembalikan satu pesan" berhasil di laptop Anda dan gagal di jaringan yang sibuk.

> **DEFINISI. Pembingkaian** (*framing*) adalah aturan yang dipakai protokol untuk menemukan di mana satu pesan berakhir dan pesan berikutnya dimulai di dalam aliran bita. Tiga aturan yang umum adalah: **pembatas** (*delimiter*; setiap pesan diakhiri baris baru), **awalan panjang** (*length prefix*; setiap pesan diawali panjangnya dalam bita), dan **ukuran tetap** (setiap pesan sama panjangnya).

Setiap protokol dalam bab ini memakai pembatas baris baru, aturan yang paling mudah di-*debug* karena lalu lintasnya dapat Anda baca sendiri; `readLine()` dan `readline()` menerapkannya,

berapa pun potongan kedatangan bitanya. Satu-satunya kewajibannya adalah bahwa pesan tidak pernah memuat karakter baris baru mentah, yang dipenuhi otomatis oleh pesan JSON kita di Subbab 15.7, karena JSON menulis baris baru di dalam string sebagai `\n`.

Pembingkaian dan pembilasan menjelaskan dua cara klasik program jaringan pertama membeku. Bila klien mengirim `Hello` *tanpa* baris baru, `readLine` server menunggu akhir baris sementara klien menunggu gemanya: sebuah **kebuntuan** (*deadlock*), seperti di Bab 14, yang kini terbentang di antara dua program. Bila klien mengirim baris baru tetapi lupa membilas, barisnya tertahan di penyangga, dengan akibat yang sama. Soal Latihan 15.3 sengaja menimbulkan keduanya.

15.4.5 Ketika pihak lain tidak ada

Dalam program yang berjalan sendirian, sebuah panggilan langsung kembali atau melempar eksepsi. Melintasi jaringan, panggilan juga bisa *macet*, karena pihak lain mungkin lambat, mati, atau terputus, dan tidak ada yang memberitahukannya. Tiga keadaan mencakup hampir semuanya:

- **Server mati ketika klien menyambung.** Mesin server menolak jabat tangan dan klien segera mendapat `ConnectException` atau `ConnectionRefusedError` (di Windows, setelah sekitar dua detik). Bila seluruh mesin tidak terjangkau, tidak ada penolakan, dan penyambungan menunggu; karena itulah Subbab 15.7 memberi batas waktu pada penyambungan.
- **Klien lenyap di tengah jalan.** Penutupan normal membuat pembacaan server mengembalikan `null` atau `""`. Komputer yang *crash* atau kabel yang tercabut mungkin tidak menghasilkan apa pun; hanya **batas waktu** (*timeout*: `setSoTimeout`, `settimeout`) yang mengubah "menunggu selamanya" menjadi eksepsi.
- **Server lenyap di tengah jalan.** Pembacaan klien berikutnya mengembalikan akhir aliran atau melempar "connection reset"; klien harus melaporkannya dan tidak boleh berpura-pura permintaan terakhir berhasil.

> **PERINGATAN:** Setiap panggilan jaringan yang terblokir memerlukan jawaban atas pertanyaan "bagaimana jika pihak lain tidak pernah menjawab?". Dalam bab ini jawabannya selalu batas waktu ditambah penangan eksepsi. Server tanpa batas waktu dapat dilumpuhkan oleh klien yang menyambung lalu diam saja; Subbab 15.9 kembali membahasnya.

15.5 Merancang Protokol di Atas Kertas Terlebih Dahulu

Protokol *echo* hanya punya satu aturan. Protokol sungguhan punya banyak: siapa berbicara lebih dahulu, apa yang boleh dikatakan setiap pihak pada setiap saat, apa yang terjadi setelah pesan yang salah, bagaimana percakapan berakhir. Ketika dua orang menulis kedua sisinya, dan dalam proyek tim selalu begitu, aturannya harus dituliskan *sebelum* siapa pun menulis kode; kalau tidak, dua gagasan tentang protokol itu baru bertemu untuk pertama kalinya pada hari demonstrasi.

> **DEFINISI. Protokol** adalah seperangkat lengkap aturan untuk percakapan antara dua program: pesan-pesan yang ada, format persisnya, urutannya, makna masing-masing, dan apa yang dilakukan setiap pihak ketika menerima pesan yang tidak diharapkannya.

Alat terbaik untuk menulis protokol di atas kertas adalah **mesin keadaan** (*state machine*) yang dipakai Bab 1 untuk riwayat hidup pesanan binatu: sekumpulan kecil *keadaan*, dan untuk setiap keadaan, pesan yang diizinkan beserta ke mana pesan itu menuju. Kita berlatih dengan lelucon *knock-knock* ("tok-tok"), yang dipopulerkan oleh tutorial Java [9], dengan server yang selalu berbicara lebih dahulu:

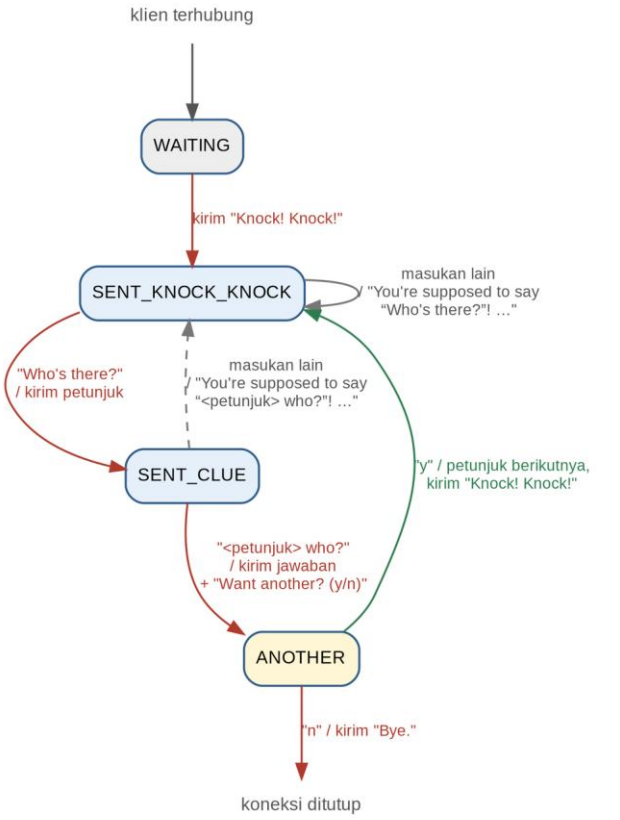
```
Server: Knock! Knock!
Client: Who's there?
Server: Lettuce
Client: Lettuce who?
Server: Lettuce in, it's cold out here! Want another? (y/n)
Client: n
Server: Bye.
```

Tabel 15.3 menuliskan aturannya sebagai tabel. Setiap baris juga menyatakan apa yang terjadi ketika klien mengatakan hal lain; baris-baris itulah yang sering dilupakan pemrogram, padahal di situlah program jaringan menghabiskan sebagian besar hidup nyatanya.

Tabel 15.3 Protokol knock-knock sebagai tabel keadaan dan pesan. Pencocokan mengabaikan huruf besar dan kecil.

Keadaan server	Klien berkata	Server menjawab	Keadaan berikutnya
`WAITING`	(menyambung)	`Knock! Knock!`	`SENT_KNOCK_KNOCK`
`SENT_KNOCK_KNOCK`	`Who's there?`	petunjuknya, mis. `Lettuce`	`SENT_CLUE`
`SENT_KNOCK_KNOCK`	hal lain	`You're supposed to say "Who's there?"! …`	`SENT_KNOCK_KNOCK`
`SENT_CLUE`	`<clue> who?`	jawaban + `Want another? (y/n)`	`ANOTHER`
`SENT_CLUE`	hal lain	`You're supposed to say "<clue> who?"! …`	`SENT_KNOCK_KNOCK`
`ANOTHER`	`y`	`Knock! Knock!` (lelucon berikutnya)	`SENT_KNOCK_KNOCK`
`ANOTHER`	hal lain	`Bye.`, lalu server menutup	`WAITING`

Gambar 15.5 menggambar aturan yang sama sebagai diagram keadaan, yang sekaligus menjadi diagram alir protokol. Tabel lebih baik untuk memeriksa bahwa setiap kasus tercakup, diagram lebih baik untuk melihat bentuk percakapan; sebuah tim sebaiknya menyimpan keduanya di repositori di samping kodenya.



Gambar 15.5 Protokol knock-knock sebagai mesin keadaan. Setiap panah berlabel "apa kata klien / apa jawaban server"; panah kelabu menangani masukan tak terduga; panah hijau memulai lelucon berikutnya.

Kini keputusan rancangan yang membuat sisa bab ini mudah: **protokol adalah sebuah objek, dan ia tidak tahu apa pun tentang socket**. **Kode 15.7** menerapkan **Tabel 15.3** sebagai kelas yang satu-satunya metode publiknya, `processInput`, menerima baris terakhir klien dan mengembalikan baris server berikutnya. Keadaannya berupa *field* privat bertipe `enum`; satu objek memegang satu percakapan.

Kode 15.7 `KnockKnockProtocol.java`: protokol **Tabel 15.3** sebagai mesin keadaan, dengan `main` yang memainkan percakapan berskenario.

```
/**
 * Protokol knock-knock sebagai mesin keadaan: satu objek per percakapan.
 * Objek ini tidak tahu apa pun tentang socket, sehingga dapat diuji tanpa jaringan.
 */
public class KnockKnockProtocol {
    private enum State { WAITING, SENT_KNOCK_KNOCK, SENT_CLUE, ANOTHER }

    private static final String[] CLUES = {"Lettuce", "Iron", "Dryer", "Atch"};
    private static final String[] ANSWERS = {
        "Lettuce in, it's cold out here!",
        "Iron your shirts? We already did!",
        "Dryer your tears, your laundry is ready!",
        "Bless you!"
    };

    private State state = State.WAITING;
    private int joke = 0;

    /** Mengembalikan baris server berikutnya, diberikan baris terakhir klien (null di
awal). */
    public String processInput(String input) {
        switch (state) {
            case WAITING:
                state = State.SENT_KNOCK_KNOCK;
                return "Knock! Knock!";
            case SENT_KNOCK_KNOCK:
                if (input.equalsIgnoreCase("Who's there?")) {
                    state = State.SENT_CLUE;
                    return CLUES[joke];
                }
                return "You're supposed to say \"Who's there?\"! Try again. Knock! Knock!";
            case SENT_CLUE:
                if (input.equalsIgnoreCase(CLUES[joke] + " who?")) {
                    state = State.ANOTHER;
                    return ANSWERS[joke] + " Want another? (y/n)";
                }
                state = State.SENT_KNOCK_KNOCK;
                return "You're supposed to say \"" + CLUES[joke]
                        + " who?\"! Try again. Knock! Knock!";
            case ANOTHER:
                if (input.equalsIgnoreCase("y")) {
                    joke = (joke + 1) % CLUES.length;
                    state = State.SENT_KNOCK_KNOCK;
                    return "Knock! Knock!";
                }
                state = State.WAITING;
                return "Bye.";
```

Kode 15.7 (Lanjutan) *KnockKnockProtocol.java: protokol Tabel 15.3 sebagai mesin keadaan, dengan main yang memainkan percakapan berskenario.*

```
            default:
                throw new IllegalStateException("unknown state " + state);
        }
    }

    public boolean isFinished(String serverLine) {
        return serverLine.equals("Bye.");
    }

    public static void main(String[] args) {
        // Percakapan berskenario, termasuk dua kesalahan dari klien.
        String[] clientLines = {
            "Who's there?", "Lettuce who?", "y",
            "Who is it?", "Who's there?", "Iron?", "Who's there?", "Iron who?", "n"
        };
        KnockKnockProtocol protocol = new KnockKnockProtocol();
        String serverLine = protocol.processInput(null);
        System.out.println("Server: " + serverLine);
        for (String clientLine : clientLines) {
            System.out.println("Client: " + clientLine);
            serverLine = protocol.processInput(clientLine);
            System.out.println("Server: " + serverLine);
            if (protocol.isFinished(serverLine)) {
                break;
            }
        }
    }
}
```

Metode `main`-nya memainkan satu percakapan utuh, termasuk dua kesalahan, tanpa jaringan sama sekali:

```
Server: Knock! Knock!
Client: Who's there?
Server: Lettuce
Client: Lettuce who?
Server: Lettuce in, it's cold out here! Want another? (y/n)
Client: y
Server: Knock! Knock!
Client: Who is it?
Server: You're supposed to say "Who's there?"! Try again. Knock! Knock!
Client: Who's there?
Server: Iron
Client: Iron?
Server: You're supposed to say "Iron who?"! Try again. Knock! Knock!
Client: Who's there?
Server: Iron
Client: Iron who?
Server: Iron your shirts? We already did! Want another? (y/n)
Client: n
Server: Bye.
```

Cocokkan keluarannya dengan **Tabel 15.3**. Setelah `y`, `joke` maju ke petunjuk kedua. `Who is it?` bukan `Who's there?`, sehingga server membetulkan klien dan tetap di `SENT_KNOCK_KNOCK`. `Iron?` bukan `Iron who?`, sehingga protokol kembali ke `SENT_KNOCK_KNOCK` dan mengulang lelucon itu, sebagaimana dinyatakan baris kelima. `n` mengakhirinya dengan `Bye.`.

`switch` meniru tabel baris demi baris, satu `case` per keadaan dan satu `return` tanpa syarat untuk "hal lain": sebuah program paling mudah diperiksa terhadap spesifikasinya bila bentuknya sama dengan bentuk spesifikasi itu. `isFinished` memberi tahu kode socket kapan harus menutup sambungan tanpa ia perlu tahu apa pun tentang lelucon.

Kode 15.8 adalah versi Python, dengan `enum.Enum` dan rangkaian pernyataan `if`.

Kode 15.8 `knock_knock_protocol.py`: mesin keadaan yang sama dalam Python.

```
"""knock_knock_protocol.py: protokol knock-knock sebagai mesin keadaan.

Satu objek per percakapan. Objek ini tidak tahu apa pun tentang socket,
sehingga dapat diuji tanpa jaringan.
"""
from enum import Enum, auto


class State(Enum):
    WAITING = auto()
    SENT_KNOCK_KNOCK = auto()
    SENT_CLUE = auto()
    ANOTHER = auto()


CLUES = ["Lettuce", "Iron", "Dryer", "Atch"]
ANSWERS = [
    "Lettuce in, it's cold out here!",
    "Iron your shirts? We already did!",
    "Dryer your tears, your laundry is ready!",
    "Bless you!",
]


class KnockKnockProtocol:
    def __init__(self):
        self._state = State.WAITING
        self._joke = 0

    def process_input(self, text):
        """Kembalikan baris server berikutnya, diberikan baris terakhir klien (None di
awal)."""
        if self._state is State.WAITING:
            self._state = State.SENT_KNOCK_KNOCK
            return "Knock! Knock!"
        if self._state is State.SENT_KNOCK_KNOCK:
            if text.lower() == "who's there?":
                self._state = State.SENT_CLUE
                return CLUES[self._joke]
            return "You're supposed to say \"Who's there?\"! Try again. Knock! Knock!"
        if self._state is State.SENT_CLUE:
            if text.lower() == (CLUES[self._joke] + " who?").lower():
                self._state = State.ANOTHER
                return ANSWERS[self._joke] + " Want another? (y/n)"
            self._state = State.SENT_KNOCK_KNOCK
            return f"You're supposed to say \"{CLUES[self._joke]} who?\"! Try again. Knock!
Knock!"
        # State.ANOTHER
```

Kode 15.8 (Lanjutan) *knock_knock_protocol.py: mesin keadaan yang sama dalam Python.*

```
        if text.lower() == "y":
            self._joke = (self._joke + 1) % len(CLUES)
            self._state = State.SENT_KNOCK_KNOCK
            return "Knock! Knock!"
        self._state = State.WAITING
        return "Bye."

    @staticmethod
    def is_finished(server_line):
        return server_line == "Bye."


def main():
    # Percakapan berskenario, termasuk dua kesalahan dari klien.
    client_lines = [
        "Who's there?", "Lettuce who?", "y",
        "Who is it?", "Who's there?", "Iron?", "Who's there?", "Iron who?", "n",
    ]
    protocol = KnockKnockProtocol()
    server_line = protocol.process_input(None)
    print("Server: " + server_line)
    for client_line in client_lines:
        print("Client: " + client_line)
        server_line = protocol.process_input(client_line)
        print("Server: " + server_line)
        if protocol.is_finished(server_line):
            break


if __name__ == "__main__":
    main()
```

Hasil cetaknya persis sama dengan versi Java:

```
Server: Knock! Knock!
Client: Who's there?
Server: Lettuce
Client: Lettuce who?
Server: Lettuce in, it's cold out here! Want another? (y/n)
Client: y
Server: Knock! Knock!
Client: Who is it?
Server: You're supposed to say "Who's there?"! Try again. Knock! Knock!
Client: Who's there?
Server: Iron
Client: Iron?
Server: You're supposed to say "Iron who?"! Try again. Knock! Knock!
Client: Who's there?
Server: Iron
Client: Iron who?
Server: Iron your shirts? We already did! Want another? (y/n)
Client: n
Server: Bye.
```

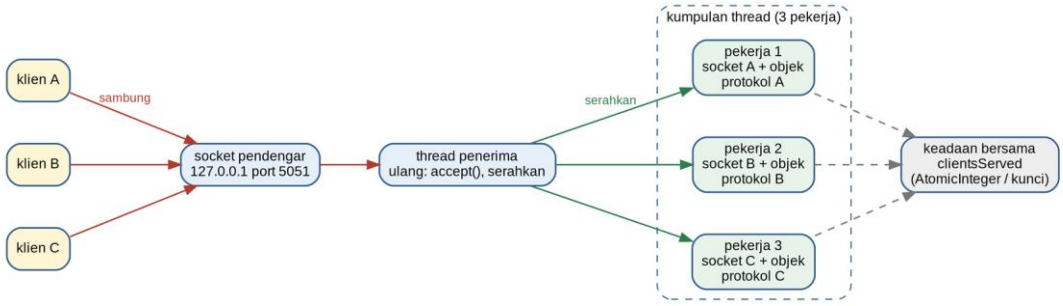
Mengapa pemisahan ini sepadan dengan kelas tambahannya? Karena ia memecah satu masalah sulit menjadi dua masalah mudah. Objek protokol dapat diuji dengan panggilan metode biasa, cepat dan deterministik, tanpa port, thread, atau urusan waktu. Kode socket menjadi perulangan pendek yang berlaku untuk protokol berbasis baris *apa saja*: baca sebaris, tanyakan kepada objek protokol, tulis jawabannya. Dalam tim, pemilik protokol dan pemilik urusan jaringan dapat bekerja sejajar, hanya bertemu di `processInput`. Layanan binatu di Subbab 15.7 memakai pemisahan yang sama, dengan metode `handle`.

> **WAWASAN KUNCI:** Tulislah protokol tiga kali sebelum menjalankannya: sebagai tabel (**Tabel 15.3**), sebagai diagram (**Gambar 15.5**), dan sebagai objek tanpa I/O (**Kode 15.7**). Setiap versi menangkap kesalahan yang disembunyikan versi lain, dan versi ketiga dapat diuji tanpa jaringan.

15.6 Melayani Banyak Klien Sekaligus

Server *echo* di Subbab 15.4 melayani satu klien lalu berhenti. Versi yang berulang (terima klien, layani sampai ia pergi, terima klien berikutnya) memang benar tetapi hampir tidak berguna: selama ia berbicara dengan klien pertama tidak ada yang memanggil `accept`, sehingga klien kedua menunggu di antrean *backlog* sampai klien pertama berkata `bye`. Bila pengguna pertama pergi makan siang, semua orang ikut menunggu makan siang itu.

Obatnya adalah pelajaran Bab 14: kerjakan pekerjaan yang saling bebas di thread yang terpisah. Perulangan utama hanya menerima; setiap socket yang diterima diserahkan ke thread lain, dan perulangan segera kembali ke `accept`. **Gambar 15.6** memperlihatkan strukturnya.



Gambar 15.6 Server multiklien. Thread penerima hanya menerima koneksi dan menyerahkan setiap socket baru kepada seorang pekerja dari kumpulan thread berukuran tetap; setiap pekerja memiliki satu socket dan satu objek protokol; satu-satunya keadaan yang dibagi para pekerja dijaga oleh pencacah atomik atau kunci.

15.6.1 Satu thread per klien, dari sebuah kumpulan

Cara paling sederhana adalah `new Thread(...)` per koneksi, yang dipakai ruang obrolan pada Soal Latihan 15.11. Kelemahannya, setiap thread memakan memori untuk tumpukannya (*stack*), baik sedang bekerja maupun menunggu. Dengan n klien dan tumpukan berukuran s per thread, tumpukan saja memerlukan sekitar

$$M \approx n \cdot s \tag{15.3}$$

memori: dengan $s = 1$ MB yang lazim, seribu klien yang menganggur memesan sekitar 1 GB. Persamaan (15.3) adalah alasan server memakai **kumpulan thread** (*thread pool*, Bab 14): sejumlah tetap pekerja yang mengambil tugas dari antrean, sehingga di bawah beban tinggi klien baru menunggu alih-alih menghabiskan memori. *Virtual thread* di Java 21 membuat satu thread per klien kembali murah, dan `asyncio` di Python melayani banyak socket dari satu thread; rancangan **Gambar 15.6** tetap berlaku untuk keduanya [10].

15.6.2 Server knock-knock multiklien

Kode 15.9 adalah server knock-knock dengan kumpulan thread, ditulis sebagai kelas dengan `start` dan `stop` agar dapat dijalankan dari baris perintah maupun di dalam uji otomatis.

Kode 15.9 `KnockKnockServer.java`: server knock-knock yang melayani banyak klien sekaligus, satu thread kumpulan per klien, dengan `stop()` yang bersih.

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.net.InetAddress;
import java.net.ServerSocket;
import java.net.Socket;
import java.net.SocketException;
import java.nio.charset.StandardCharsets;
import java.util.Set;
import java.util.concurrent.ConcurrentHashMap;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.atomic.AtomicInteger;

/** Server knock-knock yang melayani banyak klien sekaligus, satu thread kumpulan per
klien. */
public class KnockKnockServer {
    private static final int IDLE_TIMEOUT_MS = 60_000;

    private final ExecutorService pool;
    private final boolean verbose;
    private final AtomicInteger nextId = new AtomicInteger();
    private final AtomicInteger clientsServed = new AtomicInteger();
    private final Set<Socket> openSockets = ConcurrentHashMap.newKeySet();
    private ServerSocket serverSocket;
    private Thread acceptThread;
    private volatile boolean running;

    public KnockKnockServer(int maxClients, boolean verbose) {
        this.pool = Executors.newFixedThreadPool(maxClients);
        this.verbose = verbose;
    }

    /** Mengikat ke alamat loopback, mulai menerima, dan mengembalikan port yang
sebenarnya. */
    public int start(int port) throws IOException {
        serverSocket = new ServerSocket(port, 50, InetAddress.getLoopbackAddress());
        running = true;
        acceptThread = new Thread(this::acceptLoop, "acceptor");
        acceptThread.start();
        return serverSocket.getLocalPort();
    }
```

Kode 15.9 (Lanjutan) *KnockKnockServer.java: server knock-knock yang melayani banyak klien sekaligus, satu thread kumpulan per klien, dengan stop() yang bersih.*

```
    private void acceptLoop() {
        while (running) {
            try {
                Socket socket = serverSocket.accept();
                int id = nextId.incrementAndGet();
                log("client #" + id + " connected");
                pool.execute(() -> handle(socket, id));  // serahkan, lalu terima klien
berikutnya
            } catch (SocketException e) {
                break;  // stop() telah menutup socket server
            } catch (IOException e) {
                log("accept failed: " + e.getMessage());
            }
        }
    }

    private void handle(Socket socket, int id) {
        openSockets.add(socket);
        try (socket;
             BufferedReader in = new BufferedReader(new InputStreamReader(
                     socket.getInputStream(), StandardCharsets.UTF_8));
             PrintWriter out = new PrintWriter(new OutputStreamWriter(
                     socket.getOutputStream(), StandardCharsets.UTF_8), true)) {
            socket.setSoTimeout(IDLE_TIMEOUT_MS);  // klien yang diam tidak dapat menahan
thread selamanya
            KnockKnockProtocol protocol = new KnockKnockProtocol();  // satu per percakapan
            out.println(protocol.processInput(null));
            String line;
            while ((line = in.readLine()) != null) {
                String reply = protocol.processInput(line);
                out.println(reply);
                if (protocol.isFinished(reply)) {
                    break;
                }
            }
        } catch (IOException e) {
            log("client #" + id + " dropped: " + e.getMessage());
        } finally {
            openSockets.remove(socket);
            clientsServed.incrementAndGet();
            log("client #" + id + " done");
        }
    }
```

Kode 15.9 (Lanjutan) *KnockKnockServer.java: server knock-knock yang melayani banyak klien sekaligus, satu thread kumpulan per klien, dengan stop() yang bersih.*

```
    /** Berhenti menerima, menutup setiap percakapan yang terbuka, dan menunggu kumpulan
thread. */
    public void stop() throws InterruptedException {
        running = false;
        try {
            serverSocket.close();  // membuat accept() melempar SocketException
        } catch (IOException ignored) {
            // penutupan dilakukan sebisanya
        }
        acceptThread.join();
        for (Socket socket : openSockets) {
            try {
                socket.close();
            } catch (IOException ignored) {
                // penutupan dilakukan sebisanya
            }
        }
        pool.shutdown();
        pool.awaitTermination(5, TimeUnit.SECONDS);
    }

    public int clientsServed() {
        return clientsServed.get();
    }

    private void log(String message) {
        if (verbose) {
            System.out.println(message);
        }
    }

    public static void main(String[] args) throws Exception {
        int port = args.length > 0 ? Integer.parseInt(args[0]) : 5051;
        KnockKnockServer server = new KnockKnockServer(8, true);
        int actual = server.start(port);
        System.out.println("KnockKnockServer listening on port " + actual + " (Ctrl+C stops
it)");
        Runtime.getRuntime().addShutdownHook(new Thread(() -> {
            try {
                server.stop();
            } catch (InterruptedException ignored) {
                // toh program memang akan keluar
            }
            System.out.println("KnockKnockServer stopped after " + server.clientsServed()
                    + " clients");
        }));
    }
}
```

Kelas ini memiliki empat tanggung jawab, masing-masing satu metode.

- **`start(port)`** membuat socket pendengar pada loopback, menjalankan thread `acceptor`, dan mengembalikan port yang benar-benar terikat (penting untuk port 0). Ia langsung kembali, sehingga pemanggilnya tetap bebas.
- **`acceptLoop`** adalah pintu depan: ia berulang pada `accept()` dan menyerahkan setiap socket baru kepada kumpulan dengan `pool.execute(() -> handle(socket, id))`. Ia tidak pernah berbicara sendiri dengan klien, sehingga klien yang lambat tidak dapat menghalangi pintu.
- **`handle(socket, id)`** menjalankan satu percakapan utuh di thread kumpulan, dengan `KnockKnockProtocol` yang *baru*, sehingga setiap klien memiliki keadaannya sendiri. Isinya adalah perulangan Subbab 15.5: kirim baris pembuka, lalu baca, tanyakan kepada protokol,

jawab, sampai `Bye.`. `setSoTimeout(60_000)` mencegah klien yang diam menahan seorang pekerja selamanya, dan `finally` menghitung klien itu bagaimanapun percakapannya berakhir.

- **`stop()`** menutup socket pendengar (`accept()` yang terblokir melempar `SocketException`, mengakhiri perulangan), menunggu thread penerima dengan `join`, menutup setiap socket yang terbuka untuk membangunkan pekerja yang terblokir di `readLine`, lalu mematikan kumpulan thread.

Field yang disentuh beberapa thread dilindungi dengan alat Bab 14: `nextId` dan `clientsServed` bertipe `AtomicInteger`, `openSockets` adalah himpunan konkuren, dan `running` bersifat `volatile`. Selebihnya, terutama objek protokol setiap klien, dimiliki satu thread saja dan tidak memerlukan perlindungan. Prinsipnya: **bagilah sesedikit mungkin, dan lindungi apa yang terpaksa dibagi.** `main` mendaftarkan *shutdown hook*, thread yang dijalankan *runtime* Java ketika program keluar (misalnya setelah Ctrl+C), yang menghentikan server dan mencetak ringkasan.

Kode 15.10 adalah server Python: `ThreadPoolExecutor` adalah kumpulan thread-nya, `threading.Lock` melindungi pencacah dan himpunan socket, dan `threading.Event` memberi tahu perulangan penerima untuk berhenti.

Kode 15.10 `knock_knock_server.py`: server multiklien dalam Python.

```
"""knock_knock_server.py: server knock-knock yang melayani banyak klien sekaligus."""
import socket
import sys
import threading
import time
from concurrent.futures import ThreadPoolExecutor

from knock_knock_protocol import KnockKnockProtocol

LOOPBACK = "127.0.0.1"
IDLE_TIMEOUT_S = 60


class KnockKnockServer:
    def __init__(self, max_clients=8, verbose=True):
        self._pool = ThreadPoolExecutor(max_workers=max_clients)
        self._verbose = verbose
        self._lock = threading.Lock()
        self._next_id = 0
        self._clients_served = 0
        self._open_sockets = set()
        self._stopping = threading.Event()
        self._server_socket = None
        self._accept_thread = None

    def start(self, port):
        """Ikat ke alamat loopback, mulai menerima, dan kembalikan port yang sebenarnya."""
        self._server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        if sys.platform != "win32":  # pakai ulang port segera setelah dijalankan ulang
            self._server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
        self._server_socket.bind((LOOPBACK, port))
        self._server_socket.listen(50)
        self._server_socket.settimeout(0.2)  # bangun secara berkala untuk memeriksa stop()
        self._accept_thread = threading.Thread(target=self._accept_loop, name="acceptor")
        self._accept_thread.start()
        return self._server_socket.getsockname()[1]
```

Kode 15.10 (Lanjutan) *knock_knock_server.py: server multiklien dalam Python.*

```
    def _accept_loop(self):
        while not self._stopping.is_set():
            try:
                conn, _ = self._server_socket.accept()
            except socket.timeout:
                continue
            except OSError:
                break
            with self._lock:
                self._next_id += 1
                client_id = self._next_id
            self._log(f"client #{client_id} connected")
            self._pool.submit(self._handle, conn, client_id)  # serahkan, lalu terima klien
berikutnya

    def _handle(self, conn, client_id):
        with self._lock:
            self._open_sockets.add(conn)
        try:
            conn.settimeout(IDLE_TIMEOUT_S)  # klien yang diam tidak dapat menahan thread
selamanya
            with conn, conn.makefile("r", encoding="utf-8", newline="\n") as reader, \
                    conn.makefile("w", encoding="utf-8", newline="\n") as writer:
                protocol = KnockKnockProtocol()  # satu per percakapan
                writer.write(protocol.process_input(None) + "\n")
                writer.flush()
                for line in reader:
                    reply = protocol.process_input(line.rstrip("\r\n"))
                    writer.write(reply + "\n")
                    writer.flush()
                    if protocol.is_finished(reply):
                        break
        except OSError as e:
            self._log(f"client #{client_id} dropped: {e}")
        finally:
            with self._lock:
                self._open_sockets.discard(conn)
                self._clients_served += 1
            self._log(f"client #{client_id} done")
```

Kode 15.10 (Lanjutan) *knock_knock_server.py: server multiklien dalam Python.*

```
    def stop(self):
        """Berhenti menerima, tutup setiap percakapan yang terbuka, dan tunggu kumpulan
thread."""
        self._stopping.set()
        self._accept_thread.join()
        self._server_socket.close()
        with self._lock:
            sockets = list(self._open_sockets)
        for conn in sockets:
            try:
                conn.shutdown(socket.SHUT_RDWR)  # membangunkan thread yang terblokir di
readline()
            except OSError:
                pass
        self._pool.shutdown(wait=True)

    def clients_served(self):
        with self._lock:
            return self._clients_served

    def _log(self, message):
        if self._verbose:
            print(message, flush=True)


def main():
    port = int(sys.argv[1]) if len(sys.argv) > 1 else 5051
    server = KnockKnockServer(max_clients=8, verbose=True)
    actual = server.start(port)
    print(f"KnockKnockServer listening on port {actual} (Ctrl+C stops it)", flush=True)
    try:
        while True:
            time.sleep(1)  # tidur sampai Ctrl+C
    except KeyboardInterrupt:
        pass
    server.stop()
    print(f"KnockKnockServer stopped after {server.clients_served()} clients")


if __name__ == "__main__":
    main()
```

Satu perbedaan dipaksakan oleh sistem operasi: menutup socket pendengar dari thread lain tidak selalu membangunkan thread Python yang terblokir di `accept()`. Karena itu socket pendengar diberi batas waktu 0,2 detik, dan perulangan memeriksa sinyal berhenti setiap kali batas waktu itu habis. `stop()` membangunkan pekerja yang terblokir dengan `shutdown(SHUT_RDWR)`, yang membuat `readline` mengembalikan `""`. Pustaka standar menyediakan `socketserver.ThreadingTCPServer` yang mengerjakan banyak hal ini untuk Anda; kita menuliskannya sekali agar setiap bagian yang bergerak terlihat.

15.6.3 Klien, dan tiga terminal

Kode 15.11 adalah klien untuk manusia. Server berbicara lebih dahulu, sehingga perulangannya dimulai dengan membaca.

Kode 15.11 `KnockKnockClient.java`: bercakap dengan server knock-knock dari papan ketik.

```java
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.net.ConnectException;
import java.net.Socket;
import java.nio.charset.StandardCharsets;

public class KnockKnockClient {
    public static void main(String[] args) {
        String host = args.length > 0 ? args[0] : "localhost";
        int port = args.length > 1 ? Integer.parseInt(args[1]) : 5051;

        try (Socket socket = new Socket(host, port);
             PrintWriter out = new PrintWriter(new OutputStreamWriter(
                     socket.getOutputStream(), StandardCharsets.UTF_8), true);
             BufferedReader in = new BufferedReader(new InputStreamReader(
                     socket.getInputStream(), StandardCharsets.UTF_8));
             BufferedReader keyboard = new BufferedReader(new InputStreamReader(
                     System.in, StandardCharsets.UTF_8))) {
            String fromServer;
            while ((fromServer = in.readLine()) != null) {  // server berbicara lebih
dahulu
                System.out.println("Server: " + fromServer);
                if (fromServer.equals("Bye.")) {
                    break;
                }
                String fromUser = keyboard.readLine();
                if (fromUser == null) {
                    break;  // akhir masukan: tutup saja sambungannya
                }
                System.out.println("Client: " + fromUser);
                out.println(fromUser);
            }
        } catch (ConnectException e) {
            System.out.println("Cannot connect to " + host + " port " + port
                    + ": is the server running?");
        } catch (IOException e) {
            System.out.println("Connection lost: " + e.getMessage());
        }
    }
}
```

Kode 15.12 adalah klien Python.

Kode 15.12 `knock_knock_client.py`: klien knock-knock dalam Python.

```python
"""knock_knock_client.py: bercakap dengan server knock-knock dari papan ketik."""
import socket
import sys


def main():
    host = sys.argv[1] if len(sys.argv) > 1 else "localhost"
    port = int(sys.argv[2]) if len(sys.argv) > 2 else 5051

    try:
        with socket.create_connection((host, port)) as sock, \
             sock.makefile("w", encoding="utf-8", newline="\n") as writer, \
             sock.makefile("r", encoding="utf-8", newline="\n") as reader:
            for from_server in reader:  # server berbicara lebih dahulu
                from_server = from_server.rstrip("\r\n")
                print("Server: " + from_server)
                if from_server == "Bye.":
                    break
                from_user = sys.stdin.readline()
                if from_user == "":
                    break  # akhir masukan: tutup saja sambungannya
                from_user = from_user.rstrip("\r\n")
                print("Client: " + from_user)
                writer.write(from_user + "\n")
                writer.flush()
    except ConnectionRefusedError:
        print(f"Cannot connect to {host} port {port}: is the server running?")
    except OSError as e:
        print(f"Connection lost: {e}")


if __name__ == "__main__":
    main()
```

Jalankan server dengan `java KnockKnockServer 5051` lalu jalankan klien sebanyak yang Anda suka dari terminal-terminal lain. Untuk buku ini kami menjalankan klien Java lalu klien Python terhadap server Java, masing-masing diberi `KnockKnockClient.input.txt` (di Python, `knock_knock_client.input.txt`), yang memuat satu kesalahan yang disengaja:

```
Who's there?
Dryer who?
Who's there?
Lettuce who?
n
```

Setiap klien mencetak:

```
Server: Knock! Knock!
Client: Who's there?
Server: Lettuce
Client: Dryer who?
Server: You're supposed to say "Lettuce who?"! Try again. Knock! Knock!
Client: Who's there?
Server: Lettuce
Client: Lettuce who?
Server: Lettuce in, it's cold out here! Want another? (y/n)
Client: n
Server: Bye.
```

Terminal server memperlihatkan kedua klien, dan setelah Ctrl+C *shutdown hook*-nya mencetak ringkasan:

```
KnockKnockServer listening on port 5051 (Ctrl+C stops it)
client #1 connected
client #1 done
client #2 connected
client #2 done
KnockKnockServer stopped after 2 clients
```

Server Python, `python3 knock_knock_server.py 5051`, mencetak log yang sama untuk kedua klien yang sama:

```
KnockKnockServer listening on port 5051 (Ctrl+C stops it)
client #1 connected
client #1 done
client #2 connected
client #2 done
KnockKnockServer stopped after 2 clients
```

15.6.4 Membuktikan bahwa server benar-benar konkuren

Tiga terminal hanya memperlihatkan bahwa klien *dapat* menyambung, bukan bahwa mereka dilayani *pada saat yang sama*; server sekuensial akan menghasilkan transkrip yang sama. **Kode 15.13** menguji klaim itu dengan benar. Program ini menjalankan server di dalam program yang sama pada port efemeral, menjalankan tiga klien berskenario di tiga thread, lalu menghentikan server. Setiap klien, begitu disapa, menunggu di sebuah `CyclicBarrier` (Bab 14) sampai *ketiganya* sudah disapa. Server sekuensial tidak akan pernah menyapa klien kedua selagi klien pertama menunggu, penghalangnya akan habis waktu setelah lima detik, dan uji itu akan mencetak `false`.

Kode 15.13 `KnockKnockLoadTest.java`: jalankan server, jalankan tiga klien bersamaan terhadapnya, lalu hentikan.

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.net.InetAddress;
import java.net.Socket;
import java.nio.charset.StandardCharsets;
import java.util.ArrayList;
import java.util.List;
import java.util.concurrent.CyclicBarrier;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.Future;
import java.util.concurrent.TimeUnit;

/** Menjalankan server, menjalankan tiga klien bersamaan terhadapnya, lalu menghentikannya.
*/
public class KnockKnockLoadTest {
    public static void main(String[] args) throws Exception {
        KnockKnockServer server = new KnockKnockServer(3, false);
        int port = server.start(0);  // 0: port efemeral yang dipilih OS
        System.out.println("server started on an ephemeral port");

        List<List<String>> scripts = List.of(
                List.of("Who's there?", "Lettuce who?", "n"),
                List.of("Who's there?", "Lettuce who?", "y", "Who's there?", "Iron who?",
"n"),
                List.of("Hello?", "Who's there?", "Lettuce who?", "n"));
        // Setiap klien menunggu di sini sampai semuanya sudah disapa.
        CyclicBarrier together = new CyclicBarrier(scripts.size());
        ExecutorService clients = Executors.newFixedThreadPool(scripts.size());
        List<Future<List<String>>> results = new ArrayList<>();
        for (List<String> script : scripts) {
            results.add(clients.submit(() -> converse(port, script, together)));
        }

        boolean allGreetedTogether = true;
        for (int i = 0; i < results.size(); i++) {
            System.out.println("--- client " + (char) ('A' + i) + " ---");
            try {
                results.get(i).get().forEach(System.out::println);
            } catch (ExecutionException e) {
                allGreetedTogether = false;
                System.out.println("failed: " + e.getCause().getClass().getSimpleName());
            }
        }
        clients.shutdown();
```

Kode 15.13 (Lanjutan) *KnockKnockLoadTest.java: jalankan server, jalankan tiga klien bersamaan terhadapnya, lalu hentikan.*

```
        server.stop();
        System.out.println("server stopped; clients served: " + server.clientsServed());
        System.out.println("all clients were greeted before any finished: " +
allGreetedTogether);
    }

    static List<String> converse(int port, List<String> script, CyclicBarrier together)
            throws Exception {
        List<String> transcript = new ArrayList<>();
        try (Socket socket = new Socket(InetAddress.getLoopbackAddress(), port);
             BufferedReader in = new BufferedReader(new InputStreamReader(
                     socket.getInputStream(), StandardCharsets.UTF_8));
             PrintWriter out = new PrintWriter(new OutputStreamWriter(
                     socket.getOutputStream(), StandardCharsets.UTF_8), true)) {
            socket.setSoTimeout(5000);
            transcript.add("Server: " + in.readLine());
            together.await(5, TimeUnit.SECONDS);
            for (String mine : script) {
                transcript.add("Client: " + mine);
                out.println(mine);
                transcript.add("Server: " + in.readLine());
            }
        }
        return transcript;
    }
}
```

```
server started on an ephemeral port
--- client A ---
Server: Knock! Knock!
Client: Who's there?
Server: Lettuce
Client: Lettuce who?
Server: Lettuce in, it's cold out here! Want another? (y/n)
Client: n
Server: Bye.
--- client B ---
Server: Knock! Knock!
Client: Who's there?
Server: Lettuce
Client: Lettuce who?
Server: Lettuce in, it's cold out here! Want another? (y/n)
Client: y
Server: Knock! Knock!
Client: Who's there?
Server: Iron
Client: Iron who?
Server: Iron your shirts? We already did! Want another? (y/n)
Client: n
Server: Bye.
--- client C ---
Server: Knock! Knock!
Client: Hello?
Server: You're supposed to say "Who's there?"! Try again. Knock! Knock!
Client: Who's there?
Server: Lettuce
Client: Lettuce who?
Server: Lettuce in, it's cold out here! Want another? (y/n)
Client: n
Server: Bye.
server stopped; clients served: 3
all clients were greeted before any finished: true
```

Transkrip dikumpulkan dengan `Future.get()` dan baru dicetak di akhir, sehingga urutan keluaran tetap walaupun percakapannya bertumpang-tindih dalam waktu. Klien C dibetulkan secara terpisah dari A dan B, yang menegaskan bahwa setiap koneksi memiliki objek protokolnya sendiri. **Kode 15.14** adalah uji versi Python, dengan `threading.Barrier`:

Kode 15.14 `knock_knock_load_test.py`: uji konkurensi yang sama dalam Python.

```
"""knock_knock_load_test.py: jalankan server, jalankan tiga klien sekaligus, hentikan."""
import socket
import threading
from concurrent.futures import ThreadPoolExecutor

from knock_knock_server import KnockKnockServer


def converse(port, script, together):
    transcript = []
    with socket.create_connection(("127.0.0.1", port), timeout=5) as sock, \
         sock.makefile("r", encoding="utf-8", newline="\n") as reader, \
         sock.makefile("w", encoding="utf-8", newline="\n") as writer:
        transcript.append("Server: " + reader.readline().rstrip("\n"))
        together.wait(timeout=5)
        for mine in script:
            transcript.append("Client: " + mine)
            writer.write(mine + "\n")
            writer.flush()
            transcript.append("Server: " + reader.readline().rstrip("\n"))
    return transcript


def main():
    server = KnockKnockServer(max_clients=3, verbose=False)
    port = server.start(0)  # 0: port efemeral yang dipilih OS
    print("server started on an ephemeral port")

    scripts = [
        ["Who's there?", "Lettuce who?", "n"],
        ["Who's there?", "Lettuce who?", "y", "Who's there?", "Iron who?", "n"],
        ["Hello?", "Who's there?", "Lettuce who?", "n"],
    ]
    # Setiap klien menunggu di sini sampai semuanya sudah disapa.
    together = threading.Barrier(len(scripts))
    with ThreadPoolExecutor(max_workers=len(scripts)) as clients:
        futures = [clients.submit(converse, port, script, together) for script in scripts]
```

Kode 15.14 (Lanjutan) *knock_knock_load_test.py: uji konkurensi yang sama dalam Python.*

```
        all_greeted_together = True
        for i, future in enumerate(futures):
            print(f"--- client {chr(ord('A') + i)} ---")
            try:
                for line in future.result():
                    print(line)
            except Exception as e:  # penghalang rusak atau habis waktu
                all_greeted_together = False
                print(f"failed: {type(e).__name__}")
    server.stop()
    print(f"server stopped; clients served: {server.clients_served()}")
    print(f"all clients were greeted before any finished:
{str(all_greeted_together).lower()}")


if __name__ == "__main__":
    main()
```

```
server started on an ephemeral port
--- client A ---
Server: Knock! Knock!
Client: Who's there?
Server: Lettuce
Client: Lettuce who?
Server: Lettuce in, it's cold out here! Want another? (y/n)
Client: n
Server: Bye.
--- client B ---
Server: Knock! Knock!
Client: Who's there?
Server: Lettuce
Client: Lettuce who?
Server: Lettuce in, it's cold out here! Want another? (y/n)
Client: y
Server: Knock! Knock!
Client: Who's there?
Server: Iron
Client: Iron who?
Server: Iron your shirts? We already did! Want another? (y/n)
Client: n
Server: Bye.
--- client C ---
Server: Knock! Knock!
Client: Hello?
Server: You're supposed to say "Who's there?"! Try again. Knock! Knock!
Client: Who's there?
Server: Lettuce
Client: Lettuce who?
Server: Lettuce in, it's cold out here! Want another? (y/n)
Client: n
Server: Bye.
server stopped; clients served: 3
all clients were greeted before any finished: true
```

Soal Latihan 15.5 meminta Anda memperkecil kumpulan menjadi dua pekerja dan meramalkan apa yang terjadi; jawabannya menguji apakah **Gambar 15.6** sudah benar-benar meresap.

15.7 Profitina Laundry: Layanan Status Pesanan dengan JSON di atas TCP

Saatnya membangun sesuatu yang dapat dipakai binatu. Pelanggan terus menelepon meja kasir untuk bertanya apakah cucian mereka sudah siap. Pemilik menginginkan server di ruang belakang yang mengetahui status setiap pesanan dan klien di laptop kasir yang menanyakannya, dan staf harus dapat memajukan status pesanan dari klien mereka sendiri. Server dan klien mungkin ditulis orang yang berbeda dalam bahasa yang berbeda, sehingga protokolnya harus tepat dan netral terhadap bahasa.

15.7.1 Spesifikasi, di atas kertas

Kita mempertahankan satu baris per pesan dan protokol yang dapat diuji tanpa jaringan, dengan satu perubahan: setiap baris membawa satu objek **JSON**.

> **DEFINISI. JSON** (*JavaScript Object Notation*) adalah format teks untuk data terstruktur [11]. Sebuah nilai JSON adalah *objek* `{"key": value, ...}`, *larik* `[value, ...]`, *string* di dalam tanda kutip ganda, *bilangan*, `true`, `false`, atau `null`. Objek dan larik dapat bersarang. Kunci selalu berupa string.

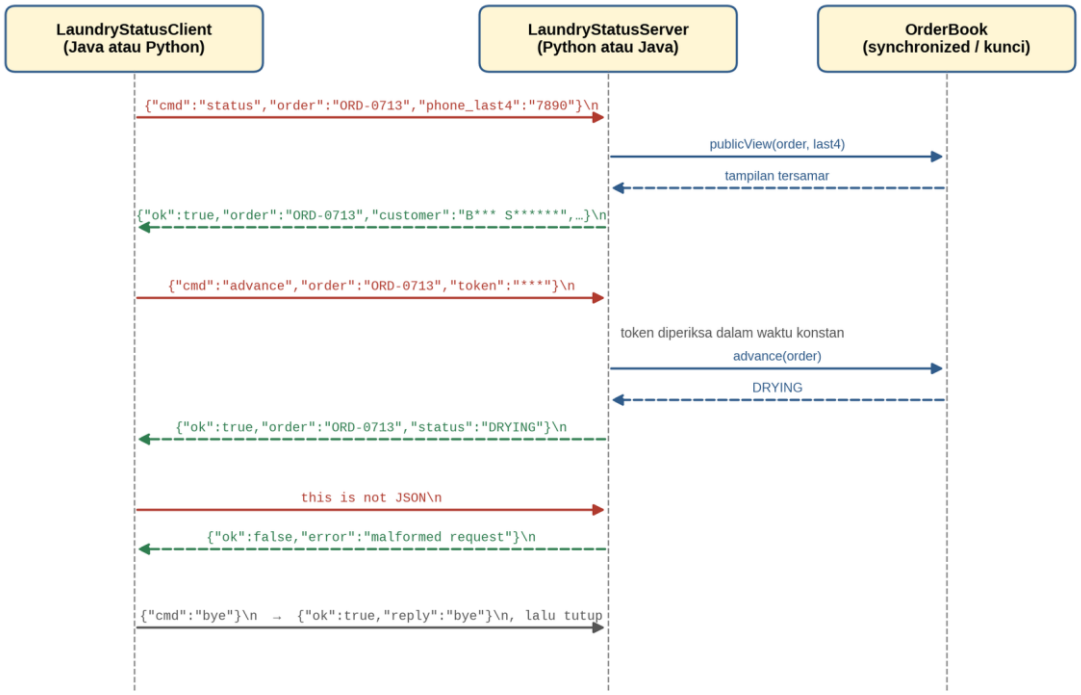
JSON dipahami setiap bahasa modern dan dapat dibaca manusia, dan karena sebuah pesan adalah kumpulan *field* bernama, *field* yang ditambahkan kemudian tidak merusak klien lama. Satu objek JSON per baris, tanpa pemisah baris di dalamnya, adalah konvensi yang disebut *JSON Lines*; ia memadukan struktur JSON dengan pembingkaian baris baru.

Tabel 15.4 adalah spesifikasi lengkapnya. Klien selalu berbicara lebih dahulu, setiap permintaan mendapat tepat satu baris tanggapan, dan setiap tanggapan memuat `ok`; penolakan membawa `error` yang dapat dibaca manusia.

Tabel 15.4 Protokol status pesanan. Satu objek JSON per baris ke setiap arah; klien berbicara lebih dahulu; setiap permintaan menerima tepat satu tanggapan.

Permintaan (klien → server)	Tanggapan berhasil (server → klien)	Penolakan
`{"cmd":"ping"}`	`{"ok":true,"reply":"pong","server":"java"}` (atau `"python"`)	—
`{"cmd":"status","order":"ORD-0713","phone_last4":"7890"}`	`{"ok":true,"order":…,"customer":"B*** S******","status":"WASHING","ready_by":…}`	galat yang sama untuk pesanan tak dikenal dan telepon salah
`{"cmd":"advance","order":…,"token":…}`	`{"ok":true,"order":…,"status":"DRYING"}`	token staf salah atau tidak ada; pesanan sudah diambil; pesanan tidak ada
`{"cmd":"bye"}`	`{"ok":true,"reply":"bye"}`, lalu server menutup	—
`{"cmd":"shutdown","token":…}`	`{"ok":true,"reply":"shutting down"}`, lalu server berhenti	token staf salah atau tidak ada
baris yang bukan objek JSON	—	`{"ok":false,"error":"malformed request"}`
baris yang lebih panjang dari 1024 karakter	—	`{"ok":false,"error":"request too long"}`, lalu tutup

Status mengikuti riwayat hidup di Bab 1, `RECEIVED`, `WASHING`, `DRYING`, `IRONING`, `READY`, `PICKED_UP`, dan `advance` memajukan pesanan satu langkah. **Gambar 15.7** memperlihatkan percakapan yang khas.



Gambar 15.7 Percakapan dengan layanan status pesanan. Permintaan berjalan sebagai baris JSON; server berkonsultasi dengan `OrderBook` bersama yang dijaga oleh kunci; baris yang rusak mendapat pesan galat, bukan kebisuan; `bye` mengakhiri percakapan. Klien dapat berupa Java dan server Python, atau sebaliknya.

Empat pilihan melindungi manusia, bukan program (Subbab 15.9 menjelaskannya): pelanggan harus memberikan empat digit terakhir nomor teleponnya, sehingga struk yang hilang tidak membocorkan apa pun; tanggapan hanya menampilkan inisial, tidak pernah nomor telepon; nomor pesanan yang salah dan nomor telepon yang salah mendapat galat yang *sama*, sehingga penebak tidak dapat mengetahui pesanan mana yang ada; dan perintah staf memerlukan token rahasia yang disimpan dalam variabel lingkungan, tidak pernah di dalam kode.

15.7.2 JSON di kedua bahasa

Python memiliki JSON dalam pustaka standarnya (`json.loads`, `json.dumps`). Pustaka standar Java, secara mengejutkan, tidak memilikinya; proyek sungguhan menambahkan Jackson atau Gson. Untuk menghindari unduhan, dan untuk memperlihatkan bahwa tidak ada sihir di dalamnya, **Kode 15.15** adalah pembaca dan penghasil JSON kecil yang lengkap, kurang dari 300 baris: pengurai *recursive-descent* (turun rekursif) klasik, satu metode untuk setiap jenis nilai, masing-masing memanggil `value()` lagi untuk nilai bersarang.

Kode 15.15 `Json.java`: pembaca dan penghasil JSON minimal; objek menjadi `LinkedHashMap`, larik menjadi `ArrayList`, bilangan bulat menjadi `Long`.

```
import java.util.ArrayList;
import java.util.LinkedHashMap;
import java.util.List;
import java.util.Map;

/**
 * Pembaca dan penulis JSON minimal, cukup untuk protokol berbasis baris.
 * Objek menjadi LinkedHashMap (urutan kunci terjaga), larik menjadi ArrayList,
 * bilangan bulat menjadi Long, bilangan lain menjadi Double.
 */
public final class Json {
    private final String text;
    private int pos = 0;

    private Json(String text) {
        this.text = text;
    }

    /** Mengurai satu nilai JSON; melempar IllegalArgumentException bila teksnya bukan
JSON. */
    public static Object parse(String text) {
        Json parser = new Json(text);
        parser.skipSpaces();
        Object value = parser.value();
        parser.skipSpaces();
        if (parser.pos != text.length()) {
            throw parser.error("extra text after the value");
        }
        return value;
    }

    /** Membangun objek dari kunci dan nilai berselang-seling: object("ok", true, "n", 3).
*/
    public static Map<String, Object> object(Object... keysAndValues) {
        Map<String, Object> map = new LinkedHashMap<>();
        for (int i = 0; i < keysAndValues.length; i += 2) {
            map.put((String) keysAndValues[i], keysAndValues[i + 1]);
        }
        return map;
    }

    /** Menulis nilai secara ringkas, tanpa spasi, seperti json.dumps(separators=(",",
":")) di Python. */
    public static String write(Object value) {
        StringBuilder sb = new StringBuilder();
        writeValue(sb, value);
        return sb.toString();
    }
```

Kode 15.15 (Lanjutan) *Json.java: pembaca dan penghasil JSON minimal; objek menjadi LinkedHashMap, larik menjadi ArrayList, bilangan bulat menjadi Long.*

```
    // ---------- membaca ----------

    private Object value() {
        if (pos >= text.length()) {
            throw error("unexpected end of text");
        }
        char c = text.charAt(pos);
        switch (c) {
            case '{': return object();
            case '[': return array();
            case '"': return string();
            case 't': return literal("true", Boolean.TRUE);
            case 'f': return literal("false", Boolean.FALSE);
            case 'n': return literal("null", null);
            default:
                if (c == '-' || Character.isDigit(c)) {
                    return number();
                }
                throw error("unexpected character '" + c + "'");
        }
    }

    private Map<String, Object> object() {
        Map<String, Object> map = new LinkedHashMap<>();
        pos++;  // lewati {
        skipSpaces();
        if (peek() == '}') {
            pos++;
            return map;
        }
        while (true) {
            skipSpaces();
            if (peek() != '"') {
                throw error("expected a key in double quotes");
            }
            String key = string();
            skipSpaces();
            expect(':');
            skipSpaces();
            map.put(key, value());
            skipSpaces();
            if (peek() == ',') {
                pos++;
            } else {
                expect('}');
                return map;
```

Kode 15.15 (Lanjutan) *Json.java: pembaca dan penghasil JSON minimal; objek menjadi LinkedHashMap, larik menjadi ArrayList, bilangan bulat menjadi Long.*

```
            }
        }
    }

    private List<Object> array() {
        List<Object> list = new ArrayList<>();
        pos++;  // lewati [
        skipSpaces();
        if (peek() == ']') {
            pos++;
            return list;
        }
        while (true) {
            skipSpaces();
            list.add(value());
            skipSpaces();
            if (peek() == ',') {
                pos++;
            } else {
                expect(']');
                return list;
            }
        }
    }

    private String string() {
        StringBuilder sb = new StringBuilder();
        pos++;  // lewati tanda kutip pembuka
        while (true) {
            if (pos >= text.length()) {
                throw error("unterminated string");
            }
            char c = text.charAt(pos++);
            if (c == '"') {
                return sb.toString();
            } else if (c == '\\') {
                if (pos >= text.length()) {
                    throw error("unterminated escape");
                }
                char e = text.charAt(pos++);
                switch (e) {
                    case '"': sb.append('"'); break;
                    case '\\': sb.append('\\'); break;
                    case '/': sb.append('/'); break;
                    case 'b': sb.append('\b'); break;
                    case 'f': sb.append('\f'); break;
```

Kode 15.15 (Lanjutan) *Json.java: pembaca dan penghasil JSON minimal; objek menjadi LinkedHashMap, larik menjadi ArrayList, bilangan bulat menjadi Long.*

```
                    case 'n': sb.append('\n'); break;
                    case 'r': sb.append('\r'); break;
                    case 't': sb.append('\t'); break;
                    case 'u':
                        if (pos + 4 > text.length()) {
                            throw error("short \\u escape");
                        }
                        sb.append((char) Integer.parseInt(text.substring(pos, pos + 4),
16));
                        pos += 4;
                        break;
                    default: throw error("bad escape \\" + e);
                }
            } else if (c < 0x20) {
                throw error("control character inside a string");
            } else {
                sb.append(c);
            }
        }
    }

    private Object number() {
        int start = pos;
        if (peek() == '-') {
            pos++;
        }
        while (pos < text.length() && "0123456789.eE+-".indexOf(text.charAt(pos)) >= 0) {
            pos++;
        }
        String s = text.substring(start, pos);
        try {
            if (s.contains(".") || s.contains("e") || s.contains("E")) {
                return Double.parseDouble(s);
            }
            return Long.parseLong(s);
        } catch (NumberFormatException e) {
            throw error("bad number " + s);
        }
    }
```

Kode 15.15 (Lanjutan) *Json.java: pembaca dan penghasil JSON minimal; objek menjadi LinkedHashMap, larik menjadi ArrayList, bilangan bulat menjadi Long.*

```
    private Object literal(String word, Object result) {
        if (!text.startsWith(word, pos)) {
            throw error("expected " + word);
        }
        pos += word.length();
        return result;
    }

    private char peek() {
        return pos < text.length() ? text.charAt(pos) : '\0';
    }

    private void expect(char c) {
        if (peek() != c) {
            throw error("expected '" + c + "'");
        }
        pos++;
    }

    private void skipSpaces() {
        while (pos < text.length() && " \t\r\n".indexOf(text.charAt(pos)) >= 0) {
            pos++;
        }
    }

    private IllegalArgumentException error(String message) {
        return new IllegalArgumentException("malformed JSON at position " + pos + ": " +
message);
    }

    // ---------- menulis ----------

    private static void writeValue(StringBuilder sb, Object value) {
        if (value == null) {
            sb.append("null");
        } else if (value instanceof String s) {
            writeString(sb, s);
        } else if (value instanceof Boolean || value instanceof Long || value instanceof
Integer
                || value instanceof Double) {
            sb.append(value);
        } else if (value instanceof Map<?, ?> map) {
            sb.append('{');
            boolean first = true;
            for (Map.Entry<?, ?> entry : map.entrySet()) {
                if (!first) {
                    sb.append(',');
                }
```

Kode 15.15 (Lanjutan) *Json.java: pembaca dan penghasil JSON minimal; objek menjadi LinkedHashMap, larik menjadi ArrayList, bilangan bulat menjadi Long.*

```
                first = false;
                writeString(sb, (String) entry.getKey());
                sb.append(':');
                writeValue(sb, entry.getValue());
            }
            sb.append('}');
        } else if (value instanceof List<?> list) {
            sb.append('[');
            for (int i = 0; i < list.size(); i++) {
                if (i > 0) {
                    sb.append(',');
                }
                writeValue(sb, list.get(i));
            }
            sb.append(']');
        } else {
            throw new IllegalArgumentException("cannot write " +
value.getClass().getSimpleName());
        }
    }

    private static void writeString(StringBuilder sb, String s) {
        sb.append('"');
        for (char c : s.toCharArray()) {
            switch (c) {
                case '"': sb.append("\\\""); break;
                case '\\': sb.append("\\\\"); break;
                case '\n': sb.append("\\n"); break;
                case '\r': sb.append("\\r"); break;
                case '\t': sb.append("\\t"); break;
                case '\b': sb.append("\\b"); break;
                case '\f': sb.append("\\f"); break;
                default:
                    if (c < 0x20) {
                        sb.append(String.format("\\u%04x", (int) c));
                    } else {
                        sb.append(c);
                    }
            }
        }
        sb.append('"');
    }
```

Kode 15.15 (Lanjutan) *Json.java: pembaca dan penghasil JSON minimal; objek menjadi LinkedHashMap, larik menjadi ArrayList, bilangan bulat menjadi Long.*

```
    public static void main(String[] args) {
        String[] samples = {
            "{\"cmd\":\"status\",\"order\":\"ORD-0713\",\"phone_last4\":\"7890\"}",
            "{ \"ok\" : true, \"kg\" : 3.5, \"price\" : 24500, \"items\" : [\"shirt\",
\"towel\"] }",
            "{\"note\":\"line 1\\nline 2 \\\"quoted\\\" caf\\u00e9\", \"paid\":null}",
            "{\"cmd\":\"status\",\"order\":}",
            "{\"cmd\":\"ping\"} extra"
        };
        for (String sample : samples) {
            System.out.println("text   : " + sample);
            try {
                Object value = parse(sample);
                System.out.println("written: " + write(value));
            } catch (IllegalArgumentException e) {
                System.out.println("rejected: not valid JSON");
            }
        }
    }
}
```

```
text   : {"cmd":"status","order":"ORD-0713","phone_last4":"7890"}
written: {"cmd":"status","order":"ORD-0713","phone_last4":"7890"}
text   : { "ok" : true, "kg" : 3.5, "price" : 24500, "items" : ["shirt", "towel"] }
written: {"ok":true,"kg":3.5,"price":24500,"items":["shirt","towel"]}
text   : {"note":"line 1\nline 2 \"quoted\" caf\u00e9", "paid":null}
written: {"note":"line 1\nline 2 \"quoted\" café","paid":null}
text   : {"cmd":"status","order":}
rejected: not valid JSON
text   : {"cmd":"ping"} extra
rejected: not valid JSON
```

Kelas ini menawarkan tiga metode statis. `parse(text)` mengembalikan `Map`, `List`, `String`, `Long`, `Double`, `Boolean`, atau `null`, dan melempar `IllegalArgumentException` untuk apa pun yang bukan tepat satu nilai JSON (contoh keempat kekurangan nilai, contoh kelima memiliki teks tambahan). `write(value)` menghasilkan teks ringkas tanpa spasi. `object(...)` membangun pesan: `Json.object("cmd", "ping")`. Peta berupa `LinkedHashMap`, sehingga *field* ditulis sesuai urutan penyisipan. Pembaca mendekode *escape* seperti `\n`, `\"`, dan `\u00e9` (é), dan bagian penghasilnya membuatnya, yang menjamin bahwa sebuah pesan tidak pernah memuat baris baru mentah.

Kode 15.16 melakukan perjalanan bolak-balik yang sama dengan modul `json` Python.

Kode 15.16 `json_roundtrip.py`: perjalanan bolak-balik yang sama dengan modul standar `json` Python.

```
"""json_roundtrip.py: baca dan tulis JSON protokol dengan modul standar json."""
import json


def write(value):
    """Tulis secara ringkas, tanpa spasi, persis seperti Json.write di Java."""
    return json.dumps(value, separators=(",", ":"), ensure_ascii=False)


def main():
    samples = [
        '{"cmd":"status","order":"ORD-0713","phone_last4":"7890"}',
        '{ "ok" : true, "kg" : 3.5, "price" : 24500, "items" : ["shirt", "towel"] }',
        '{"note":"line 1\\nline 2 \\"quoted\\" caf\\u00e9", "paid":null}',
        '{"cmd":"status","order":}',
        '{"cmd":"ping"} extra',
    ]
    for sample in samples:
        print("text   : " + sample)
        try:
            value = json.loads(sample)
            print("written: " + write(value))
        except json.JSONDecodeError:
            print("rejected: not valid JSON")


if __name__ == "__main__":
    main()
```

```
text   : {"cmd":"status","order":"ORD-0713","phone_last4":"7890"}
written: {"cmd":"status","order":"ORD-0713","phone_last4":"7890"}
text   : { "ok" : true, "kg" : 3.5, "price" : 24500, "items" : ["shirt", "towel"] }
written: {"ok":true,"kg":3.5,"price":24500,"items":["shirt","towel"]}
text   : {"note":"line 1\nline 2 \"quoted\" caf\u00e9", "paid":null}
written: {"note":"line 1\nline 2 \"quoted\" café","paid":null}
text   : {"cmd":"status","order":}
rejected: not valid JSON
text   : {"cmd":"ping"} extra
rejected: not valid JSON
```

Kedua keluaran identik, dan itu adalah keharusan, bukan kebetulan: `separators=(",", ":")` membuat Python menulis tanpa spasi, seperti `Json.write`, dan `ensure_ascii=False` menulis `é` apa adanya. Karena kedua pihak menulis teks yang sama untuk nilai yang sama, masing-masing dapat membaca pesan pihak lain. (Bila `é` tampil sebagai `?`, terminal Anda tidak memakai UTF-8; di Windows, `chcp 65001` memperbaikinya.)

15.7.3 Server

Kode 15.17 adalah servernya: kerangka server knock-knock (socket pendengar, thread penerima, kumpulan thread, `stop` yang bersih) ditambah tiga kelas domain di berkas yang sama, `Status`, `LaundryOrder`, dan `OrderBook`.

Kode 15.17 `LaundryStatusServer.java`: layanan status pesanan Profitina Laundry, dengan kelas-kelas domainnya `Status`, `LaundryOrder`, dan `OrderBook`.

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.io.Reader;
import java.net.InetAddress;
import java.net.ServerSocket;
import java.net.Socket;
import java.net.SocketException;
import java.nio.charset.StandardCharsets;
import java.security.MessageDigest;
import java.util.HashMap;
import java.util.Map;
import java.util.Set;
import java.util.concurrent.ConcurrentHashMap;
import java.util.concurrent.CountDownLatch;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.atomic.AtomicInteger;

/** Layanan status pesanan Profitina Laundry: satu objek JSON per baris, dua arah. */
public class LaundryStatusServer {
    static final int MAX_LINE = 1024;
    private static final int IDLE_TIMEOUT_MS = 60_000;

    private final OrderBook book;
    private final String staffToken;  // null: perintah staf dinonaktifkan
    private final boolean verbose;
    private final ExecutorService pool = Executors.newFixedThreadPool(16);
    private final AtomicInteger nextId = new AtomicInteger();
    private final Set<Socket> openSockets = ConcurrentHashMap.newKeySet();
    private final CountDownLatch stopped = new CountDownLatch(1);
    private ServerSocket serverSocket;
    private volatile boolean running;

    LaundryStatusServer(OrderBook book, String staffToken, boolean verbose) {
        this.book = book;
        this.staffToken = staffToken;
        this.verbose = verbose;
    }
```

Kode 15.17 (Lanjutan) *LaundryStatusServer.java: layanan status pesanan Profitina Laundry, dengan kelas-kelas domainnya Status, LaundryOrder, dan OrderBook.*

```
    /** Server yang memuat tiga pesanan contoh yang dipakai di sepanjang bab ini. */
    public static LaundryStatusServer withSampleOrders(String staffToken, boolean verbose)
{
        return new LaundryStatusServer(OrderBook.sample(), staffToken, verbose);
    }

    public int start(int port) throws IOException {
        serverSocket = new ServerSocket(port, 50, InetAddress.getLoopbackAddress());
        running = true;
        new Thread(this::acceptLoop, "acceptor").start();
        return serverSocket.getLocalPort();
    }

    private void acceptLoop() {
        while (running) {
            try {
                Socket socket = serverSocket.accept();
                int id = nextId.incrementAndGet();
                pool.execute(() -> serve(socket, id));
            } catch (SocketException e) {
                break;  // stop() telah menutup socket server
            } catch (IOException e) {
                log("accept failed: " + e.getMessage());
            }
        }
    }

    private void serve(Socket socket, int id) {
        openSockets.add(socket);
        log("client #" + id + " connected");
        try (socket;
             Reader in = new BufferedReader(new InputStreamReader(
                     socket.getInputStream(), StandardCharsets.UTF_8));
             PrintWriter out = new PrintWriter(new OutputStreamWriter(
                     socket.getOutputStream(), StandardCharsets.UTF_8), true)) {
            socket.setSoTimeout(IDLE_TIMEOUT_MS);
            String line;
            while ((line = readLimitedLine(in)) != null) {
                if (line.length() > MAX_LINE) {
                    out.println(Json.write(error("request too long")));
                    break;  // jangan membaca sisa banjiran data
                }
                Map<String, Object> request = parseRequest(line);
                String cmd = request == null ? "?" : String.valueOf(request.get("cmd"));
                Map<String, Object> response = handle(request);
                out.println(Json.write(response));
                log("client #" + id + ": " + cmd + " -> " +
(response.get("ok").equals(true)
```

Kode 15.17 (Lanjutan) *LaundryStatusServer.java: layanan status pesanan Profitina Laundry, dengan kelas-kelas domainnya Status, LaundryOrder, dan OrderBook.*

```
                        ? "ok" : "refused"));  // jangan pernah mencatat data pribadi
                if (cmd.equals("bye")) {
                    break;
                }
                if (cmd.equals("shutdown") && response.get("ok").equals(true)) {
                    new Thread(this::stop).start();  // bukan dari thread kumpulan: stop()
menunggu kumpulan
                    break;
                }
            }
        } catch (IOException e) {
            log("client #" + id + " dropped: " + e.getMessage());
        } finally {
            openSockets.remove(socket);
            log("client #" + id + " disconnected");
        }
    }

    /** Membaca satu baris, tetapi berhenti membaca setelah MAX_LINE + 1 karakter. */
    static String readLimitedLine(Reader in) throws IOException {
        StringBuilder sb = new StringBuilder();
        int c;
        while ((c = in.read()) != -1) {
            if (c == '\n') {
                return sb.toString();
            }
            if (c != '\r') {
                sb.append((char) c);
            }
            if (sb.length() > MAX_LINE) {
                return sb.toString();  // terlalu panjang: pemanggil menolaknya
            }
        }
        return sb.length() > 0 ? sb.toString() : null;
    }

    @SuppressWarnings("unchecked")
    private static Map<String, Object> parseRequest(String line) {
        try {
            Object value = Json.parse(line);
            return value instanceof Map ? (Map<String, Object>) value : null;
        } catch (IllegalArgumentException e) {
            return null;
        }
    }
```

Kode 15.17 (Lanjutan) *LaundryStatusServer.java: layanan status pesanan Profitina Laundry, dengan kelas-kelas domainnya Status, LaundryOrder, dan OrderBook.*

```
    /** Seluruh protokol, sebagai fungsi dari permintaan ke tanggapan. */
    Map<String, Object> handle(Map<String, Object> request) {
        if (request == null) {
            return error("malformed request");
        }
        Object cmd = request.get("cmd");
        if ("ping".equals(cmd)) {
            return Json.object("ok", true, "reply", "pong", "server", "java");
        } else if ("status".equals(cmd)) {
            Object order = request.get("order");
            Object last4 = request.get("phone_last4");
            if (!(order instanceof String) || !(last4 instanceof String)) {
                return error("status needs order and phone_last4");
            }
            Map<String, Object> view = book.publicView((String) order, (String) last4);
            // Jawaban sama untuk "pesanan tidak ada" dan "telepon salah": tanpa petunjuk
bagi penebak.
            return view != null ? view : error("no order with that number and phone");
        } else if ("advance".equals(cmd)) {
            if (!isStaff(request.get("token"))) {
                return error("not authorized");
            }
            Object order = request.get("order");
            if (!(order instanceof String)) {
                return error("advance needs order");
            }
            try {
                Status status = book.advance((String) order);
                return Json.object("ok", true, "order", order, "status", status.name());
            } catch (IllegalArgumentException | IllegalStateException e) {
                return error(e.getMessage());
            }
        } else if ("bye".equals(cmd)) {
            return Json.object("ok", true, "reply", "bye");
        } else if ("shutdown".equals(cmd)) {
            return isStaff(request.get("token"))
                    ? Json.object("ok", true, "reply", "shutting down")
                    : error("not authorized");
        }
        return error("unknown command: " + cmd);
    }
```

Kode 15.17 (Lanjutan) *LaundryStatusServer.java: layanan status pesanan Profitina Laundry, dengan kelas-kelas domainnya Status, LaundryOrder, dan OrderBook.*

```
    private boolean isStaff(Object token) {
        if (staffToken == null || !(token instanceof String)) {
            return false;
        }
        // Perbandingan waktu-konstan: lamanya waktu tidak membocorkan seberapa banyak yang
cocok.
        return MessageDigest.isEqual(staffToken.getBytes(StandardCharsets.UTF_8),
                ((String) token).getBytes(StandardCharsets.UTF_8));
    }

    private static Map<String, Object> error(String message) {
        return Json.object("ok", false, "error", message);
    }

    public void stop() {
        if (!running) {
            return;
        }
        running = false;
        try {
            serverSocket.close();
        } catch (IOException ignored) {
            // penutupan dilakukan sebisanya
        }
        for (Socket socket : openSockets) {
            try {
                socket.close();
            } catch (IOException ignored) {
                // penutupan dilakukan sebisanya
            }
        }
        pool.shutdown();
        try {
            pool.awaitTermination(5, TimeUnit.SECONDS);
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
        }
        stopped.countDown();
    }

    public void awaitStop() throws InterruptedException {
        stopped.await();
    }
```

Kode 15.17 (Lanjutan) *LaundryStatusServer.java: layanan status pesanan Profitina Laundry, dengan kelas-kelas domainnya Status, LaundryOrder, dan OrderBook.*

```
    private void log(String message) {
        if (verbose) {
            System.out.println(message);
        }
    }

    public static void main(String[] args) throws Exception {
        int port = args.length > 0 ? Integer.parseInt(args[0]) : 5052;
        String token = System.getenv("PROFITINA_STAFF_TOKEN");  // rahasia tidak pernah
tinggal di dalam kode
        boolean verbose = !(args.length > 1 && args[1].equals("--quiet"));
        LaundryStatusServer server = withSampleOrders(token, verbose);
        int actual = server.start(port);
        System.out.println("LaundryStatusServer (java) ready on port " + actual);
        if (token == null) {
            System.out.println("staff commands disabled: PROFITINA_STAFF_TOKEN is not
set");
        }
        Runtime.getRuntime().addShutdownHook(new Thread(server::stop));  // Ctrl+C
        server.awaitStop();
        System.out.println("LaundryStatusServer (java) stopped");
    }
}

enum Status {
    RECEIVED, WASHING, DRYING, IRONING, READY, PICKED_UP;

    Status next() {
        if (this == PICKED_UP) {
            throw new IllegalStateException("order already picked up");
        }
        return values()[ordinal() + 1];
    }
}

class LaundryOrder {
    final String id;
    final String customer;
    final String phone;
    final String readyBy;
    Status status;
```

Kode 15.17 (Lanjutan) *LaundryStatusServer.java: layanan status pesanan Profitina Laundry, dengan kelas-kelas domainnya Status, LaundryOrder, dan OrderBook.*

```
    LaundryOrder(String id, String customer, String phone, String readyBy, Status status) {
        this.id = id;
        this.customer = customer;
        this.phone = phone;
        this.readyBy = readyBy;
        this.status = status;
    }
}

/** Semua pesanan; setiap metode synchronized karena banyak thread klien berbagi satu buku.
*/
class OrderBook {
    private final Map<String, LaundryOrder> orders = new HashMap<>();

    synchronized void add(LaundryOrder order) {
        orders.put(order.id, order);
    }

    /** Yang boleh dilihat pelanggan: tanpa nomor telepon, dan hanya inisial nama. */
    synchronized Map<String, Object> publicView(String id, String phoneLast4) {
        LaundryOrder o = orders.get(id);
        if (o == null || !o.phone.endsWith(phoneLast4) || phoneLast4.length() != 4) {
            return null;
        }
        return Json.object("ok", true, "order", o.id, "customer", mask(o.customer),
                "status", o.status.name(), "ready_by", o.readyBy);
    }

    synchronized Status advance(String id) {
        LaundryOrder o = orders.get(id);
        if (o == null) {
            throw new IllegalArgumentException("no such order");
        }
        o.status = o.status.next();
        return o.status;
    }
```

Kode 15.17 (Lanjutan) *LaundryStatusServer.java: layanan status pesanan Profitina Laundry, dengan kelas-kelas domainnya Status, LaundryOrder, dan OrderBook.*

```
    static String mask(String name) {
        StringBuilder sb = new StringBuilder();
        for (String word : name.split(" ")) {
            if (sb.length() > 0) {
                sb.append(' ');
            }
            sb.append(word.charAt(0)).append("*".repeat(word.length() - 1));
        }
        return sb.toString();
    }

    static OrderBook sample() {
        OrderBook book = new OrderBook();
        book.add(new LaundryOrder("ORD-0713", "Budi Santoso", "081234567890",
                "2026-09-24 16:00", Status.WASHING));
        book.add(new LaundryOrder("ORD-0714", "Sari Wulandari", "081298765432",
                "2026-09-23 18:00", Status.READY));
        book.add(new LaundryOrder("ORD-0715", "Andi Pratama", "085711112222",
                "2026-09-25 10:00", Status.RECEIVED));
        return book;
    }
}
```

Dijalankan dengan token staf di lingkungannya (Subbab 15.7.4), dikunjungi satu klien yang menjalankan skenario **Kode 15.19**, lalu dihentikan dengan Ctrl+C, server mencetak:

```
LaundryStatusServer (java) ready on port 5052
client #1 connected
client #1: ping -> ok
client #1: status -> ok
client #1: status -> refused
client #1: status -> refused
client #1: ? -> refused
client #1: advance -> refused
client #1: advance -> ok
client #1: status -> ok
client #1: bye -> ok
client #1 disconnected
LaundryStatusServer (java) stopped
```

- **`serve(socket, id)`** adalah perulangan per klien. `readLimitedLine` menyerah setelah 1025 karakter, karena `readLine` tidak berbatas dan klien jahat dapat mengirim satu gigabita tanpa baris baru; baris yang terlalu panjang ditolak dan koneksinya ditutup. Baris yang bukan objek JSON menjadi `null`, yang dijawab `handle` dengan `malformed request`, sehingga klien selalu mendapat jawaban. Log mencatat perintah dan hasilnya, tidak pernah nomor pesanan, nama, atau digit telepon.
- **`handle(request)`** adalah seluruh **Tabel 15.4** sebagai satu fungsi dari peta permintaan ke peta tanggapan, tanpa I/O, sehingga dapat diuji tanpa jaringan. Setiap *field* diperiksa keberadaan dan tipenya (`instanceof String`) sebelum dipakai: `{"cmd":"status","order":42}` harus ditolak, bukan membuat pekerja tumbang karena `ClassCastException`.
- **`isStaff(token)`** membandingkan dengan `MessageDigest.isEqual`, yang memakan waktu sama di mana pun letak perbedaannya; `equals` berhenti pada perbedaan pertama, dan mengukur waktu banyak percobaan dapat membocorkan rahasia karakter demi karakter. Token berasal dari variabel lingkungan `PROFITINA_STAFF_TOKEN`; bila tidak diisi, setiap perintah staf ditolak.
- **`shutdown`** tidak boleh memanggil `stop()` di thread pekerja yang menerimanya: `stop()` menunggu kumpulan thread, dan pekerja itu *adalah* anggota kumpulan, sehingga ia akan menunggu dirinya sendiri selamanya. Ia menjalankan thread baru untuk memanggil `stop()`.
- **`OrderBook`** menyimpan pesanan dalam `HashMap`, yang tidak aman dipakai bersamaan, sehingga setiap metodenya `synchronized` (Bab 14). `publicView` membangun tampilan untuk pelanggan, dengan `mask` mengubah "Budi Santoso" menjadi "B*** S******"; `advance` memakai `Status.next()`, yang melempar eksepsi pada `PICKED_UP`. Kelas-kelas pembantu yang berakses paket ini dijangkau dari berkas lain melalui metode pabrik `withSampleOrders`.

`main` mencetak baris "ready", mendaftarkan *shutdown hook* untuk Ctrl+C, dan menunggu pada `CountDownLatch` yang dilepas oleh `stop()`, sehingga program berakhir setelah perintah `shutdown` atau Ctrl+C. Tanpa token, server mengatakannya saat mulai:

```
LaundryStatusServer (java) ready on port 5052
staff commands disabled: PROFITINA_STAFF_TOKEN is not set
LaundryStatusServer (java) stopped
```

Kode 15.18 adalah server Python, kelas demi kelas; `dataclass` menuliskan konstruktor `LaundryOrder`, `threading.Lock` menggantikan `synchronized`, dan `hmac.compare_digest` adalah perbandingan waktu-konstannya.

Kode 15.18 `laundry_status_server.py`: layanan status pesanan dalam Python.

```
"""laundry_status_server.py: layanan status pesanan Profitina Laundry.

Satu objek JSON per baris, dua arah. Jalankan: python3 laundry_status_server.py [port] [--
quiet]
"""
import hmac
import json
import os
import socket
import sys
import threading
from concurrent.futures import ThreadPoolExecutor
from dataclasses import dataclass
from enum import Enum

LOOPBACK = "127.0.0.1"
MAX_LINE = 1024
IDLE_TIMEOUT_S = 60


class Status(Enum):
    RECEIVED = 0
    WASHING = 1
    DRYING = 2
    IRONING = 3
    READY = 4
    PICKED_UP = 5

    def next(self):
        if self is Status.PICKED_UP:
            raise ValueError("order already picked up")
        return Status(self.value + 1)


@dataclass
class LaundryOrder:
    id: str
    customer: str
    phone: str
    ready_by: str
    status: Status


def mask(name):
    return " ".join(word[0] + "*" * (len(word) - 1) for word in name.split(" "))
```

Kode 15.18 (Lanjutan) *laundry_status_server.py: layanan status pesanan dalam Python.*

```
class OrderBook:
    """Semua pesanan; setiap metode mengambil kunci karena banyak thread klien berbagi satu buku."""

    def __init__(self):
        self._orders = {}
        self._lock = threading.Lock()

    def add(self, order):
        with self._lock:
            self._orders[order.id] = order

    def public_view(self, order_id, phone_last4):
        """Yang boleh dilihat pelanggan: tanpa nomor telepon, dan hanya inisial nama."""
        with self._lock:
            o = self._orders.get(order_id)
            if o is None or len(phone_last4) != 4 or not o.phone.endswith(phone_last4):
                return None
            return {"ok": True, "order": o.id, "customer": mask(o.customer),
                    "status": o.status.name, "ready_by": o.ready_by}

    def advance(self, order_id):
        with self._lock:
            o = self._orders.get(order_id)
            if o is None:
                raise ValueError("no such order")
            o.status = o.status.next()
            return o.status

    @staticmethod
    def sample():
        book = OrderBook()
        book.add(LaundryOrder("ORD-0713", "Budi Santoso", "081234567890",
                              "2026-09-24 16:00", Status.WASHING))
        book.add(LaundryOrder("ORD-0714", "Sari Wulandari", "081298765432",
                              "2026-09-23 18:00", Status.READY))
        book.add(LaundryOrder("ORD-0715", "Andi Pratama", "085711112222",
                              "2026-09-25 10:00", Status.RECEIVED))
        return book


def write(value):
    return json.dumps(value, separators=(",", ":"), ensure_ascii=False)
```

Kode 15.18 (Lanjutan) *laundry_status_server.py: layanan status pesanan dalam Python.*

```
def error(message):
    return {"ok": False, "error": message}


class LaundryStatusServer:
    def __init__(self, book, staff_token, verbose=True):
        self._book = book
        self._staff_token = staff_token  # None: perintah staf dinonaktifkan
        self._verbose = verbose
        self._pool = ThreadPoolExecutor(max_workers=16)
        self._lock = threading.Lock()
        self._next_id = 0
        self._open_sockets = set()
        self._stopping = threading.Event()
        self._stopped = threading.Event()
        self._server_socket = None
        self._accept_thread = None

    def start(self, port):
        self._server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        if sys.platform != "win32":  # pakai ulang port segera setelah dijalankan ulang
            self._server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
        self._server_socket.bind((LOOPBACK, port))
        self._server_socket.listen(50)
        self._server_socket.settimeout(0.2)  # bangun secara berkala untuk memeriksa stop()
        self._accept_thread = threading.Thread(target=self._accept_loop, name="acceptor")
        self._accept_thread.start()
        return self._server_socket.getsockname()[1]

    def _accept_loop(self):
        while not self._stopping.is_set():
            try:
                conn, _ = self._server_socket.accept()
            except socket.timeout:
                continue
            except OSError:
                break
            with self._lock:
                self._next_id += 1
                client_id = self._next_id
            self._pool.submit(self._serve, conn, client_id)
```

Kode 15.18 (Lanjutan) *laundry_status_server.py: layanan status pesanan dalam Python.*

```
    def _serve(self, conn, client_id):
        with self._lock:
            self._open_sockets.add(conn)
        self._log(f"client #{client_id} connected")
        try:
            conn.settimeout(IDLE_TIMEOUT_S)
            with conn, conn.makefile("r", encoding="utf-8", newline="\n") as reader, \
                    conn.makefile("w", encoding="utf-8", newline="\n") as writer:
                while True:
                    line = reader.readline(MAX_LINE + 1)  # jangan pernah membaca lebih
dari ini
                    if line == "":
                        break  # klien menutup koneksi
                    line = line.rstrip("\r\n")
                    if len(line) > MAX_LINE:
                        writer.write(write(error("request too long")) + "\n")
                        writer.flush()
                        break  # jangan membaca sisa banjiran data
                    request = parse_request(line)
                    cmd = "?" if request is None else str(request.get("cmd"))
                    response = self.handle(request)
                    writer.write(write(response) + "\n")
                    writer.flush()
                    outcome = "ok" if response["ok"] else "refused"
                    self._log(f"client #{client_id}: {cmd} -> {outcome}")  # jangan pernah
mencatat data pribadi
                    if cmd == "bye":
                        break
                    if cmd == "shutdown" and response["ok"]:
                        # bukan dari thread kumpulan: stop() menunggu kumpulan
                        threading.Thread(target=self.stop).start()
                        break
        except OSError as e:
            self._log(f"client #{client_id} dropped: {e}")
        finally:
            with self._lock:
                self._open_sockets.discard(conn)
            self._log(f"client #{client_id} disconnected")
```

Kode 15.18 (Lanjutan) *laundry_status_server.py: layanan status pesanan dalam Python.*

```
    def handle(self, request):
        """Seluruh protokol, sebagai fungsi dari permintaan ke tanggapan."""
        if request is None:
            return error("malformed request")
        cmd = request.get("cmd")
        if cmd == "ping":
            return {"ok": True, "reply": "pong", "server": "python"}
        if cmd == "status":
            order, last4 = request.get("order"), request.get("phone_last4")
            if not isinstance(order, str) or not isinstance(last4, str):
                return error("status needs order and phone_last4")
            view = self._book.public_view(order, last4)
            # Jawaban sama untuk "pesanan tidak ada" dan "telepon salah": tanpa petunjuk
bagi penebak.
            return view if view is not None else error("no order with that number and
phone")
        if cmd == "advance":
            if not self._is_staff(request.get("token")):
                return error("not authorized")
            order = request.get("order")
            if not isinstance(order, str):
                return error("advance needs order")
            try:
                status = self._book.advance(order)
            except ValueError as e:
                return error(str(e))
            return {"ok": True, "order": order, "status": status.name}
        if cmd == "bye":
            return {"ok": True, "reply": "bye"}
        if cmd == "shutdown":
            if self._is_staff(request.get("token")):
                return {"ok": True, "reply": "shutting down"}
            return error("not authorized")
        return error(f"unknown command: {'null' if cmd is None else cmd}")

    def _is_staff(self, token):
        if self._staff_token is None or not isinstance(token, str):
            return False
        # Perbandingan waktu-konstan: lamanya waktu tidak membocorkan seberapa banyak yang
cocok.
        return hmac.compare_digest(self._staff_token.encode("utf-8"), token.encode("utf-
8"))
```

Kode 15.18 (Lanjutan) *laundry_status_server.py: layanan status pesanan dalam Python.*

```
    def stop(self):
        if self._stopping.is_set():
            return
        self._stopping.set()
        self._accept_thread.join()
        self._server_socket.close()
        with self._lock:
            sockets = list(self._open_sockets)
        for conn in sockets:
            try:
                conn.shutdown(socket.SHUT_RDWR)
            except OSError:
                pass
        self._pool.shutdown(wait=True)
        self._stopped.set()

    def wait_until_stopped(self):
        while not self._stopped.wait(0.5):  # penantian singkat menjaga Ctrl+C tetap
responsif
            pass

    def _log(self, message):
        if self._verbose:
            print(message, flush=True)


def parse_request(line):
    try:
        value = json.loads(line)
    except json.JSONDecodeError:
        return None
    return value if isinstance(value, dict) else None


def main():
    args = sys.argv[1:]
    port = int(args[0]) if args else 5052
    verbose = "--quiet" not in args
    token = os.environ.get("PROFITINA_STAFF_TOKEN")  # rahasia tidak pernah tinggal di
dalam kode
    server = LaundryStatusServer(OrderBook.sample(), token, verbose)
    actual = server.start(port)
    print(f"LaundryStatusServer (python) ready on port {actual}", flush=True)
    if token is None:
        print("staff commands disabled: PROFITINA_STAFF_TOKEN is not set", flush=True)
    try:
        server.wait_until_stopped()
    except KeyboardInterrupt:  # Ctrl+C
```

Kode 15.18 (Lanjutan) *laundry_status_server.py: layanan status pesanan dalam Python.*

```
        server.stop()
    print("LaundryStatusServer (python) stopped")


if __name__ == "__main__":
    main()
```

```
LaundryStatusServer (python) ready on port 5052
client #1 connected
client #1: ping -> ok
client #1: status -> ok
client #1: status -> refused
client #1: status -> refused
client #1: ? -> refused
client #1: advance -> refused
client #1: advance -> ok
client #1: status -> ok
client #1: bye -> ok
client #1 disconnected
LaundryStatusServer (python) stopped
```

`readline(MAX_LINE + 1)` di Python langsung menerima batas. Lognya sama; satu-satunya perbedaan yang terlihat antara kedua server adalah *field* `"server"` pada jawaban `ping`, yang ditambahkan agar klien dapat mengetahui implementasi mana yang menjawab.

15.7.4 Objek klien

Klien dapat menyusun string JSON sendiri, tetapi itu akan menebarkan protokol ke setiap program yang memakai layanan. Jawaban berorientasi objek adalah kelas yang *mewakili layanan itu* di dalam klien: `client.status("ORD-0713", "7890")` tampak seperti panggilan biasa, dan kelas itu mengubahnya menjadi baris permintaan serta mengubah baris tanggapan menjadi peta. Inilah pola **Proxy**, objek lokal yang mewakili objek jarak jauh [12]. **Kode 15.19** adalah *proxy* dalam Java, dengan `main` yang menjalankan skenario tetap.

Kode 15.19 `LaundryStatusClient.java`: objek klien yang menyembunyikan format kabel, dan skenario yang menguji setiap aturan **Tabel 15.4**.

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.net.ConnectException;
import java.net.InetSocketAddress;
import java.net.Socket;
import java.nio.charset.StandardCharsets;
import java.util.LinkedHashMap;
import java.util.Map;

/** Objek klien yang menyembunyikan format kabel: pemanggil melihat metode, bukan baris
JSON. */
public class LaundryStatusClient implements AutoCloseable {
    private final Socket socket;
    private final BufferedReader in;
    private final PrintWriter out;

    public LaundryStatusClient(String host, int port) throws IOException {
        socket = new Socket();
        socket.connect(new InetSocketAddress(host, port), 2000);  // menyerah setelah 2 s
        socket.setSoTimeout(5000);  // dan jangan pernah menunggu jawaban lebih dari 5 s
        in = new BufferedReader(new InputStreamReader(
                socket.getInputStream(), StandardCharsets.UTF_8));
        out = new PrintWriter(new OutputStreamWriter(
                socket.getOutputStream(), StandardCharsets.UTF_8), true);
    }

    public Map<String, Object> ping() throws IOException {
        return send(Json.object("cmd", "ping"));
    }

    public Map<String, Object> status(String order, String phoneLast4) throws IOException {
        return send(Json.object("cmd", "status", "order", order, "phone_last4",
phoneLast4));
    }

    public Map<String, Object> advance(String order, String token) throws IOException {
        return send(Json.object("cmd", "advance", "order", order, "token", token));
    }

    public Map<String, Object> shutdownServer(String token) throws IOException {
        return send(Json.object("cmd", "shutdown", "token", token));
    }
```

Kode 15.19 (Lanjutan) *LaundryStatusClient.java: objek klien yang menyembunyikan format kabel, dan skenario yang menguji setiap aturan Tabel 15.4.*

```
    public Map<String, Object> send(Map<String, Object> request) throws IOException {
        return sendRaw(Json.write(request));
    }

    /** Mengirim satu baris persis apa adanya; berguna untuk menguji cara server menghadapi
sampah. */
    @SuppressWarnings("unchecked")
    public Map<String, Object> sendRaw(String line) throws IOException {
        out.println(line);
        String reply = in.readLine();
        if (reply == null) {
            throw new IOException("the server closed the connection");
        }
        return (Map<String, Object>) Json.parse(reply);
    }

    @Override
    public void close() throws IOException {
        try {
            if (!socket.isClosed()) {
                send(Json.object("cmd", "bye"));
            }
        } catch (IOException ignored) {
            // server mungkin sudah tiada; toh kita memang akan pergi
        } finally {
            socket.close();
        }
    }

    /** Menampilkan pesan sebagai JSON, dengan token diganti ***: rahasia tidak pernah
masuk log. */
    static String show(Map<String, Object> message) {
        Map<String, Object> copy = new LinkedHashMap<>(message);
        if (copy.containsKey("token")) {
            copy.put("token", "***");
        }
        return Json.write(copy);
    }

    private static void exchange(LaundryStatusClient client, Map<String, Object> request)
            throws IOException {
        System.out.println("> " + show(request));
        System.out.println("< " + show(client.send(request)));
    }
```

Kode 15.19 (Lanjutan) *LaundryStatusClient.java: objek klien yang menyembunyikan format kabel, dan skenario yang menguji setiap aturan Tabel 15.4.*

```
    /** Percakapan yang sama, server mana pun yang ada di ujung sana. */
    static void runScenario(LaundryStatusClient client, String token) throws IOException {
        exchange(client, Json.object("cmd", "ping"));
        exchange(client, Json.object("cmd", "status", "order", "ORD-0713", "phone_last4",
"7890"));
        exchange(client, Json.object("cmd", "status", "order", "ORD-0713", "phone_last4",
"0000"));
        exchange(client, Json.object("cmd", "status", "order", "ORD-9999", "phone_last4",
"7890"));
        System.out.println("> this is not JSON");
        System.out.println("< " + show(client.sendRaw("this is not JSON")));
        exchange(client, Json.object("cmd", "advance", "order", "ORD-0713", "token",
"guess"));
        exchange(client, Json.object("cmd", "advance", "order", "ORD-0713", "token",
                token == null ? "" : token));
        exchange(client, Json.object("cmd", "status", "order", "ORD-0713", "phone_last4",
"7890"));
    }

    public static void main(String[] args) {
        String host = args.length > 0 ? args[0] : "localhost";
        int port = args.length > 1 ? Integer.parseInt(args[1]) : 5052;
        String token = System.getenv("PROFITINA_STAFF_TOKEN");
        try (LaundryStatusClient client = new LaundryStatusClient(host, port)) {
            runScenario(client, token);
        } catch (ConnectException e) {
            System.out.println("Cannot connect to " + host + " port " + port
                    + ": is the server running?");
        } catch (IOException e) {
            System.out.println("Connection problem: " + e.getMessage());
        }
    }
}
```

Konstruktornya menyambung dengan batas dua detik (`new Socket(host, port)` dapat menunggu lama untuk mesin yang tak terjangkau) dan membatasi setiap pembacaan lima detik. `send` menulis satu baris JSON dan membaca satu baris, yang merupakan seluruh protokol dari sisi klien; `sendRaw` memungkinkan uji mengirim sampah secara sengaja; `close()` mengucapkan `bye` dan mengabaikan kegagalan, karena klien yang hendak pergi tidak semestinya tumbang hanya karena server lebih dahulu pergi. `show` mencetak pesan dengan setiap `token` diganti `***`: rahasia tidak pernah pantas masuk log, layar, atau tangkapan layar.

Untuk menjalankan pasangan ini, isi dahulu token staf di kedua terminal. Token itu rahasia: pilih string acak panjang milik Anda sendiri, dan jangan pernah meng-*commit*-nya:

```
Linux, macOS:        export PROFITINA_STAFF_TOKEN=your-own-long-random-string
Windows PowerShell:  $env:PROFITINA_STAFF_TOKEN = "your-own-long-random-string"
Windows cmd:         set PROFITINA_STAFF_TOKEN=your-own-long-random-string

Terminal 1:  java LaundryStatusServer 5052
Terminal 2:  java LaundryStatusClient localhost 5052
```

Klien mencetak setiap permintaan setelah `>` dan setiap tanggapan setelah `<`:

```
> {"cmd":"ping"}
< {"ok":true,"reply":"pong","server":"java"}
> {"cmd":"status","order":"ORD-0713","phone_last4":"7890"}
< {"ok":true,"order":"ORD-0713","customer":"B***
S******","status":"WASHING","ready_by":"2026-09-24 16:00"}
> {"cmd":"status","order":"ORD-0713","phone_last4":"0000"}
< {"ok":false,"error":"no order with that number and phone"}
> {"cmd":"status","order":"ORD-9999","phone_last4":"7890"}
< {"ok":false,"error":"no order with that number and phone"}
> this is not JSON
< {"ok":false,"error":"malformed request"}
> {"cmd":"advance","order":"ORD-0713","token":"***"}
< {"ok":false,"error":"not authorized"}
> {"cmd":"advance","order":"ORD-0713","token":"***"}
< {"ok":true,"order":"ORD-0713","status":"DRYING"}
> {"cmd":"status","order":"ORD-0713","phone_last4":"7890"}
< {"ok":true,"order":"ORD-0713","customer":"B***
S******","status":"DRYING","ready_by":"2026-09-24 16:00"}
```

Bacalah skenario ini terhadap **Tabel 15.4**: digit telepon yang benar menampilkan nama tersamar; digit yang salah dan pesanan yang tak dikenal mendapat penolakan yang persis sama; baris yang bukan JSON mendapat `malformed request` dan koneksinya tetap dapat dipakai; token tebakan ditolak dan token yang benar memajukan pesanan dari `WASHING` ke `DRYING`.

Kode 15.20 adalah *proxy* Python; `__enter__` dan `__exit__` membuatnya dapat dipakai dalam pernyataan `with`.

Kode 15.20 `laundry_status_client.py`: objek klien dalam Python.

```
"""laundry_status_client.py: objek klien yang menyembunyikan format kabel."""
import json
import os
import socket
import sys


def write(value):
    return json.dumps(value, separators=(",", ":"), ensure_ascii=False)


class LaundryStatusClient:
    def __init__(self, host, port):
        self._sock = socket.create_connection((host, port), timeout=2)  # menyerah setelah
2 s
        self._sock.settimeout(5)  # dan jangan pernah menunggu jawaban lebih dari 5 s
        self._reader = self._sock.makefile("r", encoding="utf-8", newline="\n")
        self._writer = self._sock.makefile("w", encoding="utf-8", newline="\n")

    def ping(self):
        return self.send({"cmd": "ping"})

    def status(self, order, phone_last4):
        return self.send({"cmd": "status", "order": order, "phone_last4": phone_last4})

    def advance(self, order, token):
        return self.send({"cmd": "advance", "order": order, "token": token})

    def shutdown_server(self, token):
        return self.send({"cmd": "shutdown", "token": token})

    def send(self, request):
        return self.send_raw(write(request))

    def send_raw(self, line):
        """Kirim satu baris persis apa adanya; berguna untuk menguji cara server menghadapi
sampah."""
        self._writer.write(line + "\n")
        self._writer.flush()
        reply = self._reader.readline()
        if reply == "":
            raise ConnectionError("the server closed the connection")
        return json.loads(reply)
```

Kode 15.20 (Lanjutan) *laundry_status_client.py: objek klien dalam Python.*

```
    def close(self):
        try:
            self.send({"cmd": "bye"})
        except OSError:
            pass  # server mungkin sudah tiada; toh kita memang akan pergi
        finally:
            self._reader.close()
            self._writer.close()
            self._sock.close()

    def __enter__(self):
        return self

    def __exit__(self, *exc):
        self.close()


def show(message):
    """Tampilkan pesan sebagai JSON, dengan token diganti ***: rahasia tidak pernah masuk
log."""
    copy = dict(message)
    if "token" in copy:
        copy["token"] = "***"
    return write(copy)


def _exchange(client, request):
    print("> " + show(request))
    print("< " + show(client.send(request)))


def run_scenario(client, token):
    """Percakapan yang sama, server mana pun yang ada di ujung sana."""
    _exchange(client, {"cmd": "ping"})
    _exchange(client, {"cmd": "status", "order": "ORD-0713", "phone_last4": "7890"})
    _exchange(client, {"cmd": "status", "order": "ORD-0713", "phone_last4": "0000"})
    _exchange(client, {"cmd": "status", "order": "ORD-9999", "phone_last4": "7890"})
    print("> this is not JSON")
    print("< " + show(client.send_raw("this is not JSON")))
    _exchange(client, {"cmd": "advance", "order": "ORD-0713", "token": "guess"})
    _exchange(client, {"cmd": "advance", "order": "ORD-0713", "token": token or ""})
    _exchange(client, {"cmd": "status", "order": "ORD-0713", "phone_last4": "7890"})
```

Kode 15.20 (Lanjutan) *laundry_status_client.py: objek klien dalam Python.*

```
def main():
    host = sys.argv[1] if len(sys.argv) > 1 else "localhost"
    port = int(sys.argv[2]) if len(sys.argv) > 2 else 5052
    token = os.environ.get("PROFITINA_STAFF_TOKEN")
    try:
        with LaundryStatusClient(host, port) as client:
            run_scenario(client, token)
    except ConnectionRefusedError:
        print(f"Cannot connect to {host} port {port}: is the server running?")
    except OSError as e:
        print(f"Connection problem: {e}")


if __name__ == "__main__":
    main()
```

Terhadap server Python (`python3 laundry_status_server.py 5052`), ia mencetak:

```
> {"cmd":"ping"}
< {"ok":true,"reply":"pong","server":"python"}
> {"cmd":"status","order":"ORD-0713","phone_last4":"7890"}
< {"ok":true,"order":"ORD-0713","customer":"B***
S******","status":"WASHING","ready_by":"2026-09-24 16:00"}
> {"cmd":"status","order":"ORD-0713","phone_last4":"0000"}
< {"ok":false,"error":"no order with that number and phone"}
> {"cmd":"status","order":"ORD-9999","phone_last4":"7890"}
< {"ok":false,"error":"no order with that number and phone"}
> this is not JSON
< {"ok":false,"error":"malformed request"}
> {"cmd":"advance","order":"ORD-0713","token":"***"}
< {"ok":false,"error":"not authorized"}
> {"cmd":"advance","order":"ORD-0713","token":"***"}
< {"ok":true,"order":"ORD-0713","status":"DRYING"}
> {"cmd":"status","order":"ORD-0713","phone_last4":"7890"}
< {"ok":true,"order":"ORD-0713","customer":"B***
S******","status":"DRYING","ready_by":"2026-09-24 16:00"}
```

15.7.5 Java bercakap dengan Python, dan Python bercakap dengan Java

Kita telah menyatakan bahwa protokol, bukan bahasa, yang menentukan siapa dapat berbicara dengan siapa. **Kode 15.21** membuktikannya: sebuah program Java menjalankan server **Python** sebagai proses terpisah pada port efemeral, menjalankan skenario klien Java terhadapnya, menghentikannya dengan `shutdown`, dan memeriksa bahwa ia benar-benar keluar.

Kode 15.21 `InteropDemo.java`: jalankan server Python sebagai proses terpisah dan bercakap dengannya lewat klien Java.

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.nio.charset.StandardCharsets;
import java.security.SecureRandom;
import java.util.HexFormat;
import java.util.concurrent.TimeUnit;

/** Menjalankan server PYTHON sebagai proses terpisah dan berbicara dengannya lewat klien
JAVA. */
public class InteropDemo {
    public static void main(String[] args) throws Exception {
        String script = args.length > 0 ? args[0] : "../python/laundry_status_server.py";
        boolean windows =
System.getProperty("os.name").toLowerCase().startsWith("windows");
        String python = System.getenv().getOrDefault("PYTHON", windows ? "python" :
"python3");

        // Token acak baru khusus untuk run ini; diserahkan lewat lingkungan (environment).
        byte[] random = new byte[16];
        new SecureRandom().nextBytes(random);
        String token = HexFormat.of().formatHex(random);

        ProcessBuilder builder = new ProcessBuilder(python, script, "0", "--quiet");
        builder.environment().put("PROFITINA_STAFF_TOKEN", token);
        builder.redirectErrorStream(true);
        Process server = builder.start();

        BufferedReader serverOut = new BufferedReader(new InputStreamReader(
                server.getInputStream(), StandardCharsets.UTF_8));
        int port = -1;
        String line;
        while ((line = serverOut.readLine()) != null) {
            if (line.startsWith("LaundryStatusServer")) {  // "... ready on port 43127"
                port = Integer.parseInt(line.substring(line.lastIndexOf(' ') + 1));
                break;
            }
        }
        if (port < 0) {
            System.out.println("the Python server did not start");
            return;
        }
        Thread drain = new Thread(() -> serverOut.lines().forEach(s -> { }));  // jaga pipa
tetap kosong
        drain.setDaemon(true);
        drain.start();
        System.out.println("started the Python server as a separate process");
```

Kode 15.21 (Lanjutan) *InteropDemo.java: jalankan server Python sebagai proses terpisah dan bercakap dengannya lewat klien Java.*

```
        try (LaundryStatusClient client = new LaundryStatusClient("127.0.0.1", port)) {
            System.out.println("answered by: " + client.ping().get("server"));
            LaundryStatusClient.runScenario(client, token);
            System.out.println("> " + LaundryStatusClient.show(Json.object("cmd",
"shutdown",
                    "token", token)));
            System.out.println("< " +
LaundryStatusClient.show(client.shutdownServer(token)));
        }
        boolean exited = server.waitFor(10, TimeUnit.SECONDS);
        System.out.println("Python server exited: " + exited + ", exit code "
                + (exited ? server.exitValue() : -1));
    }
}
```

Jalankan dari folder `java` dengan `java InteropDemo` (program ini mengharapkan berkas Python di `../python` dan menjalankan `python3`, atau `python` di Windows, kecuali variabel `PYTHON` menunjuk interpreter lain):

```
started the Python server as a separate process
answered by: python
> {"cmd":"ping"}
< {"ok":true,"reply":"pong","server":"python"}
> {"cmd":"status","order":"ORD-0713","phone_last4":"7890"}
< {"ok":true,"order":"ORD-0713","customer":"B***
S******","status":"WASHING","ready_by":"2026-09-24 16:00"}
> {"cmd":"status","order":"ORD-0713","phone_last4":"0000"}
< {"ok":false,"error":"no order with that number and phone"}
> {"cmd":"status","order":"ORD-9999","phone_last4":"7890"}
< {"ok":false,"error":"no order with that number and phone"}
> this is not JSON
< {"ok":false,"error":"malformed request"}
> {"cmd":"advance","order":"ORD-0713","token":"***"}
< {"ok":false,"error":"not authorized"}
> {"cmd":"advance","order":"ORD-0713","token":"***"}
< {"ok":true,"order":"ORD-0713","status":"DRYING"}
> {"cmd":"status","order":"ORD-0713","phone_last4":"7890"}
< {"ok":true,"order":"ORD-0713","customer":"B***
S******","status":"DRYING","ready_by":"2026-09-24 16:00"}
> {"cmd":"shutdown","token":"***"}
< {"ok":true,"reply":"shutting down"}
Python server exited: true, exit code 0
```

Empat tekniknya layak diketahui tersendiri.

- **Rahasia khusus untuk satu run.** `SecureRandom` menghasilkan 16 bita yang tak dapat ditebak, dan token sampai ke server melalui lingkungan proses anak, tidak pernah melalui baris perintah, tempat pengguna lain dapat melihatnya dalam daftar proses.
- **Menjalankan program lain.** `ProcessBuilder(python, script, "0", "--quiet")` menggambarkan perintahnya dan `start()` menjalankannya; `0` meminta port efemeral, `--quiet` membisukan log per klien.
- **Mengetahui port.** Demo membaca keluaran proses anak sampai baris `LaundryStatusServer (python) ready on port N` lalu mengambil kata terakhirnya. Sebuah *daemon thread* kemudian terus membaca dan membuang keluaran itu; kalau tidak, pipa akan penuh dan server terblokir.
- **Berhenti dengan benar.** `shutdown` dengan token, lalu `waitFor` dengan batas sepuluh detik; kode keluar 0 berarti berakhir normal.

Kode 15.22 bekerja ke arah sebaliknya: Python menjalankan server **Java** (kompilasi dahulu dengan `javac *.java`) dan menjalankan skenario klien Python.

Kode 15.22 `interop_demo.py`: jalankan server Java sebagai proses terpisah dan bercakap dengannya dari Python.

```
"""interop_demo.py: jalankan server JAVA sebagai proses terpisah, bercakap dengannya dari
PYTHON.

Kompilasi berkas Java lebih dahulu: cd ../java && javac *.java
"""
import os
import secrets
import shutil
import subprocess
import sys
import threading

from laundry_status_client import LaundryStatusClient, run_scenario, show


def main():
    class_dir = sys.argv[1] if len(sys.argv) > 1 else os.path.join("..", "java")
    java = shutil.which("java")
    if java is None:
        print("java was not found on the PATH")
        return

    token = secrets.token_hex(16)  # token acak baru khusus untuk run ini
    env = dict(os.environ, PROFITINA_STAFF_TOKEN=token)
    server = subprocess.Popen(
        [java, "-cp", class_dir, "LaundryStatusServer", "0", "--quiet"],
        stdout=subprocess.PIPE, stderr=subprocess.STDOUT, env=env,
        text=True, encoding="utf-8")

    port = None
    for line in server.stdout:
        if line.startswith("LaundryStatusServer"):  # "... ready on port 43127"
            port = int(line.split()[-1])
            break
    if port is None:
        print("the Java server did not start")
        return
    # Jaga pipa tetap kosong agar server tidak pernah terblokir oleh penyangga penuh.
    threading.Thread(target=server.stdout.read, daemon=True).start()
    print("started the Java server as a separate process")
```

Kode 15.22 (Lanjutan) *interop_demo.py: jalankan server Java sebagai proses terpisah dan bercakap dengannya dari Python.*

```
    with LaundryStatusClient("127.0.0.1", port) as client:
        print("answered by: " + client.ping()["server"])
        run_scenario(client, token)
        print("> " + show({"cmd": "shutdown", "token": token}))
        print("< " + show(client.shutdown_server(token)))
    try:
        code = server.wait(timeout=10)
        print(f"Java server exited: true, exit code {code}")
    except subprocess.TimeoutExpired:
        server.kill()
        print("Java server exited: false, exit code -1")


if __name__ == "__main__":
    main()
```

```
started the Java server as a separate process
answered by: java
> {"cmd":"ping"}
< {"ok":true,"reply":"pong","server":"java"}
> {"cmd":"status","order":"ORD-0713","phone_last4":"7890"}
< {"ok":true,"order":"ORD-0713","customer":"B***
S******","status":"WASHING","ready_by":"2026-09-24 16:00"}
> {"cmd":"status","order":"ORD-0713","phone_last4":"0000"}
< {"ok":false,"error":"no order with that number and phone"}
> {"cmd":"status","order":"ORD-9999","phone_last4":"7890"}
< {"ok":false,"error":"no order with that number and phone"}
> this is not JSON
< {"ok":false,"error":"malformed request"}
> {"cmd":"advance","order":"ORD-0713","token":"***"}
< {"ok":false,"error":"not authorized"}
> {"cmd":"advance","order":"ORD-0713","token":"***"}
< {"ok":true,"order":"ORD-0713","status":"DRYING"}
> {"cmd":"status","order":"ORD-0713","phone_last4":"7890"}
< {"ok":true,"order":"ORD-0713","customer":"B***
S******","status":"DRYING","ready_by":"2026-09-24 16:00"}
> {"cmd":"shutdown","token":"***"}
< {"ok":true,"reply":"shutting down"}
Java server exited: true, exit code 0
```

Selain kata “Python” dan “Java” serta *field* `server` pada `ping`, kedua keluaran identik: dua implementasi mandiri dari **Tabel 15.4** memberikan jawaban yang sama untuk setiap permintaan, termasuk penolakannya. Itulah hasil dari sebuah spesifikasi: pemilik klien Java tidak pernah membaca kode server Python, dan memang tidak perlu.

15.7.6 Biaya sebuah percakapan

Setiap permintaan dalam protokol ini adalah satu **perjalanan pulang-pergi** (*round trip*): permintaan berjalan ke server, server bekerja, tanggapan berjalan kembali. Waktu untuk satu pesan berukuran S bita melalui tautan dengan tunda satu arah (latensi) L dan lebar pita B adalah

$$T = L + \frac{S}{B} \tag{15.4}$$

dan percakapan yang terdiri atas k pasangan permintaan–tanggapan, masing-masing memerlukan waktu server t_s, memakan sekitar

$$T(k) \approx k \cdot (\mathrm{RTT} + t_s), \qquad \mathrm{RTT} = 2L + \frac{S_{\mathrm{req}} + S_{\mathrm{resp}}}{B}. \tag{15.5}$$

Dalam angka: permintaan status sekitar 60 bita dan tanggapannya sekitar 120. Antara binatu dan pusat data yang berjarak 20 ms melalui tautan 10 Mbit/s, S/B untuk 180 bita adalah $180 \times 8 / 10^7 \approx 0.14$ ms, dapat diabaikan dibanding $2L = 40$ ms. Menurut Persamaan (15.5), sepuluh permintaan terpisah memakan sekitar 400 ms dan satu permintaan untuk sepuluh pesanan sekitar 40 ms. Melalui jaringan sungguhan, banyaknya perjalanan pulang-pergi jauh lebih menentukan daripada banyaknya bita, jadi utamakan sedikit pesan yang kaya daripada banyak pesan kecil; Soal Latihan 15.6 melatih hitungannya.

15.8 HTTP: Protokol Web

Protokol status pesanan kita bersifat privat. Protokol yang paling banyak dipakai di dunia adalah **HTTP** (*Hypertext Transfer Protocol*), bahasa setiap peramban dan sebagian besar layanan yang dipanggil program melalui Internet [13]. HTTP dibangun seperti protokol kita, baris-baris teks di

atas TCP, tetapi terstandardisasi, sehingga peramban, aplikasi ponsel, dan program dalam bahasa apa pun dapat memakai server yang sama.

15.8.1 Seperti apa percakapan HTTP

Permintaan HTTP terdiri atas beberapa baris teks: baris pertama berisi **metode** (apa yang dilakukan), **path** (sumber daya mana), dan versi; baris-baris **header** `Name: value`; satu baris kosong; dan, untuk sebagian metode, sebuah badan (*body*). Tanggapan memiliki bentuk yang sama dengan **baris status** di awal. Berikut permintaan untuk pesanan ORD-0714 beserta jawabannya sebagaimana keduanya melewati socket (setiap baris diakhiri `\r\n`):

```
GET /orders/ORD-0714 HTTP/1.1
Host: 127.0.0.1
Connection: close

HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
Content-Length: 47

{"ok":true,"order":"ORD-0714","status":"READY"}
```

Metode yang umum adalah `GET` (membaca), `POST` (membuat atau mengirim), `PUT` (mengganti), dan `DELETE`. Digit pertama kode status menunjukkan golongan jawabannya; **Tabel 15.5** mendaftar kode-kode yang pertama kali ditemui pemula.

Tabel 15.5 Kode status HTTP yang umum. Kode 2xx berarti berhasil, 4xx masalah pada permintaan, 5xx masalah di server.

Kode	Makna	Contoh di binatu
200 OK	Permintaan berhasil; badan berisi jawabannya	status pesanan yang ada
201 Created	Sumber daya baru telah dibuat	pesanan baru didaftarkan
400 Bad Request	Permintaan cacat	badan bukan JSON yang sah
401 Unauthorized	Kredensial tidak ada atau salah	perintah staf tanpa token yang sah
403 Forbidden	Kredensial sah tetapi tidak mencukupi	kurir mencoba mengubah harga
404 Not Found	Sumber daya tidak ada	pesanan ORD-9999
429 Too Many Requests	Klien harus memperlambat	terlalu banyak tebakan digit telepon
500 Internal Server Error	Server gagal	eksepsi yang tidak tertangkap di penangan
503 Service Unavailable	Server kelebihan beban atau sedang dirawat	selama pembaruan

HTTP menstandardisasi persis apa yang harus kita ciptakan sendiri: operasi mana yang dimaksud (metode dan path), berhasil atau gagal (kode status), dan metadata (header seperti `Content-Type`). Pembingkaiannya: header berakhir pada baris kosong, dan `Content-Length` menyatakan berapa bita badan yang menyusul.

15.8.2 Server dan klien HTTP di setiap bahasa

Pustaka standar kedua bahasa menyertakan server HTTP kecil dan klien HTTP. **Kode 15.23** menjalankan server di dalam program pada loopback dan port efemeral, sehingga tidak memerlukan Internet; mengajukan tiga pertanyaan kepadanya dengan `java.net.http.HttpClient`; mengajukan satu pertanyaan lagi dengan mengetik HTTP secara manual ke socket polos; lalu menghentikan server.

Kode 15.23 `HttpStatusDemo.java`: server HTTP di localhost, ditanya dengan `HttpClient` dan dengan socket polos.

```
import com.sun.net.httpserver.HttpExchange;
import com.sun.net.httpserver.HttpServer;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStream;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.net.InetAddress;
import java.net.InetSocketAddress;
import java.net.Socket;
import java.net.URI;
import java.net.http.HttpClient;
import java.net.http.HttpRequest;
import java.net.http.HttpResponse;
import java.nio.charset.StandardCharsets;
import java.time.Duration;
import java.util.Map;

/** Server HTTP mungil di localhost, ditanya dengan HttpClient dan dengan socket polos. */
public class HttpStatusDemo {
    private static final Map<String, String> STATUS = Map.of(
            "ORD-0713", "WASHING", "ORD-0714", "READY", "ORD-0715", "RECEIVED");

    public static void main(String[] args) throws Exception {
        HttpServer server = HttpServer.create(
                new InetSocketAddress(InetAddress.getLoopbackAddress(), 0), 0);
        server.createContext("/health", exchange -> reply(exchange, 200,
                "text/plain; charset=utf-8", "ok"));
        server.createContext("/orders/", HttpStatusDemo::orderStatus);
        server.start();
        int port = server.getAddress().getPort();
        String base = "http://127.0.0.1:" + port;
        System.out.println("HTTP server started on an ephemeral port");

        HttpClient client = HttpClient.newBuilder()
                .proxy(HttpClient.Builder.NO_PROXY)  // bicara langsung, abaikan proxy
sistem
                .connectTimeout(Duration.ofSeconds(2))
                .build();
        for (String path : new String[] {"/health", "/orders/ORD-0714", "/orders/ORD-
9999"}) {
            HttpRequest request = HttpRequest.newBuilder(URI.create(base + path))
                    .timeout(Duration.ofSeconds(5))
                    .GET()
                    .build();
            HttpResponse<String> response = client.send(request,
                    HttpResponse.BodyHandlers.ofString(StandardCharsets.UTF_8));
```

Kode 15.23 (Lanjutan) *HttpStatusDemo.java: server HTTP di localhost, ditanya dengan HttpClient dan dengan socket polos.*

```
            System.out.println("GET " + path + " -> " + response.statusCode() + " "
                    + response.headers().firstValue("content-type").orElse("?"));
            System.out.println("    body: " + response.body());
        }

        // HTTP hanyalah teks di atas TCP: pertanyaan yang sama, diketik tangan pada
socket.
        try (Socket socket = new Socket(InetAddress.getLoopbackAddress(), port);
             PrintWriter out = new PrintWriter(new OutputStreamWriter(
                     socket.getOutputStream(), StandardCharsets.US_ASCII));
             BufferedReader in = new BufferedReader(new InputStreamReader(
                     socket.getInputStream(), StandardCharsets.UTF_8))) {
            out.print("GET /health HTTP/1.1\r\nHost: 127.0.0.1\r\nConnection:
close\r\n\r\n");
            out.flush();
            System.out.println("raw socket, first line of the answer: " + in.readLine());
        }

        server.stop(0);  // berhenti seketika; tidak ada permintaan yang sedang berjalan
        System.out.println("HTTP server stopped");
    }

    private static void orderStatus(HttpExchange exchange) throws IOException {
        String id = exchange.getRequestURI().getPath().substring("/orders/".length());
        String status = STATUS.get(id);
        if (status == null) {
            reply(exchange, 404, "application/json; charset=utf-8",
                    Json.write(Json.object("ok", false, "error", "no such order")));
        } else {
            reply(exchange, 200, "application/json; charset=utf-8",
                    Json.write(Json.object("ok", true, "order", id, "status", status)));
        }
    }

    private static void reply(HttpExchange exchange, int code, String type, String body)
            throws IOException {
        byte[] bytes = body.getBytes(StandardCharsets.UTF_8);
        exchange.getResponseHeaders().set("Content-Type", type);
        exchange.sendResponseHeaders(code, bytes.length);
        try (OutputStream os = exchange.getResponseBody()) {
            os.write(bytes);
        }
    }
}
```

```
HTTP server started on an ephemeral port
GET /health -> 200 text/plain; charset=utf-8
    body: ok
GET /orders/ORD-0714 -> 200 application/json; charset=utf-8
    body: {"ok":true,"order":"ORD-0714","status":"READY"}
GET /orders/ORD-9999 -> 404 application/json; charset=utf-8
    body: {"ok":false,"error":"no such order"}
raw socket, first line of the answer: HTTP/1.1 200 OK
HTTP server stopped
```

- `HttpServer.create` berasal dari `com.sun.net.httpserver`, bagian dari setiap JDK. `createContext(path, handler)` menghubungkan awalan path dengan penangan yang menerima sebuah `HttpExchange`, yaitu satu permintaan beserta tanggapannya.
- `reply` memasang header, mengirim kode status dan panjang badan dalam *bita* dengan `sendResponseHeaders`, lalu menulis badannya.

- `NO_PROXY` membuat klien menyambung langsung, bukan melalui *proxy* yang dikonfigurasi di komputer, yang tidak dapat menjangkau loopback. `client.send(request, BodyHandlers.ofString(UTF_8))` menunggu lalu mengembalikan `HttpResponse` dengan `statusCode()`, `headers()`, dan `body()`; kode 404 adalah tanggapan, bukan eksepsi.
- Blok terakhir membuktikan bahwa HTTP adalah teks di atas TCP: sebuah `Socket` biasa, tiga baris header dan satu baris kosong yang ditulis manual, lalu baris status `HTTP/1.1 200 OK` kembali. `server.stop(0)` menghentikan server seketika.

Kode 15.24 adalah versi Python, dengan `http.server` dan `urllib.request`.

Kode 15.24 `http_status_demo.py`: server dan pertanyaan yang sama dalam Python.

```
"""http_status_demo.py: server HTTP mungil di localhost, ditanya dengan urllib dan socket polos."""
import json
import socket
import threading
import urllib.error
import urllib.request
from http.server import BaseHTTPRequestHandler, ThreadingHTTPServer

STATUS = {"ORD-0713": "WASHING", "ORD-0714": "READY", "ORD-0715": "RECEIVED"}


def write(value):
    return json.dumps(value, separators=(",", ":"), ensure_ascii=False)


class StatusHandler(BaseHTTPRequestHandler):
    protocol_version = "HTTP/1.1"

    def do_GET(self):
        if self.path == "/health":
            self._reply(200, "text/plain; charset=utf-8", "ok")
        elif self.path.startswith("/orders/"):
            order_id = self.path[len("/orders/"):]
            status = STATUS.get(order_id)
            if status is None:
                self._reply(404, "application/json; charset=utf-8",
                            write({"ok": False, "error": "no such order"}))
            else:
                self._reply(200, "application/json; charset=utf-8",
                            write({"ok": True, "order": order_id, "status": status}))
        else:
            self._reply(404, "text/plain; charset=utf-8", "not found")

    def _reply(self, code, content_type, body):
        data = body.encode("utf-8")
        self.send_response(code)
        self.send_header("Content-Type", content_type)
        self.send_header("Content-Length", str(len(data)))
        self.end_headers()
        self.wfile.write(data)

    def log_message(self, format, *args):
        pass  # biarkan konsol tetap tenang
```

Kode 15.24 (Lanjutan) *http_status_demo.py: server dan pertanyaan yang sama dalam Python.*

```
def main():
    server = ThreadingHTTPServer(("127.0.0.1", 0), StatusHandler)
    port = server.server_address[1]
    threading.Thread(target=server.serve_forever, daemon=True).start()
    base = f"http://127.0.0.1:{port}"
    print("HTTP server started on an ephemeral port")

    # bicara langsung, abaikan proxy sistem
    opener = urllib.request.build_opener(urllib.request.ProxyHandler({}))
    for path in ["/health", "/orders/ORD-0714", "/orders/ORD-9999"]:
        try:
            response = opener.open(base + path, timeout=5)
        except urllib.error.HTTPError as e:  # 4xx dan 5xx datang sebagai eksepsi
            response = e
        with response:
            body = response.read().decode("utf-8")
            print(f"GET {path} -> {response.status} {response.headers['Content-Type']}")
            print(f"    body: {body}")

    # HTTP hanyalah teks di atas TCP: pertanyaan yang sama, diketik tangan pada socket.
    with socket.create_connection(("127.0.0.1", port), timeout=5) as sock:
        sock.sendall(b"GET /health HTTP/1.1\r\nHost: 127.0.0.1\r\nConnection:
close\r\n\r\n")
        with sock.makefile("r", encoding="utf-8", newline="\r\n") as reader:
            first = reader.readline().rstrip("\r\n")
    print("raw socket, first line of the answer: " + first)

    server.shutdown()      # hentikan serve_forever()
    server.server_close()  # lepaskan port
    print("HTTP server stopped")


if __name__ == "__main__":
    main()
```

```
HTTP server started on an ephemeral port
GET /health -> 200 text/plain; charset=utf-8
    body: ok
GET /orders/ORD-0714 -> 200 application/json; charset=utf-8
    body: {"ok":true,"order":"ORD-0714","status":"READY"}
GET /orders/ORD-9999 -> 404 application/json; charset=utf-8
    body: {"ok":false,"error":"no such order"}
raw socket, first line of the answer: HTTP/1.1 200 OK
HTTP server stopped
```

Penangan Python mewarisi `BaseHTTPRequestHandler` dan menimpa (*override*) `do_GET`: pola *template method* dari Bab 7, di mana pustaka mengurai permintaan lalu memanggil metode kita. `ThreadingHTTPServer` melayani setiap permintaan di thread-nya sendiri sampai `shutdown()`. `urllib` melaporkan 4xx dan 5xx dengan melempar `HTTPError`, yang juga merupakan tanggapan, sehingga program mencetaknya seperti jawaban lain; `ProxyHandler({})` adalah padanan `NO_PROXY` di Python.

Layanan untuk aplikasi ponsel akan berbicara HTTP melalui *framework* web (*back end* Profitina memakai FastAPI, Subbab 15.12). Protokol socket mentah tetap layak diketahui: dari situlah HTTP dibuat, gim dan aplikasi obrolan memakainya ketika pola satu-permintaan-satu-tanggapan terlalu kaku, dan protokol mentah memperlihatkan apa yang dikerjakan sebuah *framework* untuk Anda.

15.9 Keamanan dan Data Pribadi

Program yang mendengarkan di jaringan menerima masukan dari orang asing, dan sebagian dari mereka mungkin sengaja mencoba merusaknya. Selain itu, layanan yang menangani nama dan nomor telepon pelanggan tunduk pada Undang-Undang Nomor 27 Tahun 2022 tentang Pelindungan Data Pribadi (UU PDP) [14], yang berlaku penuh sejak Oktober 2024. Subbab ini bukan nasihat hukum; ia memperlihatkan bagaimana aturan-aturan itu menjadi keputusan pemrograman sehari-hari, yang sebagian besar sudah diambil layanan kita.

15.9.1 Tujuh kebiasaan untuk kode jaringan

1. **Dengarkan hanya di tempat yang perlu.** Server kita mengikat ke `127.0.0.1`. Agar laptop kasir dapat menjangkau server di ruang belakang, ikat ke alamat privat server itu atau ke semua alamat (`0.0.0.0`), dan biarkan *firewall* menentukan siapa yang boleh menyambung. Bukalah layanan ke Internet hanya di balik perlindungan yang layak.
2. **Jangan percaya apa pun dari pihak lain.** Periksa keberadaan dan tipe setiap *field*; jawab permintaan cacat dengan penolakan, jangan pernah dengan kerusakan.
3. **Batasi segalanya**: panjang baris (1024 karakter), lama berdiam diri (60 detik), banyaknya pekerja (16). Tanpa batas, satu klien jahat dapat menghabiskan memori atau thread bagi semua orang, sebuah *denial of service* (penolakan layanan).
4. **Jauhkan rahasia dari kode dan log.** Variabel lingkungan, perbandingan waktu-konstan, dicetak sebagai `***`. Rahasia yang pernah di-*commit* ke Git menjadi publik selamanya dan harus diganti.
5. **Enkripsi apa pun yang meninggalkan mesin.** TCP polos dapat dibaca siapa saja di Wi-Fi yang sama. Obatnya adalah **TLS**, yang mengubah HTTP menjadi HTTPS [15]: `SSLSocket`/`SSLServerSocket` di Java (otomatis untuk `https://` di `HttpClient`), modul `ssl` di Python. Di lingkungan produksi, *reverse proxy* atau terowongan (*tunnel*) biasanya menyediakannya di depan aplikasi.
6. **Katakan sedikit saat menolak.** Pesan terpisah untuk “pesanan tidak ada” dan “telepon salah” memungkinkan penebak menemukan pesanan yang ada lebih dahulu.
7. **Perlambat para penebak.** Empat digit telepon memberi 10^4 kemungkinan; menebak dalam urutan acak rata-rata memerlukan sekitar setengahnya, sehingga dengan r percobaan per detik, waktu harapannya adalah

$$t_{\text{guess}} \approx \frac{10^4 + 1}{2 \cdot r}. \tag{15.6}$$

Pada $r = 1000$, Persamaan (15.6) memberi sekitar lima detik; karena itulah layanan sungguhan menambahkan **pembatasan laju** (*rate limiting*). Mengizinkan lima percobaan per menit untuk satu pesanan, $r = 1/12$ per detik, memberi $5000.5 \times 12 \approx 60\,000$ detik, sekitar 17 jam, cukup lama bagi staf untuk menyadarinya. Soal Latihan 15.10 merancangnya.

15.9.2 Apa yang diminta UU PDP dari program seperti ini

UU PDP membedakan **subjek data pribadi** (pelanggan), **pengendali data pribadi** (Profitina Laundry, yang menentukan tujuan dan cara pemrosesan), dan **prosesor data pribadi** (misalnya perusahaan *hosting* yang bekerja atas namanya). Data pribadi adalah apa saja yang mengidentifikasi seseorang, sendiri atau digabungkan: nama, nomor telepon, bahkan riwayat pesanan. **Tabel 15.6** memetakan kewajiban utamanya ke layanan kita.

Tabel 15.6 Kewajiban utama UU PDP No. 27/2022, dan bagaimana masing-masing tampak dalam sebuah layanan jaringan kecil.

Kewajiban (dengan kata sederhana)	Artinya bagi layanan status pesanan
Memproses data hanya atas dasar yang sah, misalnya persetujuan atau pelaksanaan perjanjian	Mengumpulkan nama dan nomor telepon karena pelanggan memesan layanan; jelaskan alasannya di meja kasir
Mengumpulkan hanya yang diperlukan tujuan (minimalisasi data)	Layanan status memerlukan nomor pesanan dan empat digit terakhir telepon; alamat tidak diperlukan
Menjaga data tetap akurat, dan memungkinkan subjek melihat, membetulkan, dan menghapusnya	Alat staf harus mendukung pembetulan dan penghapusan catatan pelanggan
Melindungi data dengan langkah teknis yang memadai	Ikat ke loopback atau alamat privat, TLS di jaringan, rahasia di lingkungan, nama tersamar pada tanggapan
Tidak mengungkapkan lebih dari yang perlu	Tampilkan inisial saja; jangan pernah mengembalikan nomor telepon; penolakan yang sama untuk semua kegagalan
Tidak menyimpan data lebih lama dari yang perlu	Hapus atau anonimkan pesanan beberapa bulan setelah diambil
Melaporkan kegagalan pelindungan dengan segera	UU PDP mewajibkan pemberitahuan tertulis kepada subjek data dan lembaga pengawas paling lambat 3 × 24 jam setelah terjadi kegagalan pelindungan data pribadi
Menyimpan catatan pemrosesan	Log yang mencatat *apa* yang terjadi (sebuah permintaan status berhasil) tanpa menyalin data *siapa*

Dua baris tabel ini sudah terlihat dalam kode kita. Log mencatat `status -> ok`, bukan nomor pesanan atau nama, karena log disalin dan dibaca banyak orang. Dan `publicView` mengembalikan inisial dan tidak pernah nomor telepon, karena tujuannya adalah memberi tahu apakah cucian sudah siap, bukan memberi tahu siapa pun yang memegang struk siapa pelanggannya. Pelindungan data adalah serangkaian keputusan kecil yang diambil ketika merancang protokol, satu alasan lagi untuk merancangnya di atas kertas terlebih dahulu.

> **PERINGATAN:** Jangan pernah memakai data pelanggan sungguhan dalam proyek tim, uji, tangkapan layar, atau demonstrasi. Karanglah nama dan nomor, seperti yang dilakukan bab ini. Data uji yang berupa data pribadi sungguhan tunduk pada undang-undang yang sama dengan data produksi.

15.10 Menguji Program Jaringan

Kode jaringan memerlukan uji otomatis dari Bab 6 lebih daripada kode lain, karena *bug*-nya hanya muncul pada urutan waktu dan kegagalan tertentu. Empat teknik membuatnya terkendali.

- **Uji protokol tanpa jaringan**: `processInput` dan `handle` tidak melakukan I/O.
- **Jalankan server sungguhan di dalam uji, pada port 0**, sehingga ia tidak pernah bertabrakan dengan uji atau server lain; `start` mengembalikan portnya dan `stop` melepaskannya.
- **Pakai server baru untuk setiap uji**, sehingga tidak ada uji yang melihat data yang diubah uji lain.
- **Uji kegagalan secara sengaja**: sampah, banjiran data, banyak klien sekaligus, server yang sudah dihentikan. Jalur bahagia berhasil lebih dahulu; *bug* sungguhan tinggal di tempat lain.

Kode 15.25 menerapkan keempatnya. Mengikuti Bab 6, program ini adalah pemeriksa mandiri dengan `main`, yang hanya memerlukan JDK; di JUnit 5, `@BeforeEach` dan `@AfterEach` akan menjalankan dan menghentikan server.

Kode 15.25 `LaundryStatusCheck.java`: sepuluh pemeriksaan otomatis, masing-masing terhadap servernya sendiri pada port efemeral.

```
import java.io.IOException;
import java.net.ConnectException;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.Future;

/**
 * Pemeriksaan otomatis untuk layanan status pesanan. Setiap pemeriksaan menjalankan
servernya sendiri
 * pada port efemeral, berbicara dengannya lewat socket sungguhan, lalu menghentikannya
lagi.
 */
public class LaundryStatusCheck {
    private static final String TOKEN = "test-token-not-a-secret";
    private static int passed = 0;
    private static int failed = 0;

    interface Check {
        void run(int port) throws Exception;
    }

    public static void main(String[] args) {
        check("ping answers pong", port -> {
            try (LaundryStatusClient c = new LaundryStatusClient("127.0.0.1", port)) {
                expect("pong", c.ping().get("reply"));
            }
        });
        check("status shows initials only, never the phone", port -> {
            try (LaundryStatusClient c = new LaundryStatusClient("127.0.0.1", port)) {
                Map<String, Object> r = c.status("ORD-0714", "5432");
                expect("S*** W********", r.get("customer"));
                expect("READY", r.get("status"));
                expect(false, r.containsKey("phone"));
            }
        });
        check("wrong phone and unknown order look the same", port -> {
            try (LaundryStatusClient c = new LaundryStatusClient("127.0.0.1", port)) {
                expect(c.status("ORD-0714", "0000"), c.status("ORD-9999", "5432"));
            }
        });
        check("junk is refused and the connection survives", port -> {
            try (LaundryStatusClient c = new LaundryStatusClient("127.0.0.1", port)) {
                expect("malformed request", c.sendRaw("{not json").get("error"));
                expect("malformed request", c.sendRaw("[1, 2, 3]").get("error"));
                expect("unknown command: fly", c.send(Json.object("cmd",
"fly")).get("error"));
```

Kode 15.25 (Lanjutan) *LaundryStatusCheck.java: sepuluh pemeriksaan otomatis, masing-masing terhadap servernya sendiri pada port efemeral.*

```
                expect(true, c.ping().get("ok"));
            }
        });
        check("advance needs the staff token", port -> {
            try (LaundryStatusClient c = new LaundryStatusClient("127.0.0.1", port)) {
                expect("not authorized", c.advance("ORD-0715", "wrong").get("error"));
                expect("RECEIVED", c.status("ORD-0715", "2222").get("status"));
            }
        });
        check("advance walks the status forward and stops at PICKED_UP", port -> {
            try (LaundryStatusClient c = new LaundryStatusClient("127.0.0.1", port)) {
                expect("PICKED_UP", c.advance("ORD-0714", TOKEN).get("status"));
                expect("order already picked up", c.advance("ORD-0714",
TOKEN).get("error"));
            }
        });
        check("a request longer than 1024 characters is refused", port -> {
            try (LaundryStatusClient c = new LaundryStatusClient("127.0.0.1", port)) {
                expect("request too long", c.sendRaw("x".repeat(5000)).get("error"));
            }
        });
        check("twenty clients at once are all answered", port -> {
            ExecutorService pool = Executors.newFixedThreadPool(20);
            List<Future<Object>> answers = new ArrayList<>();
            for (int i = 0; i < 20; i++) {
                answers.add(pool.submit(() -> {
                    try (LaundryStatusClient c = new LaundryStatusClient("127.0.0.1",
port)) {
                        return c.status("ORD-0713", "7890").get("status");
                    }
                }));
            }
            for (Future<Object> answer : answers) {
                expect("WASHING", answer.get());
            }
            pool.shutdown();
        });
        check("five staff clients advancing at once lose no update", port -> {
            ExecutorService pool = Executors.newFixedThreadPool(5);
            List<Future<Object>> answers = new ArrayList<>();
            for (int i = 0; i < 5; i++) {  // RECEIVED + 5 langkah = PICKED_UP, tepat
                answers.add(pool.submit(() -> {
                    try (LaundryStatusClient c = new LaundryStatusClient("127.0.0.1",
port)) {
                        return c.advance("ORD-0715", TOKEN).get("ok");
                    }
                }));
            }
            for (Future<Object> answer : answers) {
```

Kode 15.25 (Lanjutan) *LaundryStatusCheck.java: sepuluh pemeriksaan otomatis, masing-masing terhadap servernya sendiri pada port efemeral.*

```
                expect(true, answer.get());
            }
            pool.shutdown();
            try (LaundryStatusClient c = new LaundryStatusClient("127.0.0.1", port)) {
                expect("PICKED_UP", c.status("ORD-0715", "2222").get("status"));
            }
        });

        // Server mati: begitu dihentikan, klien harus mendapat galat yang jelas, bukan
macet.
        LaundryStatusServer server = LaundryStatusServer.withSampleOrders(TOKEN, false);
        try {
            int port = server.start(0);
            server.stop();
            try {
                new LaundryStatusClient("127.0.0.1", port).close();
                record("a stopped server gives ConnectException", false);
            } catch (ConnectException e) {
                record("a stopped server gives ConnectException", true);
            }
        } catch (IOException e) {
            record("a stopped server gives ConnectException", false);
        }

        System.out.println((passed + failed) + " checks, " + passed + " passed, "
                + failed + " failed");
    }

    private static void check(String name, Check body) {
        LaundryStatusServer server = LaundryStatusServer.withSampleOrders(TOKEN, false);
        try {
            int port = server.start(0);  // server baru dengan data baru untuk setiap
pemeriksaan
            body.run(port);
            record(name, true);
        } catch (Throwable t) {
            record(name + " (" + t.getMessage() + ")", false);
        } finally {
            server.stop();
        }
    }

    private static void expect(Object expected, Object actual) {
        if (!expected.equals(actual)) {
            throw new AssertionError("expected " + expected + " but got " + actual);
        }
    }
```

Kode 15.25 (Lanjutan) *LaundryStatusCheck.java: sepuluh pemeriksaan otomatis, masing-masing terhadap servernya sendiri pada port efemeral.*

```
    private static void record(String name, boolean ok) {
        if (ok) {
            passed++;
        } else {
            failed++;
        }
        System.out.println((ok ? "PASS  " : "FAIL  ") + name);
    }
}
```

```
PASS  ping answers pong
PASS  status shows initials only, never the phone
PASS  wrong phone and unknown order look the same
PASS  junk is refused and the connection survives
PASS  advance needs the staff token
PASS  advance walks the status forward and stops at PICKED_UP
PASS  a request longer than 1024 characters is refused
PASS  twenty clients at once are all answered
PASS  five staff clients advancing at once lose no update
PASS  a stopped server gives ConnectException
10 checks, 10 passed, 0 failed
```

Seluruh run hanya sepersekian detik. “Twenty clients at once” memeriksa bahwa kumpulan thread dan perulangan penerima di bawah beban menjawab semua klien. “Five staff clients advancing at once” menguji pelajaran Bab 14: dari `RECEIVED`, tepat lima langkah mencapai `PICKED_UP`, sehingga kelimanya harus berhasil dan pesanan harus berakhir di `PICKED_UP`. Tanpa `synchronized` pada `OrderBook.advance`, dua thread dapat membaca status lama yang sama dan menulis status baru yang sama; satu langkah lenyap dan pesanan berakhir di `READY`. (Uji seperti ini kadang-kadang menangkap *race*; bukti ketiadaannya adalah kata kunci `synchronized` dan penalaran Bab 14.)

Pemeriksaan terakhir adalah kasus “server mati” dalam proyek tim, yang diotomatiskan: setelah `stop`, klien harus segera mendapat `ConnectException`, bukan macet.

Kode 15.26 adalah rangkaian uji yang sama dengan `unittest`, yang `setUp` dan `tearDown`-nya menjalankan dan menghentikan server baru di sekitar setiap uji.

Kode 15.26 `test_laundry_status.py`: sepuluh uji yang sama dengan `unittest`.

```
"""test_laundry_status.py: uji otomatis untuk layanan status pesanan.

Setiap uji menjalankan servernya sendiri pada port efemeral, berbicara dengannya
lewat socket sungguhan, lalu menghentikannya lagi. Jalankan: python3 test_laundry_status.py
"""
import unittest
from concurrent.futures import ThreadPoolExecutor

from laundry_status_client import LaundryStatusClient
from laundry_status_server import LaundryStatusServer, OrderBook

TOKEN = "test-token-not-a-secret"


class LaundryStatusTest(unittest.TestCase):
    def setUp(self):
        # server baru dengan data baru untuk setiap uji
        self.server = LaundryStatusServer(OrderBook.sample(), TOKEN, verbose=False)
        self.port = self.server.start(0)

    def tearDown(self):
        self.server.stop()

    def client(self):
        return LaundryStatusClient("127.0.0.1", self.port)

    def test_ping_answers_pong(self):
        with self.client() as c:
            self.assertEqual("pong", c.ping()["reply"])

    def test_status_shows_initials_only(self):
        with self.client() as c:
            r = c.status("ORD-0714", "5432")
            self.assertEqual("S*** W********", r["customer"])
            self.assertEqual("READY", r["status"])
            self.assertNotIn("phone", r)

    def test_wrong_phone_and_unknown_order_look_the_same(self):
        with self.client() as c:
            self.assertEqual(c.status("ORD-0714", "0000"), c.status("ORD-9999", "5432"))
```

Kode 15.26 (Lanjutan) *test_laundry_status.py: sepuluh uji yang sama dengan unittest.*

```
    def test_junk_is_refused_and_connection_survives(self):
        with self.client() as c:
            self.assertEqual("malformed request", c.send_raw("{not json")["error"])
            self.assertEqual("malformed request", c.send_raw("[1, 2, 3]")["error"])
            self.assertEqual("unknown command: fly", c.send({"cmd": "fly"})["error"])
            self.assertTrue(c.ping()["ok"])

    def test_advance_needs_the_staff_token(self):
        with self.client() as c:
            self.assertEqual("not authorized", c.advance("ORD-0715", "wrong")["error"])
            self.assertEqual("RECEIVED", c.status("ORD-0715", "2222")["status"])

    def test_advance_stops_at_picked_up(self):
        with self.client() as c:
            self.assertEqual("PICKED_UP", c.advance("ORD-0714", TOKEN)["status"])
            self.assertEqual("order already picked up", c.advance("ORD-0714",
TOKEN)["error"])

    def test_long_request_is_refused(self):
        with self.client() as c:
            self.assertEqual("request too long", c.send_raw("x" * 5000)["error"])

    def test_twenty_clients_at_once(self):
        def ask(_):
            with self.client() as c:
                return c.status("ORD-0713", "7890")["status"]
        with ThreadPoolExecutor(max_workers=20) as pool:
            self.assertEqual(["WASHING"] * 20, list(pool.map(ask, range(20))))

    def test_five_staff_clients_lose_no_update(self):
        def advance(_):
            with self.client() as c:
                return c.advance("ORD-0715", TOKEN)["ok"]
        with ThreadPoolExecutor(max_workers=5) as pool:  # RECEIVED + 5 langkah = PICKED_UP
            self.assertEqual([True] * 5, list(pool.map(advance, range(5))))
        with self.client() as c:
            self.assertEqual("PICKED_UP", c.status("ORD-0715", "2222")["status"])

    def test_stopped_server_refuses_connections(self):
        self.server.stop()
        with self.assertRaises(ConnectionRefusedError):
            self.client()


if __name__ == "__main__":
    unittest.main(verbosity=2)
```

```
test_advance_needs_the_staff_token
(__main__.LaundryStatusTest.test_advance_needs_the_staff_token) ... ok
test_advance_stops_at_picked_up
(__main__.LaundryStatusTest.test_advance_stops_at_picked_up) ... ok
test_five_staff_clients_lose_no_update
(__main__.LaundryStatusTest.test_five_staff_clients_lose_no_update) ... ok
test_junk_is_refused_and_connection_survives
(__main__.LaundryStatusTest.test_junk_is_refused_and_connection_survives) ... ok
test_long_request_is_refused (__main__.LaundryStatusTest.test_long_request_is_refused) ...
ok
test_ping_answers_pong (__main__.LaundryStatusTest.test_ping_answers_pong) ... ok
test_status_shows_initials_only
(__main__.LaundryStatusTest.test_status_shows_initials_only) ... ok
test_stopped_server_refuses_connections
(__main__.LaundryStatusTest.test_stopped_server_refuses_connections) ... ok
test_twenty_clients_at_once (__main__.LaundryStatusTest.test_twenty_clients_at_once) ... ok
test_wrong_phone_and_unknown_order_look_the_same
(__main__.LaundryStatusTest.test_wrong_phone_and_unknown_order_look_the_same) ... ok

----------------------------------------------------------------------
Ran 10 tests in 2.049s

OK
```

Rangkaian uji Python memakan sekitar dua detik karena setiap `stop()` menunggu batas waktu 0,2 detik berikutnya dari perulangan penerima, harga yang pantas untuk server yang selalu dapat dihentikan dengan bersih.

15.11 Proyek Tim

Segala hal dalam buku ini sejauh ini dapat dikerjakan satu orang. Aplikasi klien–server adalah titik di mana hal itu berhenti, dan di mana rancangan berorientasi objek memberi dividen terbesarnya: kelas dengan antarmuka yang jelas memungkinkan beberapa orang membangun satu program tanpa saling menginjak kaki. Subbab ini adalah panduan praktis bagi tim beranggotakan dua atau tiga orang yang membangun aplikasi GUI dan socket, dari pertemuan pertama sampai presentasi; Subbab 15.11.10 menyajikan contoh ringkasan proyek yang lengkap.

15.11.1 Memilih topik

Proyek tim yang baik memiliki satu sifat yang sama: topiknya *memerlukan* klien dan server, karena beberapa orang memakainya pada waktu yang sama dari tempat yang berbeda dan saling melihat tindakan satu sama lain. Contohnya:

- **gim dengan obrolan**, di mana server menyimpan keadaan dan aturan permainan, dan setiap klien menampilkan papan serta mengirim langkah;
- **jejaring sosial** kecil (“apa yang terjadi di binatu hari ini”), di mana klien mengirim dan membaca kiriman, dan server menyimpan serta menyebarkannya;
- **alat kolaborasi**, seperti papan tugas bersama untuk organisasi mahasiswa atau daftar belanja bersama untuk rumah kos;
- **portal interaktif**, seperti tampilan antrean binatu, papan barang hilang-dan-temuan di kampus, atau papan menu-dan-pesanan untuk warung makan;
- **gagasan *startup*** kecil milik Anda sendiri, atau apa saja yang benar-benar menarik bagi tim Anda.

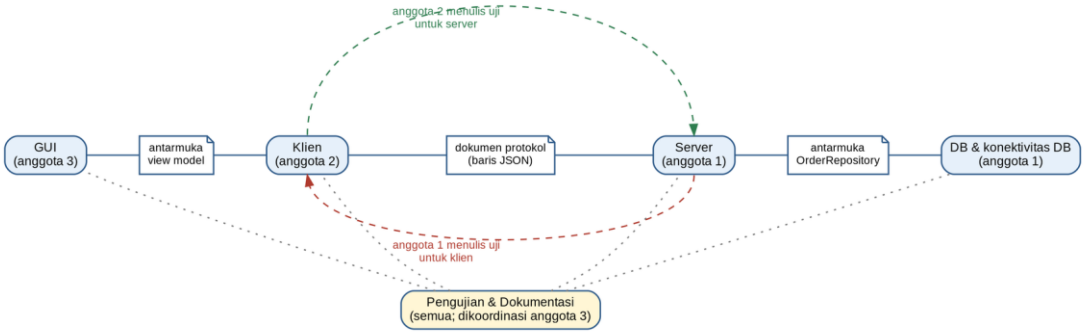
Tetapkan topik dan judul kerja pada pertemuan pertama, dengan satu paragraf *pitch*: siapa yang memakainya, untuk apa, dan mengapa memerlukan jaringan. Lalu terapkan tiga uji. *Jaringan*: bila

gagasan itu sama baiknya sebagai satu program tunggal, itu belum topik klien–server. *Ukuran*: dapatkah intinya didemonstrasikan setelah dua pertiga waktu? Bila tidak, pangkaslah sekarang. *Kegagalan*: dapatkah Anda menyebut sesuatu yang salah di jaringan (klien terputus di tengah permainan, server dijalankan ulang ketika ada kiriman) yang penting bagi pengguna? Bila ya, Anda punya sesuatu yang nyata untuk dirancang dan dipertontonkan.

15.11.2 Membagi pekerjaan

Aplikasi klien–server dengan antarmuka grafis secara alami terbagi menjadi lima komponen, seperti tampak pada **Gambar 15.8**:

- **server**, yang menerima koneksi, menjalankan protokol, dan memiliki keadaan bersama;
- **klien**, yang menyambung, mengirim permintaan, dan menafsirkan tanggapan (*proxy* di Subbab 15.7.4);
- **basis data dan konektivitas basis data**, yang menyimpan keadaan secara permanen, sehingga server yang dijalankan ulang tetap mengingat pesanannya;
- **GUI**, jendela Swing atau Tkinter dari Bab 10 dan 11, yang sungguh dilihat pengguna;
- **pengujian dan dokumentasi**, yang merupakan komponen tersendiri, bukan urusan belakangan.



Gambar 15.8 Lima komponen proyek tim, pemiliknya, dan tiga dokumen antarmuka di antaranya. Panah putus-putus menunjukkan pengujian silang: setiap pemilik menulis uji untuk komponen yang memakai komponennya, atau yang dipakai oleh komponennya.

Setiap anggota memiliki satu komponen atau lebih dan menulis **spesifikasi dan uraian tugas** singkat: apa yang dikerjakan komponen, apa yang ditawarkan, apa yang dibutuhkan, dan apa yang akan diserahkan kapan. Kepemilikan berarti "orang ini bertanggung jawab atasnya dan meninjau setiap perubahannya", bukan "tidak boleh ada orang lain yang menyentuhnya".

Batas antarkomponen lebih penting daripada komponennya. **Gambar 15.8** memiliki tiga batas, masing-masing dituliskan sebagai **dokumen antarmuka** sebelum ada kode:

- antara klien dan server, **dokumen protokol**: tabel seperti **Tabel 15.4** ditambah diagram keadaan seperti **Gambar 15.5**;
- antara server dan basis data, **antarmuka repositori**: misalnya `interface OrderRepository` di Java dengan `find(id)`, `save(order)`, `all()`, atau kelas abstrak padanannya di Python;
- antara klien dan GUI, **view model**: metode yang dipanggil jendela (`refresh()`, `advance(orderId)`) dan peristiwa yang diterimanya (`statusChanged`).

"Antarmuka" di sini bermakna umum: tidak harus `interface` Java, melainkan kesepakatan tertulis apa pun tentang apa yang ditawarkan satu bagian kepada bagian lain, entah tabel, berkas Markdown, atau tanda tangan metode dengan Javadoc atau *docstring*. Kesepakatan itu harus

tertulis, dibaca kedua pihak, dan hanya diubah atas persetujuan bersama. MVC (Bab 11) adalah gagasan yang sama di dalam satu program; di sini ia berlaku antarmanusia.

15.11.3 Dari spesifikasi ke rancangan ke implementasi

Bawalah proyek melalui tiga fase yang terlihat, dan simpan hasil setiap fase di repositori.

1. **Spesifikasi**: *apa* yang dilakukan sistem bagi pengguna, berupa fitur satu kalimat ("pelanggan dapat melihat status pesanan dengan memberikan nomornya dan empat digit terakhir telepon"), masing-masing dengan kriteria kapan dianggap berfungsi. Dokumen protokol termasuk di sini.
2. **Rancangan**: *bagaimana* bagian-bagiannya berpadu: diagram komponen, dokumen antarmuka, diagram kelas UML per komponen (Bab 7), dan diagram keadaan protokol.
3. **Implementasi**: kodenya, ditulis mengikuti rancangan, beserta ujinya.

Fase-fase ini bertumpang-tindih, karena implementasi mengajarkan banyak hal, tetapi setiap fase harus *ada* sebelum fase berikutnya dimulai; tim yang menulis kode sejak hari pertama menghasilkan tiga program yang hampir cocok. Urutan kerja yang memanfaatkan tiga orang sejak minggu pertama:

- Minggu 1: ketiganya bersama-sama menulis spesifikasi dan dokumen antarmuka, lalu menyiapkan repositori.
- Minggu 2: pemilik server menulis server dan mengujinya dengan klien berskenario; pemilik klien menulis *proxy* dan mengujinya terhadap **server tiruan** (*stub server*), tiruan mungil yang memberi jawaban tetap dan benar; pemilik GUI membangun jendela di atas klien tiruan. Kerja sejajar hanya mungkin karena antarmukanya sudah tetap.
- Minggu 3: ganti tiruan dengan bagian sungguhan, satu batas demi satu batas, dan jalankan uji silang di subbab berikutnya.
- Minggu 4: penanganan kegagalan, dokumentasi, gladi.

15.11.4 Kasus uji melintasi antarmuka

Pikirkan kasus uji sejak awal dan *melintasi* setiap batas. Bila A memakai fungsi-fungsi B, pemilik A paling tahu apa yang dibutuhkan A dari B, sehingga pemilik A merancang kasus uji untuk B, dan pemilik B merancang kasus uji untuk A. Pada **Gambar 15.8**, pemilik klien menguji server dan pemilik server menguji klien.

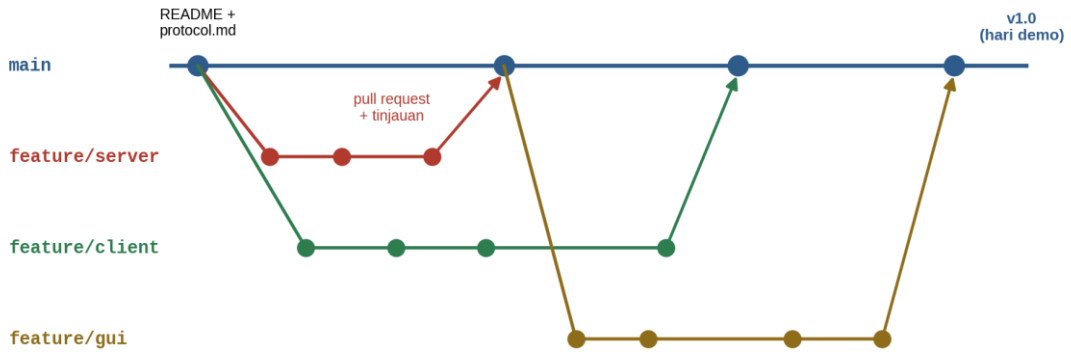
Alasannya manusiawi: orang yang menulis sebuah komponen menguji apa yang ia maksudkan; orang yang memakainya menguji apa yang ia andaikan. *Bug* hidup di celah antara keduanya, dan pengujian silang menemukannya sebelum hari demonstrasi. Uji silang disimpan di repositori, seperti **Kode 15.25**, dan dijalankan pada setiap perubahan. Rencana uji yang berguna memuat:

- **uji unit** untuk setiap kelas, seperti di Bab 6, termasuk objek protokol tanpa jaringan;
- **uji berbasis tiruan**, di mana klien tiruan menguji server sungguhan, atau server tiruan menguji klien sungguhan;
- **uji multithread**, di mana beberapa klien memakai server sekaligus, seperti **Kode 15.13** dan dua pemeriksaan konkurensi pada **Kode 15.25**;
- **uji kegagalan**: masukan cacat, server yang mati, klien yang lenyap;
- **uji terdistribusi**, dengan server di satu laptop dan klien di laptop lain melalui jaringan bersama, setidaknya sekali sebelum presentasi; loopback menyembunyikan *firewall*, alamat ikat yang salah, dan Wi-Fi yang lambat.

15.11.5 Bekerja bersama dengan Git

Saling mengirim berkas lewat aplikasi obrolan membuat pekerjaan hilang. **Git** mencatat setiap perubahan, siapa pembuatnya dan mengapa, serta memungkinkan beberapa orang bekerja pada salinan terpisah lalu menggabungkannya dengan aman [16]; repositori bersama di GitHub atau GitLab menyimpan versi yang benar dari proyek.

Alur kerja yang cocok bagi tim kecil adalah *feature branch* (cabang fitur, **Gambar 15.9**). `main` selalu dapat dikompilasi dan lulus ujinya. Setiap potong pekerjaan mendapat cabangnya sendiri; bila sudah siap, pemiliknya membuka **pull request**, seorang rekan meninjaunya (membaca, menjalankan, menyetujui atau meminta perubahan), lalu cabang itu digabungkan.



Gambar 15.9 Cabang fitur. Tim memulai `main` dengan README dan dokumen protokol; setiap anggota bekerja di cabangnya; setiap cabang kembali ke `main` melalui *pull request* yang ditinjau; versi yang didemonstrasikan diberi *tag*.

Tabel 15.7 mendaftar perintah-perintah minggu pertama; tampilan *Source Control* di VS Code juga menyediakannya.

Tabel 15.7 Perintah Git yang diperlukan anggota tim pada minggu pertama.

Perintah	Kegunaannya
`git clone <url>`	menyalin repositori bersama ke komputer Anda (sekali saja)
`git switch -c feature/server`	membuat cabang baru dan pindah ke sana
`git status`	menampilkan berkas yang berubah dan yang sudah di-*stage*
`git add <file>`	menyiapkan (*stage*) perubahan untuk *commit* berikutnya
`git commit -m "Add status command"`	merekam perubahan yang di-*stage* beserta pesannya
`git push -u origin feature/server`	mengirim cabang Anda ke repositori bersama
`git switch main` lalu `git pull`	memperbarui salinan `main` Anda
`git merge main` (di cabang Anda)	membawa `main` terbaru ke cabang Anda
`git log --oneline --graph`	menampilkan riwayat sebagai gambar seperti **Gambar 15.9**

Kebiasaan yang mencegah sebagian besar masalah:

- **Commit kecil dan sering**, setiap *commit* satu perubahan logis dengan pesan yang menjelaskan alasannya (“Refuse lines longer than 1024 characters”).

- **Tarik (*pull*) sebelum mulai dan sering gabungkan `main` ke cabang Anda**, agar konflik tetap kecil.
- **Selesaikan konflik bersama.** Git menandai kedua versi di antara `<<<<<<<`, `=======`, dan `>>>>>>>`; sepakati mana yang benar, sunting, `git add`, `git commit` (Soal Latihan 15.13).
- **Jauhkan rahasia dan sampah** dengan `.gitignore` (`*.class`, `__pycache__/`, folder editor, berkas berisi token).
- **Jaga `main` tetap hijau.** Jangan pernah menggabungkan cabang yang tidak dapat dikompilasi atau yang ujinya gagal.

15.11.6 README

Setiap repositori diawali `README.md`, hal pertama yang dilihat siapa pun. Isinya menjawab, berurutan: apa ini, bagaimana menjalankannya, bagaimana memakainya, di mana yang lain. Berikut templat untuk proyek klien–server (ditulis dalam bahasa Inggris, sebagaimana lazimnya repositori yang dibagikan):

```
# Laundry Live Board

A queue board for Profitina Laundry: the counter registers orders, staff move them
through washing, drying and ironing, and every screen in the shop shows the queue live.

## Team
- Ayu (server, database)    - Bagus (client, protocol)    - Citra (GUI, testing lead)

## Requirements
Java 21 or later (Temurin); Python 3.10 or later for the test scripts.

## Build and run
    cd server && javac *.java && java BoardServer 5060
    cd client && javac *.java && java BoardClient localhost 5060
Set PROFITINA_STAFF_TOKEN in the environment first (see docs/installation.md).

## Using it
Screenshots and a two-minute walk-through: docs/user-guide.md

## Documentation
docs/protocol.md  docs/design.md  docs/test-plan.md  docs/troubleshooting.md

## Tests
    cd server && java BoardServerCheck
```

15.11.7 Kegagalan: server mati, klien mati

Aplikasi yang hanya berjalan ketika kedua pihak sehat dan jaringannya sempurna belumlah selesai. Dua kegagalan harus ditangani dengan sengaja dan didemonstrasikan:

- **Ketika server mati, klien harus tahu.** Tidak macet, tidak memuntahkan *stack trace*: pesan yang jelas (“the board server is not reachable; retrying”), masukan pengguna yang belum terkirim tetap disimpan, percobaan ulang dengan penantian yang makin panjang (Soal Latihan 15.7), dan pesan yang jelas ketika menyerah. Setiap panggilan yang terblokir memerlukan batas waktu.
- **Ketika klien mati, server harus tahu.** Klien yang terputus atau *crash* tidak boleh menumbangkan server, membocorkan thread atau socket-nya, atau mengganggu klien lain; akhir aliran, eksepsi, dan batas waktu diam semuanya berujung di blok `finally` yang membereskan, dan dalam obrolan atau gim, klien lain diberi tahu.

Keduanya bergantung pada **penanganan eksepsi** (Bab 5) untuk galat koneksi di kedua sisi. Tulislah penangannya lebih dahulu, dan uji kedua kegagalan dengan menghentikan server di tengah sesi dan mematikan klien di tengah permintaan.

15.11.8 Dokumentasi yang sungguh dipakai

Di samping kode, tulislah empat dokumen singkat untuk empat pembaca:

- **panduan pengguna**, bagi orang yang memakai aplikasi: apa yang dikerjakannya, dengan tangkapan layar dan penelusuran singkat;
- **kriteria kinerja**, bagi tim dan peninjaunya: apa arti "berjalan dengan baik" dalam angka yang dapat diperiksa, misalnya "20 klien sekaligus, setiap jawaban status dalam 200 ms di jaringan kampus", dan bagaimana Anda mengukurnya;
- **panduan instalasi**, bagi orang yang memasangnya: langkah persis di Windows, macOS, dan Linux, port yang dipakai, variabel lingkungan yang diperlukan;
- **panduan pemecahan masalah**, bagi orang yang mendapati aplikasinya tidak berjalan: pesan galat yang dapat dicetak program, dan makna masing-masing ("Address already in use: another program uses port 5060; stop it or start the server with another port").

15.11.9 Mempresentasikan proyek

Presentasi adalah demonstrasi yang bercerita; rencanakan seperti sebuah protokol.

- **Buka dengan masalahnya**, dalam satu kalimat, dari sudut pandang pengguna: "Pelanggan menelepon meja kasir sepanjang hari untuk bertanya apakah cuciannya sudah siap."
- **Demonstrasikan secara langsung, di lebih dari satu komputer**, dengan klien yang saling melihat tindakan satu sama lain. Lalu hentikan server dengan sengaja dan biarkan hadirin melihat klien melaporkannya dan pulih kembali: tidak ada yang lebih meyakinkan daripada kegagalan yang ditangani dengan baik.
- **Tunjukkan rancangan secara singkat**: diagram komponen, tabel protokol, satu kelas yang menarik. Jelaskan satu keputusan yang Anda banggakan dan satu yang ingin Anda ubah.
- **Tunjukkan uji yang berjalan**, termasuk uji konkurensi.
- **Biarkan setiap anggota berbicara** tentang bagian miliknya; peninjau ingin mendengar setiap orang menjelaskan kodenya sendiri.
- **Bersiaplah menghadapi nasib buruk**: gladi dua kali di mesin yang sebenarnya, uji jaringan ruangan lebih awal, dan simpan rekaman layar dari run yang berhasil sebagai cadangan.
- **Tutup dengan jujur**: apa yang berjalan, apa yang belum, dan apa yang Anda pelajari.

15.11.10 Contoh ringkasan proyek yang lengkap

Ringkasan ini ditulis dengan cara yang semestinya ditempuh sebuah tim untuk proyeknya sendiri pada minggu pertama, cukup kecil untuk tiga orang dan empat minggu.

Judul. Laundry Live Board.

Gambaran singkat (*pitch*). Profitina Laundry memiliki satu meja kasir, empat mesin cuci, tiga mesin pengering, dan arus pelanggan yang ingin tahu kapan cuciannya siap. Live Board adalah aplikasi klien–server: staf di meja kasir mendaftarkan pesanan dan memajukannya melalui tahap-tahapnya; setiap layar di toko (laptop kasir, monitor yang menghadap ruang tunggu, tablet staf) menampilkan antrean dan diperbarui dalam satu detik setelah ada perubahan; pelanggan dapat

memeriksa status pesanan di layar swalayan dengan nomor pesanan dan empat digit terakhir teleponnya.

Pengguna dan fitur.

- F1. Staf kasir mendaftarkan pesanan: nama pelanggan, telepon, layanan, berat. Server memberikan nomor pesanan.
- F2. Staf memajukan pesanan ke tahap berikutnya; semua layar diperbarui dalam satu detik.
- F3. Monitor ruang tunggu menampilkan antrean: nomor pesanan, nama tersamar, tahap, perkiraan waktu siap. Nomor telepon tidak pernah tampil.
- F4. Pelanggan memeriksa satu pesanan di layar swalayan dengan nomor pesanan dan digit telepon; setelah lima kali gagal, layar itu menunggu satu menit.
- F5. Jalur obrolan staf untuk pesan singkat ("W2 is free").
- F6. Bila server dijalankan ulang, semua pesanan tetap ada; klien menyambung kembali dengan sendirinya.

Komponen dan pemilik. Ayu: server dan basis data (SQLite melalui JDBC, di balik antarmuka `OrderRepository`). Bagus: pustaka klien dan protokol, termasuk penyambungan ulang. Citra: GUI Swing untuk kasir dan monitor, koordinator pengujian, koordinator dokumentasi.

Protokol. Baris JSON di atas TCP seperti di Subbab 15.7, dengan `register`, `advance`, `status`, `queue`, `chat`, dan `bye`, ditambah satu hal: setelah `{"cmd":"subscribe"}`, server *mendorong* (*push*) baris `{"event":"changed",...}` setiap kali antrean berubah. Klien lalu memerlukan thread pendengar yang membaca peristiwa selagi GUI mengirim permintaan, seperti anggota obrolan pada Penyelesaian 15.11. Dokumen protokol memuat tabel seperti **Tabel 15.4** dan diagram keadaan sebuah sesi klien.

Tonggak. **Tabel 15.8** memberi satu hasil kerja per minggu, masing-masing diperiksa oleh orang lain selain pembuatnya.

Tabel 15.8 Rencana tonggak contoh proyek, satu hasil kerja per minggu, masing-masing diperiksa oleh orang lain selain pembuatnya.

Minggu	Hasil kerja	Diperiksa oleh
1	Spesifikasi, dokumen protokol, diagram komponen, dokumen antarmuka, repositori dengan README dan `.gitignore`	seluruh tim, bersama-sama
2	Server lulus uji unitnya dengan klien berskenario; pustaka klien lulus ujinya terhadap server tiruan; GUI berjalan di atas klien tiruan	pemilik komponen tetangga (uji silang)
3	Bagian sungguhan disatukan; F1–F4 berjalan di satu komputer; uji konkurensi dengan 20 klien lulus	Citra menjalankan seluruh rencana uji
4	F5–F6; penanganan kegagalan didemonstrasikan di tiga komputer; dokumen pengguna, instalasi, kinerja, dan pemecahan masalah; dua kali gladi	semua

Kriteria kinerja. 20 klien bersamaan; perubahan terlihat di setiap layar dalam satu detik melalui Wi-Fi toko; jawaban status dalam 200 ms; memori server di bawah 256 MB.

Data pribadi. Hanya data uji dengan nama karangan; nomor telepon tidak pernah ditampilkan atau dicatat; nama tersamar di layar publik; pesanan dihapus 90 hari setelah diambil; token staf di lingkungan.

Risiko. Pembaruan yang didorong dan thread GUI (mitigasi: aturan Swing di Bab 10, pembaruan melalui `SwingUtilities.invokeLater`); Wi-Fi saat presentasi (mitigasi: router cadangan dan sebuah rekaman).

15.11.11 Daftar periksa untuk tim

Telusuri daftar ini bersama-sama seminggu sebelum presentasi dan sekali lagi sehari sebelumnya. Ini bukan lembar skor; ini adalah pertanyaan yang akan diajukan peninjau yang cermat, atau pelanggan sungguhan.

- Dapatkah setiap orang di antara kita menyatakan topik, pengguna, dan fitur dalam dua kalimat?
- Adakah diagram komponen dengan pemilik untuk setiap komponen, dan spesifikasi tertulis untuk masing-masing?
- Apakah setiap batas memiliki dokumen antarmuka yang sudah dibaca kedua pihak, dan apakah kodenya sesuai?
- Apakah dokumen protokol menyatakan, untuk setiap keadaan, siapa yang berbicara, apa yang boleh dikatakan, dan apa yang terjadi setelah pesan tak terduga? Adakah diagram alir atau diagram keadaannya?
- Apakah pemilik setiap komponen menulis uji untuk komponen tetangganya? Apakah uji mencakup uji berbasis tiruan, multithread, dan kegagalan, dan sudahkah kita menjalankannya setidaknya sekali di beberapa komputer?
- Hentikan server di tengah sesi: apakah setiap klien mengatakannya dengan jelas, menyimpan pekerjaan pengguna, dan pulih ketika server kembali?
- Matikan klien di tengah permintaan: apakah server tetap melayani yang lain, dan apakah ia melepaskan thread dan socket klien itu?
- Apakah setiap panggilan jaringan yang terblokir memiliki batas waktu?
- Adakah README yang memungkinkan pendatang baru membangun dan menjalankan sistem dalam sepuluh menit, ditambah dokumen pengguna, instalasi, kinerja, dan pemecahan masalah?
- Apakah repositori bersih dari berkas hasil kompilasi, kata sandi, token, dan data pribadi sungguhan? Apakah `main` dapat dikompilasi dan lulus semua uji?
- Apakah riwayat memperlihatkan *commit* setiap anggota, dengan pesan yang menjelaskan alasannya?
- Sudahkah kita gladi demonstrasi dua kali di mesin yang sebenarnya, dan adakah rekaman cadangan?

15.12 Profitina dalam Praktik

Layanan status pesanan itu kecil, tetapi bentuknya adalah bentuk Profitina. Profitina adalah platform visibilitas AI yang mengukur seberapa siap situs web sebuah merek untuk dikutip oleh mesin jawab AI (*generative engine optimization*, GEO). Aplikasi webnya di app.profitina.com adalah server Python yang dibangun dengan FastAPI dan dilayani Uvicorn, dan setiap peramban, ponsel, dan tugas terjadwal yang memakainya adalah klien yang berbicara HTTP dan JSON (Subbab 15.8). Dasbornya membuat satu permintaan, `GET /api/me/dashboard`, dan menerima notifikasi, kanal yang diikuti, peringkat, dan aktivitas dalam satu jawaban alih-alih selusin permintaan: Persamaan (15.5) yang diterapkan pada sebuah produk.

Umpan komunitas waktu-nyata memakai **Server-Sent Events** (SSE): klien menjaga satu koneksi HTTP tetap terbuka dan server mendorong satu baris setiap kali ada kiriman baru, gagasan `subscribe` pada contoh ringkasan proyek, yang distandardisasi untuk peramban. Lingkungan produksinya berjalan di balik Cloudflare Tunnel yang menyediakan TLS (Subbab 15.9), dan sebuah rilis baru dianggap berhasil ketika versi baru menjawab `/api/health`, titik akhir

kesehatan seperti pada **Kode 15.23**. Fitur AI berjalan dengan model lokal (Qwen2.5 melalui Ollama), sehingga data pengguna tidak pernah meninggalkan server untuk pemrosesan AI.

Data pribadi ditangani berdasarkan UU PDP No. 27/2022 dengan **Tabel 15.6** yang diwujudkan menjadi fitur: *hashing* kata sandi dengan Argon2id, persetujuan eksplisit, pembatasan laju login (Persamaan (15.6)), ekspor data swalayan, dan penghapusan akun sepenuhnya. Tidak ada yang dirilis sebelum rangkaian uji otomatis yang berjumlah jauh lebih dari seribu uji itu lulus. Dan di seluruhnya, aplikasi mengukur dan memberi saran; penggunalah yang memutuskan.

15.13 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya

1. Lupa membilas. Baris tertahan di penyangga dan kedua program membeku. *Hindari:* `PrintWriter` dengan *autoflush* di Java; `flush()` setelah setiap pesan di Python.

2. Lupa baris baru. `print("Hello")` tidak pernah menuntaskan `readLine` di pihak lain. *Hindari:* `println`, atau `"\n"` di Python; "satu pesan, satu baris" di dokumen protokol.

3. Menganggap satu `recv` sama dengan satu pesan. TCP adalah aliran bita. *Hindari:* aturan pembingkaian dan pembaca yang menerapkannya (Subbab 15.4.4).

4. Melayani satu klien pada satu waktu. *Hindari:* serahkan setiap socket yang diterima kepada thread kumpulan (**Gambar 15.6**).

5. Berbagi keadaan tanpa perlindungan antar-thread klien, yang lambat laun kehilangan pembaruan. *Hindari:* objek per klien, dan `synchronized`, kunci, atau kelas atomik untuk sisanya (Bab 14).

6. Menunggu selamanya pada klien yang diam atau server yang tak terjangkau. *Hindari:* batas waktu baca pada setiap socket terhubung dan batas waktu sambung pada setiap klien.

7. "Address already in use". Server kedua pada port yang sama gagal, dan di Linux dan macOS server Python yang dijalankan ulang dapat gagal bahkan setelah server pertama berhenti:

```
EchoServer error: [Errno 98] Address already in use
EchoServer stopped
```

Hindari: hentikan server lama atau pilih port lain; pasang `SO_REUSEADDR` pada server Python di Linux dan macOS; pakai port 0 dalam uji.

8. Pengodean yang tidak cocok antara klien Windows dan server UTF-8. *Hindari:* sebutkan UTF-8 di kedua sisi setiap aliran.

9. Mengikat ke semua alamat selama pengembangan, yang memperlihatkan server setengah jadi. *Hindari:* `127.0.0.1` sampai Anda memerlukan lebih, lalu *firewall* dan TLS.

10. Rahasia di kode sumber atau log. *Hindari:* variabel lingkungan, perbandingan waktu-konstan, `***` pada keluaran.

11. Membocorkan data pribadi dalam pesan galat dan log, misalnya "nomor telepon salah untuk Budi Santoso". *Hindari:* penolakan yang seragam dan tidak informatif; log yang mencatat apa yang terjadi, bukan data milik siapa.

12. Menulis kode sebelum protokol disepakati. *Hindari:* tabel, diagram, dan objek protokol lebih dahulu, ditinjau bersama.

15.14 Ringkasan Bab

- Sebuah **host** memiliki **alamat IP** (IPv4, 32 bit; IPv6, 128 bit); `127.0.0.1` adalah **loopback**, "komputer ini". **Port** (16 bit, Persamaan (15.1)) memilih programnya; port 0 meminta sistem memberikan port **efemeral** yang bebas. Koneksi dikenali oleh **empat-serangkainya** (Persamaan (15.2)).
- Internet menumpuk empat **lapisan** (tautan, internet, transpor, aplikasi, **Gambar 15.1**). **TCP** memberi aliran bita yang andal dan berurutan melalui koneksi; **UDP** memberi datagram mandiri yang menjaga batasnya tetapi dapat hilang (**Tabel 15.1**).
- **Server** mengikat dan mendengarkan; **klien** menyambung. `accept()` mengembalikan **socket terhubung yang baru** untuk setiap klien sementara **socket pendengar** tetap terbuka (**Gambar 15.2**, **Gambar 15.3**, **Gambar 15.4**). Java memakai `ServerSocket` dan `Socket`; Python memakai `socket.socket` dengan `bind`, `listen`, `accept`, dan `create_connection` (**Tabel 15.2**).
- Bungkus aliran socket dengan *reader* dan *writer* **UTF-8**, **bilas** setelah setiap pesan, dan pakai aturan **pembingkaian**; baris yang dibatasi baris baru adalah yang paling sederhana. Setiap klien mengikuti lima langkah: buka socket, buka aliran, bercakap sesuai protokol, tutup aliran, tutup socket.
- **Rancang protokol di atas kertas terlebih dahulu**: tabel keadaan dan pesan, diagram keadaan, dan **objek protokol** tanpa I/O yang dapat diuji tersendiri (**Tabel 15.3**, **Gambar 15.5**).
- Layani banyak klien dengan **thread penerima dan kumpulan thread**; bagilah sesedikit mungkin dan lindungi yang dibagi (**Gambar 15.6**, Persamaan (15.3)). Buktikan konkurensi dengan uji yang akan gagal tanpanya.
- **Layanan status pesanan** berbicara dengan **baris JSON** (**Tabel 15.4**); objek **proxy** menyembunyikan format kabel; karena kedua implementasi mengikuti spesifikasi yang sama, klien Java bekerja dengan server Python dan klien Python dengan server Java. Perjalanan pulang-pergi mendominasi biaya (Persamaan (15.4), Persamaan (15.5)).
- **HTTP** adalah teks di atas TCP dengan metode, path, kode status, dan header (**Tabel 15.5**); `HttpServer`/`HttpClient` dan `http.server`/`urllib` berbicara dengannya.
- **Keamanan**: dengarkan secara sempit, validasi segalanya, batasi panjang, waktu, dan thread, simpan rahasia di lingkungan, enkripsi dengan TLS, tolak tanpa rincian, batasi laju tebakan (Persamaan (15.6)). **UU PDP No. 27/2022** meminta minimalisasi, pelindungan, retensi terbatas, dan pemberitahuan kegagalan dengan segera (**Tabel 15.6**).
- **Uji kode jaringan** dengan server di dalam proses pada port 0, server baru untuk setiap uji, dan kegagalan yang disengaja.
- **Proyek tim** berhasil melalui topik yang layak-jaringan, lima komponen dengan pemilik dan **antarmuka** tertulis, pengujian silang, **Git** dengan cabang fitur dan peninjauan, **README**, penanganan **server mati** dan **klien mati** yang disengaja, empat jenis dokumentasi, dan demonstrasi langsung yang sudah digladikan.

15.15 Soal Latihan

Soal Latihan 15.1 (empat-serangkai). Laptop kasir di `192.168.1.20` membuka dua koneksi ke server status pesanan di `192.168.1.10`, port 5052, dan tablet staf di `192.168.1.30` membuka satu koneksi. Sistem operasi memberikan port klien 51234 dan 51235 di laptop dan 49800 di tablet. (a) Tuliskan ketiga empat-serangkai menurut Persamaan (15.2). (b) *Field* mana yang memberi tahu sistem operasi server koneksi mana pemilik sebuah paket? (c) Dengan rentang efemeral di

Subbab 15.2.2, paling banyak berapa koneksi yang dapat dibuka laptop itu sekaligus ke satu port server tersebut?

Soal Latihan 15.2 (TCP atau UDP). Untuk masing-masing hal berikut, pilih TCP atau UDP dan berikan alasan satu kalimat berdasarkan **Tabel 15.1**: (a) layanan status pesanan; (b) setiap mesin cuci yang mengumumkan "saya masih hidup, sisa 23 menit" setiap dua detik; (c) gambar langsung kamera keamanan toko; (d) mengunggah struk hari ini sebagai PDF kepada akuntan; (e) mencari alamat `app.profitina.com`; (f) gim multipemain yang mengirim posisi setiap pemain tiga puluh kali per detik.

Soal Latihan 15.3 (membuatnya membeku). Ubah `EchoClient` agar mengirim dengan `out.print(userLine)` alih-alih `out.println(userLine)`. Ramalkan apa yang terjadi dan mengapa. Batalkan perubahan itu, lalu buat `PrintWriter` tanpa *autoflush*. Ramalkan lagi. Kemudian tambahkan satu baris pada klien yang mengubah kebekuan itu menjadi pesan galat setelah tiga detik, dan nyatakan bunyi pesannya di Java dan di Python.

Soal Latihan 15.4 (memperluas protokol di atas kertas lebih dahulu). Tambahkan perintah `help` pada protokol knock-knock: dalam keadaan apa pun, bila klien berkata `help`, server menjawab dengan petunjuk tentang apa yang diharapkannya berikutnya dan tetap dalam keadaan yang sama. (a) Tambahkan baris-baris baru pada **Tabel 15.3**. (b) Jelaskan bagaimana **Gambar 15.5** berubah. (c) Ubah `processInput` di kedua bahasa. (d) Tuliskan tiga baris klien yang menguji aturan baru itu.

Soal Latihan 15.5 (kumpulan yang terlalu kecil). Pada **Kode 15.13** dan **Kode 15.14**, ubah ukuran kumpulan thread server dari 3 menjadi 2, dengan tetap tiga klien. Sebelum menjalankannya, ramalkan apa yang dicetak setiap klien dan berapa lama program berjalan. Lalu jalankan dan jelaskan keluarannya dengan **Gambar 15.6**.

Soal Latihan 15.6 (biaya perjalanan pulang-pergi dan thread). Server binatu berada di pusat data dengan latensi satu arah 30 ms dari toko, melalui tautan 2 Mbit/s. Permintaan status berukuran 100 bita, tanggapannya 200 bita, dan server memerlukan 1 ms per pesanan. (a) Dengan Persamaan (15.4) dan Persamaan (15.5), berapa lama satu permintaan status? (b) Dasbor pemilik menampilkan 25 pesanan. Berapa lama bila ditanyakan satu per satu? Berapa lama dengan satu permintaan yang mengembalikan kedua puluh lima pesanan sekaligus (tanggapan sekitar 3000 bita, waktu server 25 ms)? (c) Server satu-thread-per-klien dengan ukuran tumpukan 512 KB harus menampung 5000 klien yang menganggur. Berapa memori yang dipesan tumpukannya, menurut Persamaan (15.3)?

Soal Latihan 15.7 (klien yang mencoba ulang). Tulislah, di Java dan di Python, metode `connectWithRetry(host, port, maxAttempts, afterFailure)` yang mencoba membuat `LaundryStatusClient`, dan setelah setiap koneksi yang ditolak menunggu 100 ms, lalu 200 ms, 400 ms, dan seterusnya sebelum mencoba lagi, serta menyerah setelah `maxAttempts` kali. Demonstrasikan dengan server yang baru dijalankan setelah kegagalan ketiga. Berapa total waktu tunggu klien sebelum percobaan ke-k?

Soal Latihan 15.8 (perintah baru tanpa data pribadi). Tambahkan perintah `{"cmd":"queue"}` pada layanan status pesanan yang menjawab berapa banyak pesanan dalam setiap status, untuk layar ruang tunggu, misalnya `{"ok":true,"RECEIVED":1,"WASHING":1,"DRYING":0,...}`. Ubah kedua server, tambahkan satu pemeriksaan pada setiap rangkaian uji, dan jelaskan mengapa perintah ini tidak memerlukan nomor telepon maupun token.

Soal Latihan 15.9 (kode status HTTP). Bila layanan status pesanan ditawarkan melalui HTTP, kode status mana dari **Tabel 15.5** yang semestinya dihasilkan setiap keadaan berikut? (a) Permintaan status yang sah untuk pesanan yang ada. (b) Nomor pesanan yang tidak ada. (c) Badan yang bukan JSON. (d) Perintah staf tanpa token. (e) Token sah milik kurir, yang boleh mengonfirmasi pengantaran tetapi tidak boleh memajukan pesanan melalui tahap pencucian. (f) Basis data di belakang server tidak dapat dijangkau. (g) Tebakan telepon keenam yang salah untuk satu pesanan dalam satu menit.

Soal Latihan 15.10 (tinjauan keamanan). Tinjaulah **Kode 15.17** seperti seorang penyerang. (a) Sebutkan empat kelemahan yang masih tersisa walaupun server itu mengikuti tujuh kebiasaan di Subbab 15.9.1 sejauh yang dilakukannya. (b) Rancanglah kelas `FailureLimiter` yang mengizinkan lima tebakan telepon yang salah per pesanan lalu menolak pesanan itu selama 60 detik; sebutkan di mana server memanggilnya dan bagaimana kelas itu harus dilindungi dari pemakaian bersamaan. (c) Pakai Persamaan (15.6) untuk membandingkan waktu tebak harapan tanpa dan dengan pembatas Anda.

Soal Latihan 15.11 (ruang obrolan). Tulislah, di Java dan di Python, server obrolan yang baris pertama kiriman kliennya adalah nama klien dan setiap baris berikutnya disiarkan kepada semua klien yang tersambung sebagai `name: text`, dengan pengumuman `* name joined` dan `* name left` serta perintah `/quit`. Demonstrasikan dengan tiga anggota berskenario yang transkripnya dicetak di akhir. Mengapa siaran harus memegang kunci selama mengirim, dan apa yang salah bila jaringan salah satu anggota sangat lambat?

Soal Latihan 15.12 (ringkasan proyek Anda sendiri). Tulislah ringkasan seperti di Subbab 15.11.10 untuk *Papan Barang Hilang dan Temuan Kampus*: mahasiswa mengirim barang temuan beserta deskripsi foto dan tempatnya, pemilik mengklaimnya, dan kantor keamanan mengonfirmasi serah terimanya. Sertakan *pitch*, sedikitnya lima fitur, komponen dengan pemilik untuk tim bertiga, tabel protokol dengan sedikitnya lima permintaan, tabel tonggak empat minggu, kriteria kinerja, data pribadi yang terlibat dan cara melindunginya, serta lima uji silang.

Soal Latihan 15.13 (konflik penggabungan). Ayu mengubah baris perintah jalan di README menjadi `java BoardServer 5060`, lalu meng-*commit* dan mendorongnya (*push*). Pada saat yang sama, di cabangnya sendiri, Bagus mengubah baris yang sama menjadi `java BoardServer 5050 --quiet` dan meng-*commit*-nya. Ketika Bagus menggabungkan `main` ke cabangnya, apa yang dicetak Git dan seperti apa README-nya? Selesaikan konfliknya sehingga kedua maksud itu bertahan, dan berikan perintah-perintahnya.

15.16 Penyelesaian Soal Latihan

Penyelesaian 15.1. (a) Ketiga koneksi itu adalah (192.168.1.20, 51234, 192.168.1.10, 5052), (192.168.1.20, 51235, 192.168.1.10, 5052), dan (192.168.1.30, 49800, 192.168.1.10, 5052). (b) Keempat *field* dipakai bersama; di sini *field* sisi server sama untuk ketiga koneksi, sehingga alamat klien membedakan tablet dari laptop, dan port klien membedakan dua koneksi laptop. (c) Alamat klien, alamat server, dan port server tetap, sehingga hanya p_c yang dapat berubah. Rentang efemeral 49152–65535 memuat $65\,535 - 49\,152 + 1 = 16\,384$ port, jadi paling banyak 16.384 koneksi bersamaan, batas yang hanya penting bagi alat uji beban. (Linux secara bawaan memakai rentang 32768–60999, yang memberi 28.232.)

Penyelesaian 15.2. (a) TCP: pesan status yang hilang atau tertukar urutannya akan memberi pelanggan jawaban yang salah. (b) UDP: detak yang hilang tidak berbahaya karena detak lain tiba dua detik kemudian, dan hilangnya serangkaian detak justru menjadi tanda bahwa mesin mati. (c)

UDP: bingkai video yang terlambat tidak berguna, dan mengirimnya ulang hanya menunda bingkai berikutnya. (d) TCP: setiap bita PDF harus tiba, berurutan. (e) UDP: satu pertanyaan kecil dan satu jawaban kecil, dan pertanyaan yang hilang cukup diajukan lagi (DNS beralih ke TCP untuk jawaban yang sangat besar). (f) UDP: posisi terbaru menggantikan yang lama tiga puluh kali per detik, sehingga mengirim ulang posisi lama tidak menolong siapa pun.

Penyelesaian 15.3. Dengan `print`, barisnya tidak memiliki baris baru, sehingga `readLine` server menunggu akhir baris sementara `readLine` klien menunggu gemanya: kebuntuan yang terbentang di antara dua program. Tanpa *autoflush*, baris yang lengkap tidak pernah meninggalkan penyangga klien, dengan akibat yang sama. Menambahkan `socket.setSoTimeout(3000);` setelah socket dibuka membuat `readLine` yang terblokir melempar `java.net.SocketTimeoutException` dengan pesan `Read timed out`, yang dicetak sebagai `I/O error: Read timed out`. Di Python, `sock.settimeout(3)` membuat `readline` melempar `TimeoutError` (disebut `socket.timeout` sebelum Python 3.10) dengan pesan `timed out`. Batas waktu tidak memperbaiki *bug*-nya; ia mengubah kebekuan yang tak terlihat menjadi galat yang terlihat dan dapat dijelaskan.

Penyelesaian 15.4. (a) Tiga baris baru, satu untuk setiap keadaan yang mengharapkan masukan:

Keadaan server	Klien berkata	Server menjawab	Keadaan berikutnya
`SENT_KNOCK_KNOCK`	`help`	`Say "Who's there?"`	`SENT_KNOCK_KNOCK`
`SENT_CLUE`	`help`	`Say "<clue> who?"`	`SENT_CLUE`
`ANOTHER`	`help`	`Say y for another joke, n to stop`	`ANOTHER`

(b) Ketiga keadaan itu masing-masing mendapat panah lingkar berlabel `"help" / petunjuk` yang kembali ke keadaan yang sama; tidak ada keadaan yang ditambahkan. (c) Perubahan yang paling bersih memeriksa `help` sekali, sebelum `switch`, sehingga aturannya ditulis sekali saja:

```
Java, di awal processInput:
    if (input != null && input.equalsIgnoreCase("help") && state != State.WAITING) {
        switch (state) {
            case SENT_KNOCK_KNOCK: return "Say \"Who's there?\"";
            case SENT_CLUE:        return "Say \"" + CLUES[joke] + " who?\"";
            default:               return "Say y for another joke, n to stop";
        }
    }

Python, di awal process_input:
    if text is not None and text.lower() == "help" and self._state is not State.WAITING:
        if self._state is State.SENT_KNOCK_KNOCK:
            return "Say \"Who's there?\""
        if self._state is State.SENT_CLUE:
            return f"Say \"{CLUES[self._joke]} who?\""
        return "Say y for another joke, n to stop"
```

(d) Skenario `help`, `Who's there?`, `help`, `Lettuce who?`, `help`, `n` harus menghasilkan, setelah `Knock! Knock!`: `Say "Who's there?"`, `Lettuce`, `Say "Lettuce who?"`, jawabannya dengan `Want another? (y/n)`, `Say y for another joke, n to stop`, `Bye.`. Perhatikan bahwa sebelum perubahan ini, `help` dalam keadaan `ANOTHER` berarti "berhenti": perubahan protokol dapat mengubah makna pesan lama, dan karena itulah kedua pihak harus menyepakatinya di atas kertas.

Penyelesaian 15.5. Ramalan: kedua pekerja menyapa dua klien, yang lalu menunggu di penghalang untuk klien ketiga yang tidak pernah disapa, karena koneksinya tertahan di antrean kumpulan tanpa pekerja yang bebas. Setelah lima detik penghalang habis waktu; satu klien yang menunggu mendapat eksepsi batas waktu, yang lain mendapat penghalang yang rusak, dan batas

waktu baca lima detik milik klien ketiga pun sudah habis. Program berjalan sekitar lima detik dan melaporkan `false`. Satu run versi Java mencetak:

```
server started on an ephemeral port
--- client A ---
failed: TimeoutException
--- client B ---
failed: SocketTimeoutException
--- client C ---
failed: BrokenBarrierException
server stopped; clients served: 3
all clients were greeted before any finished: false
```

dan versi Python:

```
server started on an ephemeral port
--- client A ---
failed: BrokenBarrierError
--- client B ---
failed: BrokenBarrierError
--- client C ---
failed: BrokenBarrierError
server stopped; clients served: 3
all clients were greeted before any finished: false
```

Klien mana mendapat eksepsi apa berubah dari run ke run; putusan `false` tidak. Di Python, batas waktu pada `threading.Barrier` merusaknya untuk semua pihak, sehingga ketiganya melaporkan `BrokenBarrierError`. Kumpulan thread yang lebih kecil daripada banyaknya percakapan yang harus aktif *bersamaan* berubah menjadi antrean, dan antrean di belakang percakapan yang menunggu anggota lain dari antrean itu adalah kebuntuan: para filsuf makan malam dari Bab 14 dalam kostum baru.

Penyelesaian 15.6. (a) Menurut Persamaan (15.5), $\text{RTT} = 2 \times 30\ \text{ms} + \frac{(100+200)\times 8\ \text{bit}}{2\times 10^6\ \text{bit/s}} = 60\ \text{ms} + 1.2\ \text{ms} = 61.2\ \text{ms}$, dan satu permintaan memakan $61.2 + 1 = 62.2$ ms. (b) Satu per satu: $25 \times 62.2 = 1555$ ms, sekitar 1,6 detik, cukup lama untuk dirasakan pemilik. Satu permintaan gabungan: $\text{RTT} = 60 + \frac{(100+3000)\times 8}{2\times 10^6} \times 1000 = 60 + 12.4 = 72.4$ ms, ditambah 25 ms waktu server, sekitar 97 ms: enam belas kali lebih cepat, walaupun membawa lebih banyak bita. (c) Menurut Persamaan (15.3), $M \approx 5000 \times 512\ \text{KB} = 2\,560\,000\ \text{KB}$, sekitar 2,4 GiB tumpukan yang dipesan oleh thread yang tidak mengerjakan apa pun; karena itulah server semacam itu memerlukan kumpulan thread, *virtual thread*, atau `asyncio`.

Penyelesaian 15.7. **Kode 15.27** dan **Kode 15.28** adalah kedua programnya. `main` lebih dahulu memperoleh port bebas lalu menutupnya, sehingga tidak ada yang mendengarkan di sana, dan fungsi panggil-balik (*callback*) menjalankan server pada port itu tepat setelah kegagalan ketiga; tidak diperlukan trik pengaturan waktu.

Kode 15.27 `RetryDemo.java`: menyambung dengan *backoff* eksponensial; server muncul setelah kegagalan ketiga.

```
import java.io.IOException;
import java.net.ConnectException;
import java.net.InetAddress;
import java.net.ServerSocket;
import java.util.function.IntConsumer;

/** Klien yang tidak menyerah pada "connection refused" yang pertama. */
public class RetryDemo {
    /** Mencoba hingga maxAttempts kali, menunggu 100, 200, 400, ... ms di antara
percobaan. */
    static LaundryStatusClient connectWithRetry(String host, int port, int maxAttempts,
            IntConsumer afterFailure) throws IOException, InterruptedException {
        long delayMs = 100;
        for (int attempt = 1; ; attempt++) {
            try {
                LaundryStatusClient client = new LaundryStatusClient(host, port);
                System.out.println("attempt " + attempt + ": connected");
                return client;
            } catch (ConnectException e) {
                System.out.println("attempt " + attempt + ": server not reachable");
                if (attempt == maxAttempts) {
                    throw e;
                }
                afterFailure.accept(attempt);
                System.out.println("  waiting " + delayMs + " ms");
                Thread.sleep(delayMs);
                delayMs *= 2;  // backoff eksponensial: jangan menggempur server yang
sedang mati
            }
        }
    }

    public static void main(String[] args) throws Exception {
        int port;
        try (ServerSocket probe = new ServerSocket(0, 1, InetAddress.getLoopbackAddress()))
{
            port = probe.getLocalPort();  // port bebas; setelah ini tak ada yang
mendengarkan di sana
        }
        LaundryStatusServer server = LaundryStatusServer.withSampleOrders(null, false);
```

Kode 15.27 (Lanjutan) *RetryDemo.java: menyambung dengan backoff eksponensial; server muncul setelah kegagalan ketiga.*

```
        IntConsumer startServerAfterThirdFailure = attempt -> {
            if (attempt == 3) {
                try {
                    server.start(port);
                    System.out.println("  (the server comes up now)");
                } catch (IOException e) {
                    System.out.println("  (the server could not start: " + e.getMessage() +
")");
                }
            }
        };
        try (LaundryStatusClient client = connectWithRetry("127.0.0.1", port, 5,
                startServerAfterThirdFailure)) {
            System.out.println("ping -> " + client.ping().get("reply"));
        }
        server.stop();
        System.out.println("server stopped");

        try {
            connectWithRetry("127.0.0.1", port, 2, attempt -> { });
        } catch (ConnectException e) {
            System.out.println("gave up after 2 attempts; tell the user the service is
down");
        }
    }
}
```

```
attempt 1: server not reachable
  waiting 100 ms
attempt 2: server not reachable
  waiting 200 ms
attempt 3: server not reachable
  (the server comes up now)
  waiting 400 ms
attempt 4: connected
ping -> pong
server stopped
attempt 1: server not reachable
  waiting 100 ms
attempt 2: server not reachable
gave up after 2 attempts; tell the user the service is down
```

Kode 15.28 `retry_demo.py`: klien yang mencoba ulang yang sama dalam Python.

```
"""retry_demo.py: klien yang tidak menyerah pada "connection refused" yang pertama."""
import socket
import time

from laundry_status_client import LaundryStatusClient
from laundry_status_server import LaundryStatusServer, OrderBook


def connect_with_retry(host, port, max_attempts, after_failure):
    """Coba hingga max_attempts kali, menunggu 100, 200, 400, ... ms di antara
percobaan."""
    delay_ms = 100
    attempt = 1
    while True:
        try:
            client = LaundryStatusClient(host, port)
            print(f"attempt {attempt}: connected")
            return client
        except ConnectionRefusedError:
            print(f"attempt {attempt}: server not reachable")
            if attempt == max_attempts:
                raise
            after_failure(attempt)
            print(f"  waiting {delay_ms} ms")
            time.sleep(delay_ms / 1000)
            delay_ms *= 2  # backoff eksponensial: jangan menggempur server yang sedang
mati
            attempt += 1


def main():
    with socket.socket() as probe:
        probe.bind(("127.0.0.1", 0))
        port = probe.getsockname()[1]  # port bebas; setelah ini tak ada yang mendengarkan
di sana
    server = LaundryStatusServer(OrderBook.sample(), None, verbose=False)

    def start_server_after_third_failure(attempt):
        if attempt == 3:
            server.start(port)
            print("  (the server comes up now)")

    with connect_with_retry("127.0.0.1", port, 5, start_server_after_third_failure) as
client:
        print("ping -> " + client.ping()["reply"])
    server.stop()
    print("server stopped")
```

Kode 15.28 (Lanjutan) *retry_demo.py: klien yang mencoba ulang yang sama dalam Python.*

```
    try:
        connect_with_retry("127.0.0.1", port, 2, lambda attempt: None)
    except ConnectionRefusedError:
        print("gave up after 2 attempts; tell the user the service is down")


if __name__ == "__main__":
    main()
```

```
attempt 1: server not reachable
  waiting 100 ms
attempt 2: server not reachable
  waiting 200 ms
attempt 3: server not reachable
  (the server comes up now)
  waiting 400 ms
attempt 4: connected
ping -> pong
server stopped
attempt 1: server not reachable
  waiting 100 ms
attempt 2: server not reachable
gave up after 2 attempts; tell the user the service is down
```

Sebelum percobaan ke-k, klien telah menunggu $100 + 200 + \cdots + 100 \cdot 2^{k-2} = 100\,(2^{k-1} - 1)$ ms: 700 ms sebelum percobaan keempat, 1,5 detik sebelum yang kelima. *Backoff eksponensial* ini mencegah kerumunan klien menggempur server yang sedang dijalankan ulang; klien produksi juga menambahkan *jitter* acak dan membatasi lama penantian maksimum. Dan setelah `maxAttempts`, klien menyerah dan mengatakannya.

Penyelesaian 15.8. Tambahkan pada `OrderBook` metode yang menghitung di bawah kunci yang sama dengan metode lainnya, lalu satu cabang di `handle`:

```
Java (OrderBook):
    synchronized Map<String, Object> countByStatus() {
        Map<String, Object> counts = Json.object("ok", true);
        for (Status s : Status.values()) {
            counts.put(s.name(), 0L);
        }
        for (LaundryOrder o : orders.values()) {
            counts.put(o.status.name(), (Long) counts.get(o.status.name()) + 1);
        }
        return counts;
    }
Java (handle):   } else if ("queue".equals(cmd)) { return book.countByStatus(); }

Python (OrderBook):
    def count_by_status(self):
        with self._lock:
            counts = {"ok": True, **{s.name: 0 for s in Status}}
            for o in self._orders.values():
                counts[o.status.name] += 1
            return counts
Python (handle):  if cmd == "queue": return self._book.count_by_status()

Pemeriksaan / uji:
    expect(1L, c.send(Json.object("cmd", "queue")).get("WASHING"));          // Java
    self.assertEqual(1, c.send({"cmd": "queue"})["WASHING"])                  # Python
```

Dengan pesanan contoh, kedua server menjawab `{"ok":true,"RECEIVED":1,"WASHING":1,"DRYING":0,"IRONING":0,"READY":1,"PICKED_UP":0}`. Setiap status dimulai dari nol agar layar selalu menerima kunci yang sama, dan Java memakai `Long` karena itulah yang dihasilkan `Json.parse` di sisi klien. Perintah ini tidak memerlukan digit telepon maupun token karena jawabannya berupa agregat dan tidak membocorkan apa pun tentang orang atau pesanan tertentu: minimalisasi data (**Tabel 15.6**) sebagai aturan rancangan.

Penyelesaian 15.9. (a) 200 OK. (b) 404 Not Found; mengikuti kebiasaan keenam di Subbab 15.9.1, nomor telepon yang salah semestinya mendapat 404 dan badan yang sama. (c) 400 Bad Request. (d) 401 Unauthorized: kredensial tidak ada. (e) 403 Forbidden: kredensial sah, tetapi haknya tidak mencukupi. (f) 503 Service Unavailable, gangguan sementara di luar permintaan,

yang memberi tahu klien untuk mencoba lagi nanti (500 juga dapat dipertanggungjawabkan). (g) 429 Too Many Requests, idealnya dengan header `Retry-After`.

Penyelesaian 15.10. (a) *Tanpa enkripsi*: siapa pun di jaringan dapat membaca nomor pesanan, digit telepon, dan token staf (perbaikan: TLS). *Tanpa pembatasan laju*: empat digit jebol dalam hitungan detik menurut Persamaan (15.6). *Satu token staf untuk bersama*: log tidak dapat menyebut siapa yang memajukan pesanan, dan karyawan yang keluar tetap membawa rahasianya (perbaikan: token pribadi yang dapat dicabut). *Nomor pesanan yang dapat ditebak*: penebak cukup berjalan dari ORD-0001 ke atas (perbaikan: kode acak, atau pembatasan laju). (b) Sketsanya:

```
class FailureLimiter {                        // satu objek bersama, dipakai setiap
pekerja
    private final Map<String, Integer> failures = new HashMap<>();
    private final Map<String, Long> lockedUntil = new HashMap<>();

    synchronized boolean isLocked(String order, long nowMs) {
        Long until = lockedUntil.get(order);
        return until != null && nowMs < until;
    }

    synchronized void recordFailure(String order, long nowMs) {
        int n = failures.merge(order, 1, Integer::sum);
        if (n >= 5) {
            lockedUntil.put(order, nowMs + 60_000);
            failures.remove(order);
        }
    }

    synchronized void recordSuccess(String order) {
        failures.remove(order);
    }
}
```

Di `handle`, cabang `status` lebih dahulu memanggil `isLocked(order, now)` dan, bila terkunci, menolak dengan pesan yang biasa (melalui HTTP, 429); bila tidak, ia memanggil `publicView`, lalu `recordFailure` atau `recordSuccess`. Waktu dikirim sebagai parameter agar kelas ini dapat diuji tanpa menunggu satu menit. Metode-metodenya `synchronized` karena beberapa pekerja dapat mencatat kegagalan untuk pesanan yang sama sekaligus, dan dua kegagalan bersamaan tidak boleh terhitung sebagai satu, pembaruan yang hilang dari Bab 14. Versi produksi juga akan melupakan entri lama. (c) Tanpa pembatas, pada $r = 1000$ per detik, Persamaan (15.6) memberi sekitar 5 detik; dengan pembatas, $r = 1/12$ per detik memberi $5000.5 \times 12 \approx 60\,000$ detik, hampir 17 jam, sementara log dipenuhi penolakan yang dapat disadari staf.

Penyelesaian 15.11. **Kode 15.29** dan **Kode 15.30** memberi setiap anggota satu thread, menyimpan objek *writer* milik para anggota dalam sebuah peta, dan menyiarkan setiap baris kepada semuanya. Anggota berskenario dalam demonstrasi mencatat apa yang diterimanya, dan setiap langkah menunggu sampai setiap anggota menerima apa yang semestinya, sehingga keluarannya sama pada setiap run.

Kode 15.29 `ChatRoom.java`: ruang obrolan yang menyiarkan setiap baris, dengan anggota berskenario untuk demonstrasi yang dapat diulang.

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.net.InetAddress;
import java.net.ServerSocket;
import java.net.Socket;
import java.nio.charset.StandardCharsets;
import java.util.ArrayList;
import java.util.LinkedHashMap;
import java.util.List;
import java.util.Map;
import java.util.Set;
import java.util.concurrent.ConcurrentHashMap;

/** Ruang obrolan: setiap baris yang dikirim satu anggota disiarkan ke semua anggota. */
public class ChatRoom {
    private final Map<String, PrintWriter> members = new LinkedHashMap<>();  // dijaga oleh
this
    private final Set<Socket> sockets = ConcurrentHashMap.newKeySet();
    private ServerSocket serverSocket;

    public int start(int port) throws IOException {
        serverSocket = new ServerSocket(port, 50, InetAddress.getLoopbackAddress());
        Thread acceptor = new Thread(() -> {
            while (true) {
                try {
                    Socket socket = serverSocket.accept();
                    new Thread(() -> serve(socket)).start();  // satu thread per anggota
                } catch (IOException e) {
                    return;  // stop() telah menutup socket server
                }
            }
        });
        acceptor.start();
        return serverSocket.getLocalPort();
    }
```

Kode 15.29 (Lanjutan) *ChatRoom.java: ruang obrolan yang menyiarkan setiap baris, dengan anggota berskenario untuk demonstrasi yang dapat diulang.*

```
    private void serve(Socket socket) {
        sockets.add(socket);
        String name = null;
        try (socket;
             BufferedReader in = new BufferedReader(new InputStreamReader(
                     socket.getInputStream(), StandardCharsets.UTF_8));
             PrintWriter out = new PrintWriter(new OutputStreamWriter(
                     socket.getOutputStream(), StandardCharsets.UTF_8), true)) {
            String wanted = in.readLine();  // protokol: baris pertama adalah nama anggota
            if (wanted == null || !join(wanted, out)) {
                out.println("! name missing or taken");
                return;
            }
            name = wanted;
            broadcast("* " + name + " joined");
            String line;
            while ((line = in.readLine()) != null && !line.equals("/quit")) {
                broadcast(name + ": " + line);
            }
        } catch (IOException e) {
            // anggota lenyap; perlakukan persis seperti /quit
        } finally {
            sockets.remove(socket);
            if (name != null) {
                leave(name);
                broadcast("* " + name + " left");
            }
        }
    }

    private synchronized boolean join(String name, PrintWriter out) {
        if (name.isBlank() || members.containsKey(name)) {
            return false;
        }
        members.put(name, out);
        return true;
    }

    private synchronized void leave(String name) {
        members.remove(name);
    }
```

Kode 15.29 (Lanjutan) *ChatRoom.java: ruang obrolan yang menyiarkan setiap baris, dengan anggota berskenario untuk demonstrasi yang dapat diulang.*

```
    /** Memegang kunci selama mengirim berarti semua anggota melihat baris dalam urutan
yang sama. */
    private synchronized void broadcast(String line) {
        for (PrintWriter out : members.values()) {
            out.println(line);
        }
    }

    public void stop() throws IOException {
        serverSocket.close();
        for (Socket socket : sockets) {
            socket.close();
        }
    }

    /** Anggota berskenario untuk demonstrasi: ia mencatat setiap baris yang diterimanya.
*/
    static class Member implements AutoCloseable {
        final String name;
        private final Socket socket;
        private final PrintWriter out;
        private final List<String> received = new ArrayList<>();

        Member(String name, int port) throws IOException {
            this.name = name;
            socket = new Socket(InetAddress.getLoopbackAddress(), port);
            out = new PrintWriter(new OutputStreamWriter(
                    socket.getOutputStream(), StandardCharsets.UTF_8), true);
            BufferedReader in = new BufferedReader(new InputStreamReader(
                    socket.getInputStream(), StandardCharsets.UTF_8));
            Thread listener = new Thread(() -> {
                try {
                    String line;
                    while ((line = in.readLine()) != null) {
                        synchronized (this) {
                            received.add(line);
                            notifyAll();
                        }
                    }
                } catch (IOException e) {
                    // socket tertutup: tidak ada lagi yang terdengar
                }
            });
            listener.setDaemon(true);
            listener.start();
            say(name);
        }
```

Kode 15.29 (Lanjutan) *ChatRoom.java: ruang obrolan yang menyiarkan setiap baris, dengan anggota berskenario untuk demonstrasi yang dapat diulang.*

```
        void say(String line) {
            out.println(line);
        }

        /** Menunggu (paling lama 5 s) sampai anggota ini menerima total n baris. */
        synchronized void awaitLines(int n) throws InterruptedException {
            long deadline = System.currentTimeMillis() + 5000;
            while (received.size() < n && System.currentTimeMillis() < deadline) {
                wait(100);
            }
        }

        synchronized List<String> received() {
            return new ArrayList<>(received);
        }

        @Override
        public void close() throws IOException {
            socket.close();
        }
    }

    public static void main(String[] args) throws Exception {
        ChatRoom room = new ChatRoom();
        int port = room.start(0);
        Member ayu = new Member("Ayu", port);
        ayu.awaitLines(1);
        Member budi = new Member("Budi", port);
        ayu.awaitLines(2);
        budi.awaitLines(1);
        ayu.say("Is machine W2 free?");
        budi.awaitLines(2);
        budi.say("Yes, go ahead.");
        ayu.awaitLines(4);
        Member citra = new Member("Citra", port);
        ayu.awaitLines(5);
        citra.say("Hi all!");
        budi.awaitLines(5);
        budi.say("/quit");
        citra.awaitLines(3);
        ayu.say("Bye, Budi!");
        ayu.awaitLines(8);
        citra.awaitLines(4);
```

Kode 15.29 (Lanjutan) *ChatRoom.java: ruang obrolan yang menyiarkan setiap baris, dengan anggota berskenario untuk demonstrasi yang dapat diulang.*

```
        for (Member m : List.of(ayu, budi, citra)) {
            System.out.println("--- what " + m.name + " saw ---");
            m.received().forEach(System.out::println);
        }
        for (Member m : List.of(ayu, budi, citra)) {
            m.close();  // baru sekarang: menutup lebih awal akan menambah "* Ayu left"
pada tampilan Citra
        }
        room.stop();
    }
}
```

```
--- what Ayu saw ---
* Ayu joined
* Budi joined
Ayu: Is machine W2 free?
Budi: Yes, go ahead.
* Citra joined
Citra: Hi all!
* Budi left
Ayu: Bye, Budi!
--- what Budi saw ---
* Budi joined
Ayu: Is machine W2 free?
Budi: Yes, go ahead.
* Citra joined
Citra: Hi all!
--- what Citra saw ---
* Citra joined
Citra: Hi all!
* Budi left
Ayu: Bye, Budi!
```

Kode 15.30 `chat_room.py`: ruang obrolan dalam Python.

```
"""chat_room.py: setiap baris yang dikirim satu anggota disiarkan ke semua anggota."""
import socket
import sys
import threading
import time


class ChatRoom:
    def __init__(self):
        self._members = {}  # nama -> penulis, dijaga oleh self._lock
        self._lock = threading.Lock()
        self._sockets = set()
        self._server_socket = None

    def start(self, port):
        self._server_socket = socket.socket()
        if sys.platform != "win32":  # pakai ulang port segera setelah dijalankan ulang
            self._server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
        self._server_socket.bind(("127.0.0.1", port))
        self._server_socket.listen(50)
        threading.Thread(target=self._accept_loop, daemon=True).start()
        return self._server_socket.getsockname()[1]

    def _accept_loop(self):
        while True:
            try:
                conn, _ = self._server_socket.accept()
            except OSError:
                return  # stop() telah menutup socket server
            # satu thread per anggota
            threading.Thread(target=self._serve, args=(conn,), daemon=True).start()

    def _serve(self, conn):
        self._sockets.add(conn)
        name = None
        try:
            with conn, conn.makefile("r", encoding="utf-8", newline="\n") as reader, \
                    conn.makefile("w", encoding="utf-8", newline="\n") as writer:
                wanted = reader.readline().rstrip("\n")  # protokol: baris pertama adalah
nama
                if not self._join(wanted, writer):
                    writer.write("! name missing or taken\n")
                    writer.flush()
                    return
                name = wanted
                self._broadcast(f"* {name} joined")
                for line in reader:
```

Kode 15.30 (Lanjutan) *chat_room.py: ruang obrolan dalam Python.*

```
                    line = line.rstrip("\n")
                    if line == "/quit":
                        break
                    self._broadcast(f"{name}: {line}")
            except OSError:
                pass  # anggota lenyap; perlakukan persis seperti /quit
            finally:
                self._sockets.discard(conn)
                if name is not None:
                    self._leave(name)
                    self._broadcast(f"* {name} left")

    def _join(self, name, writer):
        with self._lock:
            if not name.strip() or name in self._members:
                return False
            self._members[name] = writer
            return True

    def _leave(self, name):
        with self._lock:
            self._members.pop(name, None)

    def _broadcast(self, line):
        """Memegang kunci selama mengirim berarti semua anggota melihat baris dalam urutan
yang sama."""
        with self._lock:
            for writer in self._members.values():
                try:
                    writer.write(line + "\n")
                    writer.flush()
                except OSError:
                    pass  # anggota itu sedang pergi; thread-nya sendiri yang membereskan

    def stop(self):
        self._server_socket.close()
        for conn in list(self._sockets):
            try:
                conn.shutdown(socket.SHUT_RDWR)
            except OSError:
                pass


class Member:
    """Anggota berskenario untuk demonstrasi: ia mencatat setiap baris yang diterimanya."""
```

Kode 15.30 (Lanjutan) *chat_room.py: ruang obrolan dalam Python.*

```
    def __init__(self, name, port):
        self.name = name
        self._sock = socket.create_connection(("127.0.0.1", port))
        self._writer = self._sock.makefile("w", encoding="utf-8", newline="\n")
        self._reader = self._sock.makefile("r", encoding="utf-8", newline="\n")
        self._received = []
        self._cond = threading.Condition()
        threading.Thread(target=self._listen, daemon=True).start()
        self.say(name)

    def _listen(self):
        try:
            for line in self._reader:
                with self._cond:
                    self._received.append(line.rstrip("\n"))
                    self._cond.notify_all()
        except (OSError, ValueError):
            pass  # socket tertutup: tidak ada lagi yang terdengar

    def say(self, line):
        self._writer.write(line + "\n")
        self._writer.flush()

    def await_lines(self, n):
        """Tunggu (paling lama 5 s) sampai anggota ini menerima total n baris."""
        deadline = time.monotonic() + 5
        with self._cond:
            while len(self._received) < n and time.monotonic() < deadline:
                self._cond.wait(0.1)

    def received(self):
        with self._cond:
            return list(self._received)

    def close(self):
        self._sock.close()
```

Kode 15.30 (Lanjutan) *chat_room.py: ruang obrolan dalam Python.*

```
def main():
    room = ChatRoom()
    port = room.start(0)
    ayu = Member("Ayu", port)
    ayu.await_lines(1)
    budi = Member("Budi", port)
    ayu.await_lines(2)
    budi.await_lines(1)
    ayu.say("Is machine W2 free?")
    budi.await_lines(2)
    budi.say("Yes, go ahead.")
    ayu.await_lines(4)
    citra = Member("Citra", port)
    ayu.await_lines(5)
    citra.say("Hi all!")
    budi.await_lines(5)
    budi.say("/quit")
    citra.await_lines(3)
    ayu.say("Bye, Budi!")
    ayu.await_lines(8)
    citra.await_lines(4)

    for m in (ayu, budi, citra):
        print(f"--- what {m.name} saw ---")
        for line in m.received():
            print(line)
    for m in (ayu, budi, citra):
        m.close()  # baru sekarang: menutup lebih awal akan menambah "* Ayu left" pada
tampilan Citra
    room.stop()


if __name__ == "__main__":
    main()
```

```
--- what Ayu saw ---
* Ayu joined
* Budi joined
Ayu: Is machine W2 free?
Budi: Yes, go ahead.
* Citra joined
Citra: Hi all!
* Budi left
Ayu: Bye, Budi!
--- what Budi saw ---
* Budi joined
Ayu: Is machine W2 free?
Budi: Yes, go ahead.
* Citra joined
Citra: Hi all!
--- what Citra saw ---
* Citra joined
Citra: Hi all!
* Budi left
Ayu: Bye, Budi!
```

Setiap anggota hanya melihat apa yang dikatakan selama ia hadir. `broadcast` memegang kunci ruang karena dua alasan. Peta anggota tidak boleh berubah selagi ditelusuri (kalau tidak, muncul `ConcurrentModificationException` di Java, “dictionary changed size during iteration” di Python). Dan ketika dua anggota berbicara bersamaan, satu siaran selesai sebelum yang lain dimulai, sehingga semua orang melihat urutan yang sama. Harganya adalah masalah anggota lambat: menulis kepada anggota yang penyangga socket-nya penuh akan terblokir sambil memegang kunci, sehingga *semua orang* menunggu anggota yang paling lambat. Server obrolan sungguhan

memberi setiap anggota antrean keluaran dan thread pengirim sendiri (pola produsen–konsumen dari Bab 14) dan memutus anggota yang antreannya tumbuh terlalu panjang.

Penyelesaian 15.12. Sebuah ringkasan yang sudah dikerjakan:

Judul dan pitch. Papan Barang Hilang dan Temuan Kampus. Setiap minggu mahasiswa kehilangan kartu, *flash drive*, dan payung, sementara kantor keamanan menyimpan daftar kertas yang tak dibaca siapa pun. Penemu mengirim barang, setiap layar langsung menampilkannya, pemilik mengklaim barang dengan mendeskripsikannya, dan kantor keamanan mengonfirmasi serah terima.

Fitur. F1 kirim barang temuan (kategori, deskripsi, tempat, waktu); F2 daftar langsung barang yang belum diklaim, disaring per kategori; F3 klaim dengan rincian pribadi yang hanya diketahui pemilik; F4 kantor keamanan membandingkan lalu mengonfirmasi atau menolak; F5 barang yang tak diklaim setelah 60 hari ditandai untuk disumbangkan; F6 data bertahan melewati *restart* (SQLite).

Komponen dan pemilik. Anggota 1: server dan basis data di balik antarmuka `ItemRepository`. Anggota 2: pustaka klien, protokol, penyambungan ulang. Anggota 3: GUI untuk mahasiswa dan kantor keamanan, koordinator pengujian.

Protokol (baris JSON).

Permintaan	Tanggapan	Penolakan
`{"cmd":"login","nim":…,"pin":…}`	`{"ok":true,"role":"student"}` atau `"security"`	kredensial salah (pesan sama untuk kedua *field*); terlalu banyak percobaan
`{"cmd":"post","category":…,"description":…,"place":…}`	`{"ok":true,"item":"F-1043"}`	*field* tidak ada; deskripsi lebih dari 300 karakter
`{"cmd":"list","category":…}`	`{"ok":true,"items":[…]}` (tanpa identitas penemu)	kategori tidak dikenal
`{"cmd":"claim","item":…,"detail":…}`	`{"ok":true,"claim":"C-77"}`	barang sudah diserahkan
`{"cmd":"decide","claim":…,"accept":true}`	`{"ok":true}`	bukan peran keamanan
`{"cmd":"subscribe"}`	lalu didorong `{"event":"posted",…}` / `{"event":"closed",…}`	—
`{"cmd":"bye"}`	`{"ok":true}`, lalu tutup	—

Tonggak. Minggu 1: spesifikasi, protokol, diagram keadaan sesi, diagram komponen, repositori. Minggu 2: server, pustaka klien, dan GUI masing-masing di atas tiruan. Minggu 3: integrasi, F1–F4, uji beban 30 klien. Minggu 4: F5–F6, demonstrasi kegagalan di tiga komputer, dokumentasi, gladi.

Kriteria kinerja. 30 klien bersamaan; kiriman baru tampil di semua layar dalam 2 detik melalui Wi-Fi kampus; tanggapan daftar dalam 300 ms untuk 500 barang.

Data pribadi. NIM dan PIN (PIN hanya disimpan sebagai *hash* bergaram, tidak pernah dicatat di log); penemu tidak pernah ditampilkan kepada mahasiswa lain; rincian klaim hanya terlihat oleh peran keamanan; barang yang sudah selesai dihapus setelah 90 hari; data uji karangan.

Uji silang. (1) Anggota 2: server menolak `decide` dari akun mahasiswa. (2) Anggota 2: `list` tidak pernah memuat NIM penemu. (3) Anggota 1: klien menyambung ulang dan berlangganan ulang setelah server dijalankan ulang. (4) Anggota 1: klien melaporkan, bukan melempar, penolakan untuk deskripsi 301 karakter. (5) Anggota 3: dengan 30 klien berskenario, sebuah kiriman sampai ke semuanya dalam 2 detik.

Penyelesaian 15.13. Ketika Bagus menjalankan `git merge main`, Git mencetak

```
Auto-merging README.md
CONFLICT (content): Merge conflict in README.md
Automatic merge failed; fix conflicts and then commit the result.
```

dan README kini memuat kedua versi di antara penanda:

```
<<<<<<< HEAD
    cd server && javac *.java && java BoardServer 5050 --quiet
=======
    cd server && javac *.java && java BoardServer 5060
>>>>>>> main
```

`HEAD` adalah cabang Bagus dan `main` adalah yang digabungkannya. Kedua maksud muat dalam satu baris, `cd server && javac *.java && java BoardServer 5060 --quiet`. Bagus mengganti kelima baris itu dengan baris tersebut, menghapus penanda-penandanya, memeriksa dengan `git diff`, lalu `git add README.md` dan `git commit`, menjalankan program dan ujinya sebelum `git push`, dan memberi tahu Ayu, karena konflik berarti dua orang memutuskan hal yang sama tanpa saling berbicara.

Daftar Pustaka

[1] J. F. Kurose and K. W. Ross, *Computer Networking: A Top-Down Approach*, 8th ed. Hoboken, NJ, USA: Pearson, 2021.

[2] S. Deering and R. Hinden, “Internet Protocol, Version 6 (IPv6) Specification,” RFC 8200, IETF, Jul. 2017.

[3] M. Cotton, L. Eggert, J. Touch, M. Westerlund, and S. Cheshire, “Internet Assigned Numbers Authority (IANA) Procedures for the Management of the Service Name and Transport Protocol Port Number Registry,” RFC 6335, IETF, Aug. 2011.

[4] W. Eddy, Ed., “Transmission Control Protocol (TCP),” RFC 9293, IETF, Aug. 2022.

[5] J. Postel, “User Datagram Protocol,” RFC 768, IETF, Aug. 1980.

[6] F. Yergeau, “UTF-8, a transformation format of ISO 10646,” RFC 3629, IETF, Nov. 2003.

[7] Python Software Foundation, “socket — Low-level networking interface,” and G. McMillan, “Socket Programming HOWTO,” *The Python Standard Library*, version 3.14. [Online]. Available: https://docs.python.org/3/library/socket.html

[8] W. R. Stevens, B. Fenner, and A. M. Rudoff, *UNIX Network Programming, Volume 1: The Sockets Networking API*, 3rd ed. Boston, MA, USA: Addison-Wesley, 2004.

[9] Oracle, “Lesson: All About Sockets,” *The Java Tutorials*, Custom Networking trail. [Online]. Available: https://docs.oracle.com/javase/tutorial/networking/sockets/

[10] R. Pressler and A. Bateman, “JEP 444: Virtual Threads,” OpenJDK, 2023. [Online]. Available: https://openjdk.org/jeps/444

[11] T. Bray, Ed., “The JavaScript Object Notation (JSON) Data Interchange Format,” RFC 8259, IETF, Dec. 2017.

[12] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Reading, MA, USA: Addison-Wesley, 1994.

[13] R. Fielding, M. Nottingham, and J. Reschke, Eds., “HTTP Semantics,” RFC 9110, IETF, Jun. 2022.

[14] Republik Indonesia, *Undang-Undang Nomor 27 Tahun 2022 tentang Pelindungan Data Pribadi* (Law No. 27 of 2022 on Personal Data Protection), Lembaran Negara Tahun 2022 Nomor 196, Oct. 2022.

[15] E. Rescorla, “The Transport Layer Security (TLS) Protocol Version 1.3,” RFC 8446, IETF, Aug. 2018.

[16] S. Chacon and B. Straub, *Pro Git*, 2nd ed. New York, NY, USA: Apress, 2014. [Online]. Available: https://git-scm.com/book

Bab 16

Evaluasi dan Pembahasan IV: Iterator, Konkurensi, Jaringan, dan Keseluruhan Buku

Inilah bab terakhir buku ini, dan seperti Bab 4, 8, dan 12, bab ini tidak menambahkan fitur bahasa yang baru. Bab ini mengerjakan soal, dan kali ini soalnya besar. Delapan soal pertama berasal dari satu tugas proyek tim: membangun aplikasi klien–server dengan antarmuka grafis yang berkomunikasi melalui soket. Kita menyelesaikan tugas itu seperti yang sungguh-sungguh dilakukan sebuah tim kecil, satu hasil kerja demi satu hasil kerja, pada contoh berjalan yang cukup kecil untuk dicetak utuh: server lelucon *knock-knock* dengan klien konsol dan klien berjendela. Di sepanjang jalan kita memutuskan siapa mengerjakan apa, menulis antarmuka di antara para anggota, mengubah diagram alir menjadi mesin keadaan, melayani banyak klien sekaligus, menguji setiap sisi jaringan terhadap tiruan (*stub*) sisi lainnya, membuat kedua sisi menyadari ketika pihak lain menghilang, menjaga jendela tetap tanggap selagi jaringan menunggu, dan mengukur angka-angka yang dijanjikan dokumentasi. Enam soal lainnya baru, dan masing-masing memusatkan perhatian pada satu gagasan dari Bab 13 sampai Bab 15: koleksi dengan iteratornya sendiri, objek nilai imutabel yang aman dijadikan kunci *map*, pencacah bersama dengan *race* yang kita ukur lalu kita hilangkan, lini cuci produsen–konsumen, protokol permintaan–balasan mungil buatan kita sendiri, dan akhirnya sebuah proyek puncak (*capstone*) yang menggabungkan seluruh isi buku: sistem Profitina Laundry yang kecil tetapi lengkap, dengan kelas, antarmuka, pewarisan, koleksi, eksepsi, berkas, pola Model–View–Controller, dan API soket, ditulis dua kali, dalam Java dan dalam Python, beserta ujinya. Setiap soal menempuh jalan yang sudah Anda kenal dari bab-bab evaluasi sebelumnya: pernyataan soal, soal itu dengan bahasa kita sendiri, rancangan, penyelesaian Java, penyelesaian Python, keluaran yang sebenarnya, penjelasan baris demi baris, penyelesaian keliru yang menggoda beserta cara persisnya ia gagal, dan varian *uji diri* beserta jawabannya. Bab ini ditutup dengan peta seluruh buku, panduan tentang apa yang sebaiknya dipelajari berikutnya, dan sepatah kata penutup.

16.1 Titik Tolak Kita

Bab 13, 14, dan 15 membawa kotak perkakas berorientasi objek dari dua belas bab pertama keluar dari dunia satu *thread* dan satu mesin. Sebelum mulai, ada baiknya kita menyebut dengan tepat perkakas apa saja yang mereka tambahkan, sebab menyebut perkakas sudah separuh dari memilihnya. **Tabel 16.1** mendaftarkannya.

Tabel 16.1 Perkakas yang ditambahkan Bab 13 sampai Bab 15, dalam kedua bahasa. Setiap soal dalam bab ini memakai beberapa di antaranya.

Bab	Perkakas	Di Java	Di Python
13	menelusuri koleksi	`Iterator`, `Iterable`, gelung for-each	`__iter__`, `__next__`, generator dengan `yield`
13	mengubah koleksi sambil menelusurinya	`ConcurrentModificationException`, `Iterator.remove`, `removeIf`	`RuntimeError` untuk dict dan set, *comprehension*
13	mengurutkan objek	`Comparable`, `Comparator`	fungsi `key=`, `__lt__`
13	menyalin	salinan dangkal dan salinan dalam	`copy.copy`, `copy.deepcopy`
13	imutabilitas	*field* `final`, *record*, tampilan yang tak dapat diubah	*dataclass* beku, *tuple*
14	*thread*	`Thread`, `Runnable`, `start`, `join`, `interrupt`	`threading.Thread`, `join`
14	*race* dan penawarnya	`synchronized`, kunci, variabel atomik	`threading.Lock`, `with lock:`
14	menunggu suatu kondisi	`wait`, `notifyAll`, `BlockingQueue`	`threading.Condition`, `queue.Queue`
14	*deadlock* dan *livelock*	urutan penguncian, batas waktu	gagasan yang sama
15	soket	`ServerSocket`, `Socket`, *stream*	`socket.create_server`, `create_connection`, `makefile`
15	protokol	protokol *knock-knock*, satu *thread* per klien	sama

Tentu saja bagian lain buku ini tetap ada, dan soal puncak di akhir bab ini memakai hampir semuanya: kelas dan objek (Bab 3), larik, *list*, eksepsi, dan berkas (Bab 5), dokumentasi, uji, dan antarmuka (Bab 6), pewarisan dan polimorfisme (Bab 7), koleksi dan generik (Bab 9), antarmuka grafis dan peristiwa (Bab 10), serta pola Model–View–Controller (Bab 11).

Bab ini disusun dalam lima lapis.

1. **Sebuah metode** (Subbab 16.2): sepuluh langkah yang membawa tim kecil dari tugas satu halaman menjadi aplikasi klien–server yang teruji dan terdokumentasi.
2. **Tugas proyek, dinyatakan ulang** (Subbab 16.3), dengan tabel yang memetakan setiap tuntutannya ke satu soal yang dikerjakan.
3. **Delapan soal yang melaksanakan tugas itu** (Subbab 16.4 sampai 16.6), yaitu Soal 16.1 sampai 16.8, semuanya pada contoh *knock-knock*.
4. **Enam soal baru** (Subbab 16.7 sampai 16.10), yaitu Soal 16.9 sampai 16.14, diakhiri dengan proyek puncak.
5. **Subbab penutup**: perbandingan keempat belas soal, peta seluruh buku, arah belajar selanjutnya, Profitina dalam praktik, kesalahan yang sering terjadi, ringkasan dan kata penutup, soal latihan beserta penyelesaian lengkapnya.

Setiap soal disajikan dalam urutan yang sama, dengan label tebal yang mudah Anda temukan kembali: **Soal**, **Dengan bahasa kita sendiri**, **Rancangan**, **Penyelesaian Java**, **Penyelesaian Python**, **Baris demi baris**, **Penyelesaian keliru yang menggoda**, dan **Uji diri**. Seperti di Bab 8, program Java menguji dirinya sendiri dengan metode kecil `check` yang mencetak `PASS` beserta sebuah nama, dan program Python memakai `unittest` [1]. Karena itu keluaran yang dicetak sesudah setiap kode sekaligus merupakan laporan ujinya.

> **CARA MEMAKAI BAB INI.** Soal-soal di paruh pertama saling membangun: Soal 16.3 memakai kelas-kelas dari Soal 16.1 dan 16.2, dan seterusnya. Bacalah secara berurutan pada kesempatan pertama. Sebelum membaca setiap penyelesaian, tutuplah buku selama sepuluh menit dan buatlah sketsa rancangan Anda sendiri di atas kertas: kelas-kelasnya, pesan-pesannya, uji yang akan Anda tulis lebih dulu. Sesudah itu, bandingkan. Program tentang *thread* dan jaringan adalah jenis program yang jarak antara "saya membacanya dan paham" dan "saya sanggup menulisnya" paling jauh, jadi ketiklah sendiri setidaknya program-program yang lebih kecil lalu jalankan.

Sepatah kata tentang keluaran. Sebagian besar program dalam bab ini mencetak baris yang persis sama pada setiap kali dijalankan, karena setiap uji ditulis agar tidak bergantung pada untung-untungan. Beberapa program *mengukur* sesuatu, entah waktu entah banyaknya pembaruan yang hilang, dan angka-angka itu berubah dari satu kali jalan ke kali berikutnya dan dari satu mesin ke mesin lain. Teks selalu menyebutkannya, dan keluaran yang ditampilkan adalah keluaran sebenarnya dari satu kali jalan pada mesin yang membangun buku ini.

16.2 Cara Menghadapi Proyek Satu Sistem Utuh

Sampai sekarang, setiap soal dalam buku ini muat di satu kepala dan satu berkas. Aplikasi klien–server berjendela tidak demikian. Aplikasi semacam itu memiliki sedikitnya dua program yang berjalan bersamaan, sering di mesin yang berbeda; memiliki beberapa *thread* di dalam setiap program; memiliki protokol yang harus dipatuhi kedua program dengan tepat; dan biasanya dibangun oleh beberapa orang sekaligus. Kesulitan utamanya bukan lagi "bagaimana saya menulis metode ini?", melainkan "bagaimana kita memastikan semua bagian cocok satu sama lain ketika akhirnya digabungkan?" Subbab ini memberikan metode sepuluh langkah. Langkah-langkah ini adalah delapan langkah Subbab 8.2 yang telah tumbuh dewasa untuk sebuah tim dan sebuah jaringan.

> **DEFINISI.** *Kontrak* adalah janji tertulis pada batas antara dua bagian sistem: apa yang boleh diminta satu pihak, apa yang dijanjikan pihak lain untuk dijawab, dan apa yang terjadi ketika sesuatu berjalan salah. `interface` Java adalah salah satu jenis kontrak; protokol jaringan adalah jenis lain; format berkas adalah jenis ketiga.

Langkah 1 — Pilih topik yang intinya kecil. Proyek yang baik memiliki inti yang kecil dan ruang untuk tumbuh di tepiannya. "Obrolan dengan permainan" boleh saja, asalkan versi pertama permainannya adalah *tic-tac-toe*. Tulislah satu kalimat tentang apa yang dapat dilakukan pengguna dengan program yang sudah jadi, dan satu kalimat tentang apa yang *tidak* akan dilakukannya.

Langkah 2 — Tulis spesifikasi sebagai hal-hal yang dilakukan pengguna. "Pengguna menyambung, melihat daftar ruang, bergabung ke salah satunya, mengirim pesan, melihat pesan dari orang lain, lalu pergi." Setiap kalimat menjadi sebuah fitur dan, kelak, sebuah uji.

Langkah 3 — Gambarlah bagian-bagiannya dan beri setiap bagian seorang pemilik. Aplikasi klien–server memiliki lima bagian alami: server, logika klien, antarmuka grafis, penyimpanan (basis data atau berkas), serta pengujian dan dokumentasi yang mengikat semuanya. Dengan tiga orang, satu orang memiliki dua bagian. Memiliki berarti "saya membuatnya berjalan dan saya menjawab pertanyaan tentangnya", bukan "tak seorang pun boleh menyentuhnya".

Langkah 4 — Tulis kontrak sebelum kode. Untuk setiap panah di antara dua bagian, tulislah kontraknya: antarmuka Java atau kelas abstrak Python untuk bagian-bagian di dalam satu

program, protokol untuk bagian-bagian di dua sisi jaringan. Simpanlah kontrak-kontrak itu di repositori sejak hari pertama, walaupun belum ada yang mengimplementasikannya.

Langkah 5 — Rancang protokol sebagai diagram alir, lalu sebagai mesin keadaan. Diagram alir memperlihatkan percakapan bagi manusia; mesin keadaan, dengan tabel "dalam keadaan ini, pada masukan ini, balaslah ini dan pindahlah ke sana", memperlihatkannya bagi program. Yang kedua itulah yang Anda implementasikan.

Langkah 6 — Biarkan setiap pihak menulis uji untuk pihak lain. Orang yang *memakai* sebuah kontrak paling tahu apa yang harus dilakukannya, sehingga pemilik klien menulis uji untuk server dan pemilik server menulis uji untuk klien, masing-masing terhadap sebuah *stub*: tiruan mungil pihak lawan.

Langkah 7 — Bangun kerangka yang sudah bisa berjalan. Buatlah versi setipis mungkin dari seluruh sistem berjalan dari ujung ke ujung, yaitu satu pesan dari jendela melalui klien dan jaringan ke server lalu kembali, sebelum memperkaya bagian mana pun. Dengan begitu, masalah penggabungan muncul pada hari ketiga, bukan pada hari terakhir.

Langkah 8 — Tambahkan konkurensi dan penanganan kegagalan dengan sengaja. Putuskan objek mana yang dipakai bersama oleh beberapa *thread* dan bagaimana masing-masing dilindungi; putuskan apa yang dilakukan klien ketika server mati dan apa yang dilakukan server ketika klien menghilang. Ujilah keduanya.

Langkah 9 — Ukur, lalu dokumentasikan. Ukurlah angka-angka yang akan Anda janjikan (berapa klien, seberapa cepat), kemudian tulislah panduan pengguna, panduan instalasi, kriteria kinerja, dan tabel pemecahan masalah.

Langkah 10 — Tulis laporan sambil berjalan. Simpanlah catatan singkat tentang keputusan dan tentang siapa mengerjakan apa. Laporan di akhir pun tinggal soal menyunting, bukan soal mengingat-ingat.

Tabel 16.2 meringkas metode ini; setiap soal di Subbab 16.4 sampai 16.6 adalah satu atau dua langkahnya yang dilaksanakan.

Tabel 16.2 Metode sepuluh langkah untuk proyek tim, dan soal dalam bab ini tempat setiap langkah dilaksanakan.

Langkah	Yang Anda hasilkan	Soal dalam bab ini
1–3	tujuan satu kalimat, cerita pengguna, bagian beserta pemiliknya	16.1
4	kontrak: antarmuka dan protokol	16.1, 16.2
5	diagram alir, mesin keadaan, tabel keadaan	16.2
6	uji silang dengan *stub*	16.4
7	kerangka yang berjalan dari ujung ke ujung	16.3
8	keamanan *thread* dan penanganan kegagalan	16.5, 16.6, 16.7
9	pengukuran dan dokumentasi produk	16.8
10	laporan proyek	16.8

MENGAPA KONTRAK LEBIH DULU. Tiga orang yang menyepakati sebuah antarmuka pada hari Senin dapat bekerja sejajar sepanjang minggu, masing-masing dengan *stub* untuk bagian yang bukan miliknya. Tiga orang yang hanya bersepakat "secara lisan" baru mendapati pada hari Jumat bahwa salah seorang menomori lelucon mulai dari 1 dan yang lain mulai dari 0. Soal 16.1 memperlihatkan *bug* persis itu dan uji yang menangkapnya.

16.3 Tugas Proyek, Dinyatakan Ulang

Delapan soal dalam tiga subbab berikut berasal dari satu tugas proyek. Berikut tugas itu selengkapnya, seperti yang diterima sebuah tim, dalam bahasa aslinya.

> **SOAL PROYEK.** Based on everything in this book, build a client–server application with a graphical user interface, in which the client and the server communicate through sockets. Examples: a game with a chat, a small "social network", a collaboration tool, an interactive portal, a start-up idea, or anything else the team finds exciting. Work in a team of at most three people. (a) Decide the topic and title of the project immediately. (b) Divide the work: each member writes the specification and the job description of their own part; the natural parts are the server, the client, the database and its connectivity, the graphical interface, and testing and documentation. (c) Describe the specification clearly and document every interface, where "interface" is a general term and need not be a Java `interface`. (d) Describe every protocol and flow chart clearly, for example a knock-knock protocol between server and client. (e) Think about test cases: if member A uses the functions of member B through an interface, A designs test cases for B, and B designs test cases for A. (f) Describe the design, from specification to design to implementation, and the product documentation: a user-level guide, performance criteria, installation, and troubleshooting. (g) Develop and use test plans, for example with stubs, multithreaded tests and distributed tests. (h) Take care that the client knows when the server is down, that the server knows when a client is down, and that connection errors on both sides are handled as exceptions. (i) Write a report on what was done, the output, the answers to these points, pseudocode and anything else that matters, including each member's job description and share of the work.
>
> *Dengan kata lain:* dengan bekal seluruh isi buku ini, bangunlah aplikasi klien–server berantarmuka grafis yang klien dan server-nya berkomunikasi melalui soket, misalnya permainan dengan fitur obrolan, "jejaring sosial" kecil, alat kolaborasi, portal interaktif, gagasan usaha rintisan, atau apa pun yang menurut tim menarik. Tim terdiri atas paling banyak tiga orang. (a) Tetapkan topik dan judul proyek sesegera mungkin. (b) Bagilah pekerjaan: setiap anggota menulis spesifikasi dan uraian tugas bagiannya sendiri; bagian alaminya adalah server, klien, basis data beserta konektivitasnya, antarmuka grafis, serta pengujian dan dokumentasi. (c) Uraikan spesifikasi dengan jelas dan dokumentasikan setiap antarmuka, dalam arti umum, tidak harus berupa `interface` Java. (d) Uraikan setiap protokol dan diagram alir dengan jelas, misalnya protokol *knock-knock* antara server dan klien. (e) Pikirkan kasus uji: jika anggota A memakai fungsi anggota B melalui sebuah antarmuka, A merancang kasus uji untuk B, dan B merancang kasus uji untuk A. (f) Uraikan rancangannya, dari spesifikasi ke rancangan ke implementasi, serta dokumentasi produknya: panduan tingkat pengguna, kriteria kinerja, instalasi, dan pemecahan masalah. (g) Susun dan pakailah rencana uji, misalnya dengan *stub*, uji *multithread*, dan uji terdistribusi. (h) Pastikan klien mengetahui ketika server mati, server mengetahui ketika klien mati, dan galat koneksi di kedua sisi ditangani sebagai eksepsi. (i) Tulislah laporan tentang apa yang telah dikerjakan, keluarannya, jawaban atas butir-butir ini, pseudokode, dan hal lain yang penting, termasuk uraian tugas dan bagian kerja setiap anggota.

Tugas ini tidak menyebut topik, sehingga kita memilih topik yang cukup kecil untuk diselesaikan tuntas di atas kertas namun tetap memiliki setiap bagian yang diminta: **Knock-Knock Club**, sebuah server yang menceritakan lelucon *knock-knock* kepada berapa pun klien sekaligus, klien konsol, klien berjendela, dan sebuah berkas lelucon yang berperan sebagai basis data. Percakapan *knock-knock* adalah contoh protokol yang disarankan tugas itu sendiri, dan Bab 15 telah memperkenalkannya; di sini kita membangunnya dengan sungguh-sungguh. Tim sungguhan akan

memilih topik yang lebih kaya, entah obrolan, permainan, atau sistem binatu pada Soal 16.14, dan setiap penyelesaian di bawah ini tetap berlaku tanpa perubahan.

Tabel 16.3 memetakan tuntutan tugas itu ke soal-soal yang memenuhinya.

Tabel 16.3 Pembagian tuntutan tugas proyek ke dalam Soal 16.1 sampai 16.8.

Tuntutan tugas	Soal	Yang kita hasilkan
(a), (b) topik, pembagian kerja	16.1	topik, cerita pengguna, tabel bagian, pemilik, dan kontrak
(c) spesifikasi dan antarmuka	16.1	`JokeStore`, dua implementasi, satu uji kontrak
(d) protokol dan diagram alir	16.2	diagram alir, mesin keadaan, `KnockKnockProtocol`
server dan klien itu sendiri	16.3	server banyak-klien dan klien konsol
(e) A menguji B, B menguji A	16.4	uji silang dengan *stub*, tanpa jaringan
(g) uji *multithread* dan terdistribusi	16.5	dua puluh klien serentak terhadap server sungguhan
(h) server mati, klien mati	16.6	empat skenario kegagalan, masing-masing dideteksi dan dilaporkan
antarmuka grafis	16.7	jendela Swing dan jendela Tkinter, MVC, uji tanpa layar
(f), (i) dokumentasi, kinerja, laporan	16.8	kriteria terukur, panduan pengguna, pemecahan masalah, kerangka laporan

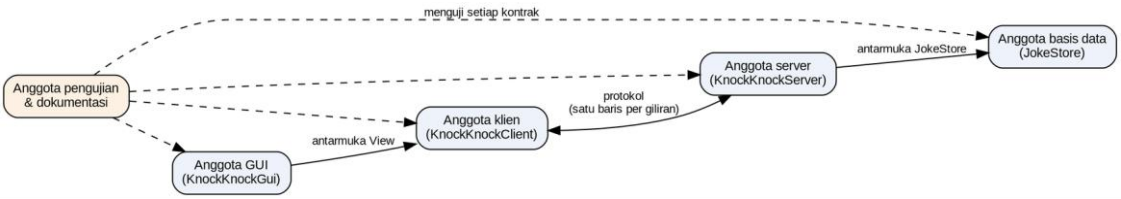
16.4 Soal-Soal tentang Perencanaan dan Rancangan Protokol

16.4.1 Soal 16.1 — Topik, pembagian kerja, dan kontrak pertama

> **SOAL.** Choose the topic and title of the project, divide the work among the members, and write the specification and job description of each part. Document the interface between the storage part and the rest of the system so precisely that another member can implement it, and show with a test that two different implementations keep the same promise.
>
> *Dengan kata lain:* pilihlah topik dan judul proyek, bagilah pekerjaan di antara para anggota, dan tulislah spesifikasi serta uraian tugas setiap bagian. Dokumentasikan antarmuka antara bagian penyimpanan dan bagian sistem lainnya sedemikian tepat sehingga anggota lain dapat mengimplementasikannya, lalu tunjukkan dengan sebuah uji bahwa dua implementasi yang berbeda menepati janji yang sama.

Dengan bahasa kita sendiri. Sebelum ada yang menulis server, tim harus menyepakati tiga hal: untuk apa program itu, siapa membangun bagian mana, dan apa persisnya yang dijanjikan setiap bagian kepada bagian lain. Hal terakhir itulah yang sering dilewatkan tim, padahal justru itu yang paling penting, sebab hanya itulah yang memungkinkan orang bekerja pada waktu yang sama. Kita akan menulis janji pertama, antara "basis data" dan semua bagian lain, sebagai sebuah antarmuka, dan kita akan menulis uji yang harus dilalui setiap implementasi. Kemudian kita akan menyaksikan uji itu menangkap sebuah salah paham yang realistis.

Rancangan. Langkah 1 sampai 3 dari metode kita menghasilkan hal-hal berikut. *Tujuan:* "Pengguna menyambung ke Knock-Knock Club, mendengar sebuah lelucon, menjawab di tempat yang tepat, dan boleh meminta lelucon lain." *Bukan tujuan:* akun, skor, atau lelucon karangan pengguna. *Cerita pengguna:* menyambung; mendengar "Knock! Knock!"; menjawab "Who's there?"; mendengar petunjuk; menjawab "*petunjuk* who?"; mendengar kalimat pamungkas dan "Want another? (y/n)"; menjawab `y` untuk lelucon lain atau `n` untuk pergi; melihat pesan yang jelas bila server tidak ada. **Gambar 16.1** memperlihatkan bagian-bagian, pemiliknya, dan kontrak di antara mereka, sedangkan **Tabel 16.4** adalah spesifikasi dan uraian tugas setiap bagian, satu baris per peran anggota.



Gambar 16.1 Lima bagian Knock-Knock Club dan kontrak di antara mereka. Panah utuh berarti "memakai"; panah putus-putus menunjukkan bahwa anggota pengujian menguji setiap kontrak.

Tabel 16.4 Spesifikasi dan uraian tugas setiap bagian Knock-Knock Club. Dengan tiga anggota, anggota 2 dan 3 masing-masing memiliki dua bagian.

Bagian (pemilik)	Spesifikasi	Kontrak yang ditawarkan	Kontrak yang dipakai
Basis data (anggota 1)	menyimpan lelucon; memuatnya dari `jokes.txt`	`JokeStore`: `size()`, `get(i)`	tidak ada
Server (anggota 2)	menerima banyak klien, satu percakapan per klien	protokol *knock-knock*	`JokeStore`
Klien (anggota 2)	menjalankan satu percakapan untuk satu pengguna	`converse(reader, writer, user)`	protokol
GUI (anggota 3)	jendela yang menampilkan percakapan, tidak pernah membeku	`View`: `showLine`, `setStatus`	klien
Pengujian dan dokumentasi (anggota 3)	uji kontrak, uji kegagalan, panduan pengguna	tidak ada	semuanya

Kontrak penyimpanan adalah yang terkecil, jadi kita menulisnya lebih dulu. Sebuah penyimpan berisi paling sedikit satu lelucon, lelucon-leluconnya bernomor 0 sampai `size() - 1`, dan permintaan untuk nomor lain mana pun melempar `IndexOutOfBoundsException` (Python: `IndexError`). Menuliskan aturan penomoran itu tampak berlebihan. Justru di situlah pokok persoalannya.

Penyelesaian Java. **Kode 16.1** adalah kontrak itu sendiri. Isinya adalah *record* `Joke` (nilai kecil yang imutabel, Bab 13), dua metode abstrak beserta dokumentasinya, dan *factory* statis `of` yang membangun penyimpan di memori: *stub* yang dapat dipakai setiap anggota lain sejak jam pertama, sebelum basis data sungguhan ada.

Kode 16.1 `JokeStore.java`: kontrak penyimpanan, dengan *record* `Joke` dan *stub* di memori yang dibangun oleh `JokeStore.of`.

```
// JokeStore.java - kontrak antara anggota basis data dan semua anggota lain.
import java.util.List;

public interface JokeStore {

    /** Satu lelucon knock-knock: petunjuk ("Lettuce") dan jawabannya ("Lettuce in, ...").
*/
    record Joke(String clue, String answer) {
    }

    /** @return banyaknya lelucon dalam penyimpan; selalu paling sedikit 1. */
    int size();

    /**
     * Mengembalikan lelucon bernomor index, dihitung mulai dari 0.
     *
     * @throws IndexOutOfBoundsException jika index tidak berada dalam 0 .. size()-1
     */
    Joke get(int index);

    /** Penyimpan di memori: stub yang dapat dipakai setiap anggota lain sejak hari
pertama. */
    static JokeStore of(Joke... jokes) {
        if (jokes.length == 0) {
            throw new IllegalArgumentException("a store needs at least one joke");
        }
        List<Joke> copy = List.of(jokes);               // salinan yang tidak dapat diubah
        return new JokeStore() {
            @Override
            public int size() {
                return copy.size();
            }

            @Override
            public Joke get(int index) {
                return copy.get(index);                 // List.get memeriksa indeksnya
            }
        };
    }

    /** Tiga lelucon yang dipakai contoh-contoh dalam bab ini. */
    static JokeStore classics() {
        return of(new Joke("Lettuce", "Lettuce in, it's cold out here!"),
                  new Joke("Harry", "Harry up, the laundry is ready!"),
                  new Joke("Olive", "Olive the clean shirts, thank you!"));
    }
}
```

Kode 16.2 adalah hasil kerja sungguhan anggota basis data: penyimpan yang membaca berkas `jokes.txt`, penyimpan kedua yang mengandung kekeliruan realistis, dan sebuah *uji kontrak*, yaitu satu metode yang memeriksa janji itu dan dapat dijalankan terhadap implementasi apa pun. Berkas contohnya memuat sebuah komentar, sebuah baris kosong, dan empat lelucon:

```
# clue;answer
Lettuce;Lettuce in, it's cold out here!
Harry;Harry up, the laundry is ready!

Olive;Olive the clean shirts, thank you!
Iva;Iva sore hand from knocking!
```

Kode 16.2 `JokeStores.java`: `JokeStore` yang berbasis berkas, penyimpan bernomor mulai 1 yang melanggar kontrak, dan satu uji kontrak yang dijalankan terhadap ketiga penyimpan.

```java
// JokeStores.java - Soal 16.1: dua implementasi JokeStore dan satu uji kontrak.
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.util.ArrayList;
import java.util.List;

public class JokeStores {

    /** Penyimpan sungguhan milik anggota basis data: membaca baris "clue;answer" dari
berkas teks. */
    static final class FileJokeStore implements JokeStore {
        private final List<Joke> jokes;

        FileJokeStore(Path file) throws IOException {
            List<Joke> loaded = new ArrayList<>();
            int lineNo = 0;
            for (String line : Files.readAllLines(file)) {
                lineNo++;
                if (line.isBlank() || line.startsWith("#")) {
                    continue;                                   // baris komentar atau
kosong
                }
                String[] parts = line.split(";", 2);            // paling banyak dua potong
                if (parts.length != 2 || parts[0].isBlank()) {
                    throw new IOException(file + " line " + lineNo + ": expected
clue;answer");
                }
                loaded.add(new Joke(parts[0].trim(), parts[1].trim()));
            }
            if (loaded.isEmpty()) {
                throw new IOException(file + " contains no jokes");
            }
            this.jokes = List.copyOf(loaded);                   // bekukan list-nya
        }

        @Override
        public int size() {
            return jokes.size();
        }

        @Override
        public Joke get(int index) {
            return jokes.get(index);
        }
    }
```

Kode 16.2 (Lanjutan) *JokeStores.java: JokeStore yang berbasis berkas, penyimpan bernomor mulai 1 yang melanggar kontrak, dan satu uji kontrak yang dijalankan terhadap ketiga penyimpan.*

```
    /** Menggoda tetapi keliru: anggota ini mengira "lelucon nomor 1" adalah lelucon
pertama. */
    static final class OneBasedFileJokeStore implements JokeStore {
        private final FileJokeStore inner;

        OneBasedFileJokeStore(Path file) throws IOException {
            inner = new FileJokeStore(file);
        }

        @Override
        public int size() {
            return inner.size();
        }

        @Override
        public Joke get(int index) {
            return inner.get(index - 1);                        // selisih satu
        }
    }

    /** Kontrak, ditulis SEKALI, dijalankan terhadap setiap implementasi. Mengembalikan
kegagalannya. */
    static List<String> contract(JokeStore s) {
        List<String> failures = new ArrayList<>();
        if (s.size() < 1) {
            failures.add("size() must be at least 1");
        }
        try {
            if (s.get(0) == null || s.get(0).clue().isBlank()) {
                failures.add("get(0) must be a real joke");
            }
        } catch (RuntimeException e) {
            failures.add("get(0) threw " + e.getClass().getSimpleName());
        }
        for (int bad : new int[] {-1, s.size()}) {
            try {
                s.get(bad);
                failures.add("get(" + bad + ") must throw IndexOutOfBoundsException");
            } catch (IndexOutOfBoundsException expected) {
                // persis seperti yang dijanjikan kontrak
            }
        }
        return failures;
    }

    private static int passed = 0;
```

Kode 16.2 (Lanjutan) *JokeStores.java: JokeStore yang berbasis berkas, penyimpan bernomor mulai 1 yang melanggar kontrak, dan satu uji kontrak yang dijalankan terhadap ketiga penyimpan.*

```
    private static void check(String name, Object expected, Object actual) {
        if (!java.util.Objects.equals(expected, actual)) {
            throw new AssertionError(name + ": expected " + expected + " but got " +
actual);
        }
        passed++;
        System.out.println("PASS " + name);
    }

    public static void main(String[] args) throws IOException {
        Path file = Path.of("jokes.txt");
        JokeStore stub = JokeStore.classics();
        JokeStore real = new FileJokeStore(file);
        JokeStore wrong = new OneBasedFileJokeStore(file);

        System.out.println("stub : " + stub.size() + " jokes, first = " +
stub.get(0).clue());
        System.out.println("file : " + real.size() + " jokes, first = " +
real.get(0).clue());
        System.out.println("wrong: " + contract(wrong));

        check("stub keeps the contract", List.of(), contract(stub));
        check("file store keeps the contract", List.of(), contract(real));
        check("one-based store is caught", 2, contract(wrong).size());
        try {
            new FileJokeStore(Path.of("no_such_file.txt"));
            throw new AssertionError("no exception");
        } catch (IOException e) {
            check("missing file", "NoSuchFileException", e.getClass().getSimpleName());
        }
        System.out.println("All " + passed + " checks passed.");
    }
}
```

```
stub : 3 jokes, first = Lettuce
file : 4 jokes, first = Lettuce
wrong: [get(0) threw ArrayIndexOutOfBoundsException, get(4) must throw
IndexOutOfBoundsException]
PASS stub keeps the contract
PASS file store keeps the contract
PASS one-based store is caught
PASS missing file
All 4 checks passed.
```

Penyelesaian Python. Python tidak memiliki kata kunci `interface`, sehingga kontraknya berupa kelas dasar abstrak (Bab 6). **Kode 16.3** adalah `joke_store.py`, modul yang diimpor setiap berkas Python lainnya, dan **Kode 16.4** memuat penyimpan berkas, penyimpan yang rusak, dan kontrak yang sama.

Kode 16.3 `joke_store.py`: kontrak penyimpanan sebagai kelas dasar abstrak, *dataclass* beku `Joke`, dan *stub* di memori `ListJokeStore`.

```
# joke_store.py - kontrak antara anggota basis data dan semua anggota lain.
from abc import ABC, abstractmethod
from dataclasses import dataclass


@dataclass(frozen=True)
class Joke:
    """Satu lelucon knock-knock: petunjuk ("Lettuce") dan jawabannya."""
    clue: str
    answer: str


class JokeStore(ABC):
    """Sumber lelucon, bernomor mulai dari 0."""

    @abstractmethod
    def size(self):
        """Banyaknya lelucon dalam penyimpan; selalu paling sedikit 1."""

    @abstractmethod
    def get(self, index):
        """Lelucon bernomor index; memunculkan IndexError kecuali 0 <= index < size()."""


class ListJokeStore(JokeStore):
    """Penyimpan di memori: stub yang dapat dipakai setiap anggota lain sejak hari
pertama."""

    def __init__(self, *jokes):
        if not jokes:
            raise ValueError("a store needs at least one joke")
        self._jokes = tuple(jokes)                  # salinan yang tidak dapat diubah

    def size(self):
        return len(self._jokes)

    def get(self, index):
        if not 0 <= index < len(self._jokes):      # tuple akan menerima -1!
            raise IndexError(f"joke {index} does not exist")
        return self._jokes[index]


def classics():
    """Tiga lelucon yang dipakai contoh-contoh dalam bab ini."""
    return ListJokeStore(Joke("Lettuce", "Lettuce in, it's cold out here!"),
                         Joke("Harry", "Harry up, the laundry is ready!"),
                         Joke("Olive", "Olive the clean shirts, thank you!"))
```

Kode 16.4 `joke_stores.py`: penyimpan berkas, penyimpan bernomor mulai 1, dan kontraknya, dengan uji `unittest`.

```
# joke_stores.py - Soal 16.1: dua implementasi JokeStore dan satu uji kontrak.
import sys
import unittest
from joke_store import Joke, JokeStore, ListJokeStore, classics


class FileJokeStore(JokeStore):
    """Penyimpan sungguhan milik anggota basis data: membaca baris "clue;answer" dari
berkas teks."""

    def __init__(self, file_name):
        loaded = []
        with open(file_name, encoding="utf-8") as f:
            for line_no, line in enumerate(f, start=1):
                line = line.strip()
                if not line or line.startswith("#"):
                    continue                            # baris komentar atau kosong
                parts = line.split(";", 1)              # paling banyak dua potong
                if len(parts) != 2 or not parts[0].strip():
                    raise OSError(f"{file_name} line {line_no}: expected clue;answer")
                loaded.append(Joke(parts[0].strip(), parts[1].strip()))
        if not loaded:
            raise OSError(f"{file_name} contains no jokes")
        self._inner = ListJokeStore(*loaded)

    def size(self):
        return self._inner.size()

    def get(self, index):
        return self._inner.get(index)


class OneBasedFileJokeStore(FileJokeStore):
    """Menggoda tetapi keliru: anggota ini mengira "lelucon nomor 1" adalah lelucon
pertama."""

    def get(self, index):
        return super().get(index - 1)                   # selisih satu
```

Kode 16.4 (Lanjutan) *joke_stores.py: penyimpan berkas, penyimpan bernomor mulai 1, dan kontraknya, dengan uji unittest.*

```
def contract(store):
    """Kontrak, ditulis SEKALI, dijalankan terhadap setiap implementasi."""
    failures = []
    if store.size() < 1:
        failures.append("size() must be at least 1")
    try:
        if not store.get(0).clue.strip():
            failures.append("get(0) must be a real joke")
    except Exception as e:
        failures.append(f"get(0) raised {type(e).__name__}")
    for bad in (-1, store.size()):
        try:
            store.get(bad)
            failures.append(f"get({bad}) must raise IndexError")
        except IndexError:
            pass                                        # persis seperti yang dijanjikan
kontrak
    return failures


class TestStores(unittest.TestCase):
    def test_stub_keeps_the_contract(self):
        self.assertEqual([], contract(classics()))

    def test_file_store_keeps_the_contract(self):
        self.assertEqual([], contract(FileJokeStore("jokes.txt")))

    def test_one_based_store_is_caught(self):
        self.assertEqual(2, len(contract(OneBasedFileJokeStore("jokes.txt"))))

    def test_missing_file(self):
        with self.assertRaises(FileNotFoundError):
            FileJokeStore("no_such_file.txt")


if __name__ == "__main__":
    stub, real = classics(), FileJokeStore("jokes.txt")
    print("stub :", stub.size(), "jokes, first =", stub.get(0).clue)
    print("file :", real.size(), "jokes, first =", real.get(0).clue)
    print("wrong:", contract(OneBasedFileJokeStore("jokes.txt")))
    sys.stdout.flush()                                  # cetak sebelum laporan uji
    unittest.main(verbosity=2)
```

```
stub : 3 jokes, first = Lettuce
file : 4 jokes, first = Lettuce
wrong: ['get(0) raised IndexError', 'get(4) must raise IndexError']
test_file_store_keeps_the_contract (__main__.TestStores.test_file_store_keeps_the_contract)
... ok
test_missing_file (__main__.TestStores.test_missing_file) ... ok
test_one_based_store_is_caught (__main__.TestStores.test_one_based_store_is_caught) ... ok
test_stub_keeps_the_contract (__main__.TestStores.test_stub_keeps_the_contract) ... ok

----------------------------------------------------------------------
Ran 4 tests in 0.000s

OK
```

Baris demi baris. Pada **Kode 16.1**, `record Joke(String clue, String answer)` mendeklarasikan sebuah kelas dengan dua *field* privat yang final, sebuah konstruktor, dua pengakses `clue()` dan `answer()`, serta `equals`, `hashCode`, dan `toString`, semuanya ditulis oleh kompilator. *Record* yang bersarang di dalam antarmuka otomatis bersifat `static` dan `public`, sehingga berkas lain menuliskan `JokeStore.Joke`. Javadoc dari `get` menyatakan aturan

penomoran dan eksepsinya; komentar itulah *yang menjadi* spesifikasi. `static JokeStore of(Joke... jokes)` menerima lelucon berapa pun banyaknya (`...` membuat parameter *varargs*), menyalinnya dengan `List.of` yang menghasilkan *list* tak dapat diubah, lalu mengembalikan sebuah *kelas anonim*: kelas tanpa nama yang ditulis dan diinstansiasi dalam satu ekspresi dan mengimplementasikan kedua metode. `copy.get(index)` sudah melempar `IndexOutOfBoundsException` untuk indeks yang salah, sehingga *stub* ini menepati kontrak tanpa usaha tambahan.

Pada **Kode 16.2**, `FileJokeStore` membaca setiap baris dengan `Files.readAllLines`, melewati komentar dan baris kosong persis seperti pengurai Soal 8.16, lalu memecah setiap baris dengan `split(";", 2)`. Batas 2 itu penting: jawaban yang mengandung titik koma tetap utuh. Baris yang salah bentuk atau berkas yang kosong dilaporkan sebagai `IOException` beserta nomor barisnya, sebab berkas lelucon yang rusak adalah masalah lingkungan, bukan masalah program. `List.copyOf(loaded)` membekukan *list*-nya, sehingga tak seorang pun dapat menambahkan lelucon ke penyimpan setelah penyimpan itu dibangun. Metode `contract` memeriksa empat janji: paling sedikit satu lelucon, lelucon sungguhan pada indeks 0, dan eksepsi untuk indeks `-1` serta `size()`. Metode itu mengumpulkan kegagalan dalam sebuah *list* alih-alih berhenti pada kegagalan pertama, sehingga laporannya memberi tahu pengimplementasi semua kesalahan sekaligus. Keluarannya memperlihatkan kedua penyimpan yang benar menepati kontrak dan penyimpan yang keliru gagal dua kali.

Di Python, `@dataclass(frozen=True)` menuliskan `__init__`, `__eq__`, `__hash__`, dan `__repr__` untuk `Joke` serta melarang penugasan ke *field*-nya. `ListJokeStore.get` memuat pemeriksaan yang tidak diperlukan di Java: *tuple menerima* indeks negatif (`t[-1]` adalah elemen terakhir), sehingga tanpa pemeriksaan eksplisit `0 <= index < len(...)` *stub* itu sendiri diam-diam melanggar kontrak untuk `-1`. Inilah contoh kecil yang nyata tentang kontrak yang lebih ketat daripada bahasanya, dan tentang uji kontrak yang layak ditulis bahkan untuk sebuah *stub*. `OneBasedFileJokeStore` mewarisi `FileJokeStore` dan hanya meng-*override* `get`, sehingga versi yang rusak tetap sependek kekeliruannya.

Penyelesaian keliru yang menggoda. `OneBasedFileJokeStore` adalah kekeliruan anggota yang menganggap "lelucon nomor 1" sebagai lelucon pertama. Tidak ada apa pun dalam bahasa yang mencegahnya: kode itu terkompilasi, mengembalikan lelucon sungguhan untuk sebagian besar indeks, dan uji manual sekilas ("ambilkan lelucon 1") bahkan tampak benar. Uji kontrak langsung menangkapnya, dan dengan tepat: `get(0)` melempar eksepsi, karena meminta indeks −1 dari *list* di dalamnya, dan `get(4)` *tidak* melempar eksepsi, karena diam-diam mengembalikan lelucon terakhir. Anggota server, yang menulis `get(current)` dengan `current` bermula dari 0, akan mengalami *crash* pada koneksi yang paling pertama. Kekeliruan yang lebih dalam adalah bersepakat "secara lisan". Kontrak tertulis beserta ujinya mengubah bencana penggabungan di Jumat malam menjadi satu pemeriksaan yang gagal pada hari Senin.

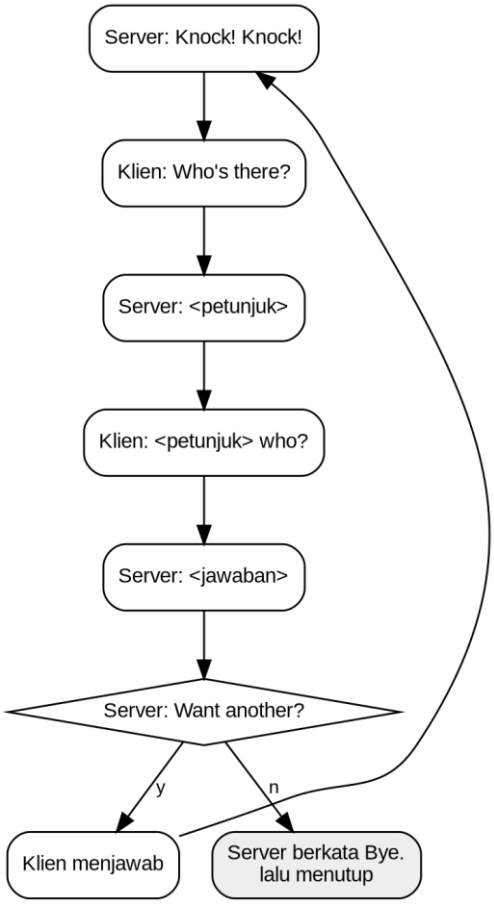
> **UJI DIRI.** Seorang anggota keempat menawarkan `RandomJokeStore`, yang `get(i)`-nya mengabaikan `i` dan mengembalikan lelucon acak. Lolos atau tidakkah ia dari uji kontrak? Seharusnya lolos atau tidak?
>
> *Jawaban.* Tidak lolos: `get(-1)` dan `get(size())` mengembalikan lelucon alih-alih melempar eksepsi, sehingga kontrak melaporkan dua kegagalan. Memang seharusnya gagal, karena server mengandalkan arti `get(i)` sebagai "lelucon nomor `i`" untuk menceritakan lelucon satu demi satu. Jika lelucon acak diinginkan, tempat yang tepat untuk itu adalah pilihan `i` oleh server, bukan penyimpannya.

16.4.2 Soal 16.2 — Protokol sebagai diagram alir dan mesin keadaan

> **SOAL.** Describe the protocol between the server and the client clearly, with a flow chart. The knock-knock conversation goes: server "Knock, knock"; client "Who's there?"; server a clue; client "*clue* who?"; server the punch line; server "Want more?"; if the client answers yes, the server knocks again; if no, the server terminates. Implement the server's side of this protocol as a class that can be tested without any network.
>
> *Dengan kata lain:* uraikan protokol antara server dan klien dengan jelas, menggunakan diagram alir. Percakapan *knock-knock* berjalan begini: server "Knock, knock"; klien "Who's there?"; server menyebut petunjuk; klien "*petunjuk* who?"; server menyebut kalimat pamungkas; server bertanya "Want more?"; jika klien menjawab ya, server mengetuk lagi; jika tidak, server berhenti. Implementasikan sisi server dari protokol ini sebagai kelas yang dapat diuji tanpa jaringan.

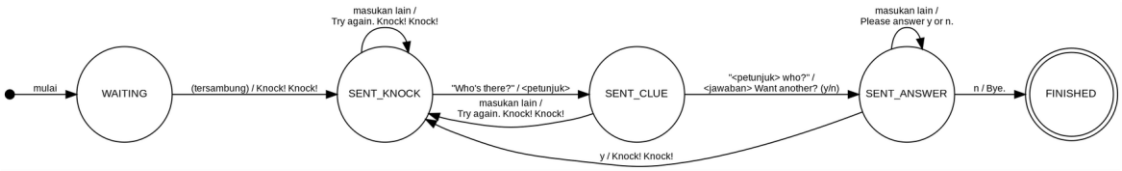
Dengan bahasa kita sendiri. Protokol adalah percakapan yang berkaidah. Manusia paling mudah memahaminya sebagai diagram alir; program memerlukannya sebagai *mesin keadaan*: pada setiap saat server berada dalam salah satu dari beberapa keadaan, dan setiap baris dari klien, bersama keadaan saat itu, menentukan balasan dan keadaan berikutnya. Kita akan menggambar keduanya, mengubah mesin keadaan itu menjadi tabel, dan mengimplementasikan tabel itu sebagai kelas dengan satu metode, `processInput`, yang menerima baris klien dan mengembalikan baris server. Karena kelas itu tidak pernah menyentuh soket, ia dapat diuji dengan *string* biasa.

Rancangan. **Gambar 16.2** adalah diagram alir dari tugas proyek. Gambar itu bagus untuk jalur mulus, tetapi meninggalkan setiap pertanyaan yang harus dijawab sebuah program: bagaimana jika klien berkata "Who is it?" alih-alih "Who's there?", atau mengembalikan petunjuk yang salah, atau menjawab "mungkin"?



Gambar 16.2 Protokol *knock-knock* sebagai diagram alir: percakapan bagi manusia, hanya dengan jawaban-jawaban yang diharapkan.

Gambar 16.3 adalah protokol yang sama sebagai mesin keadaan, dengan setiap masukan tertangani. Setiap panah diberi label "masukan / keluaran": misalnya, dalam keadaan `SENT_KNOCK`, masukan "Who's there?" menghasilkan keluaran *petunjuk* dan membawa ke keadaan `SENT_CLUE`. Aturan untuk baris yang salah adalah keputusan rancangan kita: server dengan sopan mengulang lelucon yang sedang berjalan.



Gambar 16.3 Protokol yang sama sebagai mesin keadaan. Setiap keadaan memiliki panah untuk setiap masukan yang mungkin, termasuk "masukan lain".

> **DEFINISI.** *Mesin keadaan hingga* (*finite state machine*) adalah sekumpulan keadaan, sebuah keadaan awal, dan sebuah *fungsi transisi* yang memetakan pasangan (keadaan, masukan) ke pasangan (keluaran, keadaan berikutnya). Inilah cara baku untuk menspesifikasikan protokol, karena cara ini memaksa adanya jawaban untuk setiap masukan dalam setiap keadaan.

Ditulis sebagai fungsi, mesin itu adalah

$$\delta(s, x) = (y, s') \tag{16.1}$$

dengan s keadaan saat ini, x baris dari klien, y balasan server, dan s' keadaan berikutnya. Persamaan (16.1) persis merupakan tanda tangan `processInput`: argumennya adalah x, nilai

kembaliannya adalah y, dan metode itu memperbarui *field* `state` dari s menjadi s'. **Tabel 16.5** mendaftarkan setiap transisi; tabel itulah spesifikasi yang harus dicocokkan implementasi baris demi baris.

Tabel 16.5 Tabel transisi protokol *knock-knock*. Setiap keadaan memiliki baris untuk "masukan lain", yang tidak tampak pada diagram alir.

Keadaan s	Baris klien x	Balasan server y	Keadaan berikutnya s'
`WAITING`	(tidak ada: klien baru saja menyambung)	`Knock! Knock!`	`SENT_KNOCK`
`SENT_KNOCK`	`Who's there?` (huruf besar-kecil bebas)	petunjuknya	`SENT_CLUE`
`SENT_KNOCK`	masukan lain	`You're supposed to say "Who's there?"! Try again. Knock! Knock!`	`SENT_KNOCK`
`SENT_CLUE`	*petunjuk* `who?` (huruf besar-kecil bebas)	jawabannya dan `Want another? (y/n)`	`SENT_ANSWER`
`SENT_CLUE`	masukan lain	`You're supposed to say "`*petunjuk*`who?"! Try again. Knock! Knock!`	`SENT_KNOCK`
`SENT_ANSWER`	`y`	`Knock! Knock!` (lelucon berikutnya)	`SENT_KNOCK`
`SENT_ANSWER`	`n`	`Bye.`	`FINISHED`
`SENT_ANSWER`	masukan lain	`Please answer y or n.`	`SENT_ANSWER`

Satu keputusan lagi termasuk dalam protokol: *pembingkaian* (*framing*), yaitu aturan yang memberi tahu penerima di mana sebuah pesan berakhir. Aturan kita adalah yang paling sederhana: setiap pesan tepat satu baris teks yang diakhiri karakter baris baru, dan server selalu menjawab satu baris untuk setiap baris yang diterimanya. Itulah sebabnya kalimat pamungkas dan "Want another? (y/n)" berjalan bersama dalam satu baris, walaupun diagram alir menggambarnya sebagai dua kotak. Dengan satu baris per giliran, klien tidak mungkin bingung tentang giliran siapa sekarang.

Penyelesaian Java. **Kode 16.5** mengimplementasikan **Tabel 16.5** dengan sebuah `enum` untuk keadaan-keadaannya dan sebuah `switch` pada keadaan saat ini.

Kode 16.5 `KnockKnockProtocol.java`: tabel transisi sebagai kelas, versi keliru tanpa keadaan, dan metode bantu `play` yang menjalankan percakapan tanpa jaringan.

```java
// KnockKnockProtocol.java - Soal 16.2: diagram alir knock-knock sebagai mesin keadaan.
import java.util.ArrayList;
import java.util.List;

public class KnockKnockProtocol {

    /** Empat kotak diagram alir tempat server menunggu, ditambah akhir. */
    enum State { WAITING, SENT_KNOCK, SENT_CLUE, SENT_ANSWER, FINISHED }

    static final String KNOCK = "Knock! Knock!";
    static final String ANOTHER = " Want another? (y/n)";
    static final String BYE = "Bye.";

    private final JokeStore jokes;
    private State state = State.WAITING;
    private int current = 0;                        // indeks lelucon yang sedang
diceritakan

    public KnockKnockProtocol(JokeStore jokes) {
        this.jokes = jokes;
    }

    public boolean finished() {
        return state == State.FINISHED;
    }

    /**
     * Berdasarkan ucapan klien barusan (null sebelum baris pertama), mengembalikan
     * ucapan server berikutnya. Satu panggilan adalah satu panah diagram alir.
     */
    public String processInput(String input) {
        JokeStore.Joke joke = jokes.get(current);
        switch (state) {
            case WAITING:
                state = State.SENT_KNOCK;
                return KNOCK;
            case SENT_KNOCK:
                if ("Who's there?".equalsIgnoreCase(input)) {
                    state = State.SENT_CLUE;
                    return joke.clue();
                }
                return "You're supposed to say \"Who's there?\"! Try again. " + KNOCK;
            case SENT_CLUE:
                if ((joke.clue() + " who?").equalsIgnoreCase(input)) {
                    state = State.SENT_ANSWER;
                    return joke.answer() + ANOTHER;
                }
```

Kode 16.5 (Lanjutan) *KnockKnockProtocol.java: tabel transisi sebagai kelas, versi keliru tanpa keadaan, dan metode bantu play yang menjalankan percakapan tanpa jaringan.*

```
                state = State.SENT_KNOCK;
                return "You're supposed to say \"" + joke.clue() + " who?\"! Try again. " +
KNOCK;
            case SENT_ANSWER:
                if ("y".equalsIgnoreCase(input)) {
                    current = (current + 1) % jokes.size();     // lelucon berikutnya,
kembali ke awal
                    state = State.SENT_KNOCK;
                    return KNOCK;
                }
                if ("n".equalsIgnoreCase(input)) {
                    state = State.FINISHED;
                    return BYE;
                }
                return "Please answer y or n.";
            default:
                throw new IllegalStateException("the conversation is over");
        }
    }

    /** Menggoda tetapi keliru: sama sekali tanpa keadaan, server hanya melihat baris
terakhir. */
    static String statelessReply(String input) {
        if (input == null || input.equalsIgnoreCase("y")) {
            return KNOCK;
        }
        if (input.equalsIgnoreCase("Who's there?")) {
            return "Lettuce";
        }
        if (input.endsWith(" who?")) {
            return "Lettuce in, it's cold out here!" + ANOTHER;
        }
        return BYE;
    }

    /** Memainkan seluruh percakapan: baris klien masuk, transkrip keluar. */
    static List<String> play(KnockKnockProtocol p, String... clientLines) {
        List<String> transcript = new ArrayList<>();
        transcript.add("S: " + p.processInput(null));
        for (String line : clientLines) {
            transcript.add("C: " + line);
            transcript.add("S: " + p.processInput(line));
        }
        return transcript;
    }

    private static int passed = 0;
```

Kode 16.5 (Lanjutan) *KnockKnockProtocol.java: tabel transisi sebagai kelas, versi keliru tanpa keadaan, dan metode bantu play yang menjalankan percakapan tanpa jaringan.*

```
    private static void check(String name, Object expected, Object actual) {
        if (!java.util.Objects.equals(expected, actual)) {
            throw new AssertionError(name + ": expected " + expected + " but got " +
actual);
        }
        passed++;
        System.out.println("PASS " + name);
    }

    public static void main(String[] args) {
        KnockKnockProtocol p = new KnockKnockProtocol(JokeStore.classics());
        List<String> t = play(p, "Who's there?", "Lettuce who?", "y",
                              "who's there?", "Harry who?", "n");
        t.forEach(System.out::println);
        check("finished after n", true, p.finished());
        check("second joke told", "S: Harry", t.get(8));

        KnockKnockProtocol q = new KnockKnockProtocol(JokeStore.classics());
        List<String> u = play(q, "Who is it?", "Who's there?", "Cabbage who?");
        check("wrong greeting", "S: You're supposed to say \"Who's there?\"! Try again.
Knock! Knock!", u.get(2));
        check("wrong clue", "S: You're supposed to say \"Lettuce who?\"! Try again. Knock!
Knock!", u.get(6));
        check("not finished", false, q.finished());
        check("stateless reply is fooled", "Lettuce in, it's cold out here! Want another?
(y/n)",
              statelessReply("Cabbage who?"));
        System.out.println("All " + passed + " checks passed.");
    }
}
```

```
S: Knock! Knock!
C: Who's there?
S: Lettuce
C: Lettuce who?
S: Lettuce in, it's cold out here! Want another? (y/n)
C: y
S: Knock! Knock!
C: who's there?
S: Harry
C: Harry who?
S: Harry up, the laundry is ready! Want another? (y/n)
C: n
S: Bye.
PASS finished after n
PASS second joke told
PASS wrong greeting
PASS wrong clue
PASS not finished
PASS stateless reply is fooled
All 6 checks passed.
```

Penyelesaian Python. **Kode 16.6** adalah mesin yang sama. Pernyataan `match` Python juga dapat dipakai pada Python 3.10 ke atas, tetapi rangkaian pernyataan `if` pada sebuah `Enum` sama jelasnya dan berjalan di mana saja.

Kode 16.6 `knock_knock_protocol.py`: mesin keadaan yang sama dalam Python, dengan uji yang sama.

```
# knock_knock_protocol.py - Soal 16.2: diagram alir knock-knock sebagai mesin keadaan.
import sys
import unittest
from enum import Enum, auto
from joke_store import classics

KNOCK = "Knock! Knock!"
ANOTHER = " Want another? (y/n)"
BYE = "Bye."


class State(Enum):
    """Empat kotak diagram alir tempat server menunggu, ditambah akhir."""
    WAITING = auto()
    SENT_KNOCK = auto()
    SENT_CLUE = auto()
    SENT_ANSWER = auto()
    FINISHED = auto()


class KnockKnockProtocol:
    def __init__(self, jokes):
        self._jokes = jokes
        self._state = State.WAITING
        self._current = 0                           # indeks lelucon yang sedang
diceritakan

    def finished(self):
        return self._state is State.FINISHED

    def process_input(self, text):
        """Berdasarkan ucapan klien barusan (None sebelum baris pertama), mengembalikan
        ucapan server berikutnya. Satu panggilan adalah satu panah diagram alir."""
        joke = self._jokes.get(self._current)
        said = (text or "").lower()
        if self._state is State.WAITING:
            self._state = State.SENT_KNOCK
            return KNOCK
        if self._state is State.SENT_KNOCK:
            if said == "who's there?":
                self._state = State.SENT_CLUE
                return joke.clue
            return f"You're supposed to say \"Who's there?\"! Try again. {KNOCK}"
        if self._state is State.SENT_CLUE:
            if said == f"{joke.clue} who?".lower():
                self._state = State.SENT_ANSWER
                return joke.answer + ANOTHER
```

Kode 16.6 (Lanjutan) *knock_knock_protocol.py: mesin keadaan yang sama dalam Python, dengan uji yang sama.*

```
            self._state = State.SENT_KNOCK
            return f"You're supposed to say \"{joke.clue} who?\"! Try again. {KNOCK}"
        if self._state is State.SENT_ANSWER:
            if said == "y":
                self._current = (self._current + 1) % self._jokes.size()   # kembali ke
awal
                self._state = State.SENT_KNOCK
                return KNOCK
            if said == "n":
                self._state = State.FINISHED
                return BYE
            return "Please answer y or n."
        raise RuntimeError("the conversation is over")


def stateless_reply(text):
    """Menggoda tetapi keliru: sama sekali tanpa keadaan, server hanya melihat baris
terakhir."""
    if text is None or text.lower() == "y":
        return KNOCK
    if text.lower() == "who's there?":
        return "Lettuce"
    if text.endswith(" who?"):
        return "Lettuce in, it's cold out here!" + ANOTHER
    return BYE


def play(protocol, *client_lines):
    """Memainkan seluruh percakapan: baris klien masuk, transkrip keluar."""
    transcript = ["S: " + protocol.process_input(None)]
    for line in client_lines:
        transcript.append("C: " + line)
        transcript.append("S: " + protocol.process_input(line))
    return transcript


class TestProtocol(unittest.TestCase):
    def test_two_jokes_then_bye(self):
        p = KnockKnockProtocol(classics())
        t = play(p, "Who's there?", "Lettuce who?", "y", "who's there?", "Harry who?", "n")
        self.assertTrue(p.finished())
        self.assertEqual("S: Harry", t[8])
```

Kode 16.6 (Lanjutan) *knock_knock_protocol.py: mesin keadaan yang sama dalam Python, dengan uji yang sama.*

```
    def test_wrong_lines_restart_the_joke(self):
        q = KnockKnockProtocol(classics())
        u = play(q, "Who is it?", "Who's there?", "Cabbage who?")
        self.assertEqual("S: You're supposed to say \"Who's there?\"! Try again. Knock!
Knock!", u[2])
        self.assertEqual("S: You're supposed to say \"Lettuce who?\"! Try again. Knock!
Knock!", u[6])
        self.assertFalse(q.finished())

    def test_stateless_reply_is_fooled(self):
        self.assertEqual("Lettuce in, it's cold out here! Want another? (y/n)",
                         stateless_reply("Cabbage who?"))


if __name__ == "__main__":
    for line in play(KnockKnockProtocol(classics()), "Who's there?", "Lettuce who?", "y",
                     "who's there?", "Harry who?", "n"):
        print(line)
    sys.stdout.flush()
    unittest.main(verbosity=2)
```

```
S: Knock! Knock!
C: Who's there?
S: Lettuce
C: Lettuce who?
S: Lettuce in, it's cold out here! Want another? (y/n)
C: y
S: Knock! Knock!
C: who's there?
S: Harry
C: Harry who?
S: Harry up, the laundry is ready! Want another? (y/n)
C: n
S: Bye.
test_stateless_reply_is_fooled (__main__.TestProtocol.test_stateless_reply_is_fooled) ...
ok
test_two_jokes_then_bye (__main__.TestProtocol.test_two_jokes_then_bye) ... ok
test_wrong_lines_restart_the_joke (__main__.TestProtocol.test_wrong_lines_restart_the_joke)
... ok

----------------------------------------------------------------------
Ran 3 tests in 0.000s

OK
```

Baris demi baris. `enum State` menamai kelima keadaan pada **Gambar 16.3**; *enum* lebih baik daripada `int` atau `String` karena kompilator menolak keadaan apa pun yang tidak ada. Kedua *field* `state` dan `current` adalah seluruh ingatan percakapan: di kotak mana kita berada dan lelucon mana yang sedang diceritakan. `processInput` mula-mula mengambil lelucon saat ini, lalu bercabang menurut keadaannya; setiap `case` adalah satu kelompok baris pada **Tabel 16.5**. `"Who's there?".equalsIgnoreCase(input)` menaruh konstanta di depan, sehingga masukan `null` menghasilkan `false` alih-alih melempar `NullPointerException`. Cabang `y` menghitung `(current + 1) % jokes.size()`: operator sisa bagi membuat nomor lelucon kembali dari lelucon terakhir ke 0, sehingga pengguna boleh meminta lebih banyak lelucon daripada yang tersimpan. Cabang `default` hanya tercapai dalam keadaan `FINISHED`; memanggil protokol setelah "Bye." adalah kesalahan pemrograman, sehingga ia melempar `IllegalStateException`. Metode bantu `play` mengalirkan daftar baris klien melalui protokol dan mencatat transkripnya, dengan `S:` untuk server dan `C:` untuk klien. Uji pertama memainkan dua lelucon dan memeriksa bahwa baris

kesembilan adalah petunjuk kedua; uji kedua mengirim sapaan yang salah dan petunjuk yang salah, lalu memeriksa kedua pengulangan yang sopan. Versi Python mengubah masukan menjadi huruf kecil satu kali (`said`) dan memakai `(text or "")` sehingga `None` menjadi *string* kosong, padanan perbandingan konstanta-di-depan pada Java.

Penyelesaian keliru yang menggoda. `statelessReply` sama sekali tidak memiliki *field*; ia hanya melihat baris terakhir. Kodenya lebih pendek, dan bahkan menangani jalur mulus. Namun tanpa keadaan ia tidak dapat mengetahui petunjuk *mana* yang telah ia sebutkan, sehingga ia menjawab dengan kalimat pamungkas untuk baris apa pun yang berakhir dengan ” who?“: pemeriksaan terakhir di setiap program memperlihatkannya membalas”Lettuce in, it’s cold out here!” kepada “Cabbage who?”. Ia juga tidak dapat menceritakan lelucon kedua, karena tidak ingat lelucon mana yang pertama. Setiap kali balasan yang benar bergantung pada apa yang dikatakan sebelumnya, objek itu memerlukan keadaan, dan mesin keadaan adalah rancangan yang membuat keadaan itu eksplisit.

> **UJI DIRI.** Tambahkan sebuah aturan: dalam keadaan `SENT_KNOCK`, masukan `quit` langsung mengakhiri percakapan dengan balasan `Bye.`. Baris mana yang Anda tambahkan ke **Tabel 16.5**, dan baris kode mana yang berubah?
>
> *Jawaban.* Barisnya adalah (`SENT_KNOCK`, `quit`, `Bye.`, `FINISHED`). Di Java, di dalam `case SENT_KNOCK`, sebelum pemeriksaan “Who’s there?”: `if ("quit".equalsIgnoreCase(input)) { state = State.FINISHED; return BYE; }`. Di Python, tiga baris yang sama di dalam cabang `SENT_KNOCK`. Klien tidak perlu diubah, karena ia memang sudah berhenti pada `Bye.`.

16.5 Soal-Soal tentang Membangun dan Menguji Sistem Klien–Server

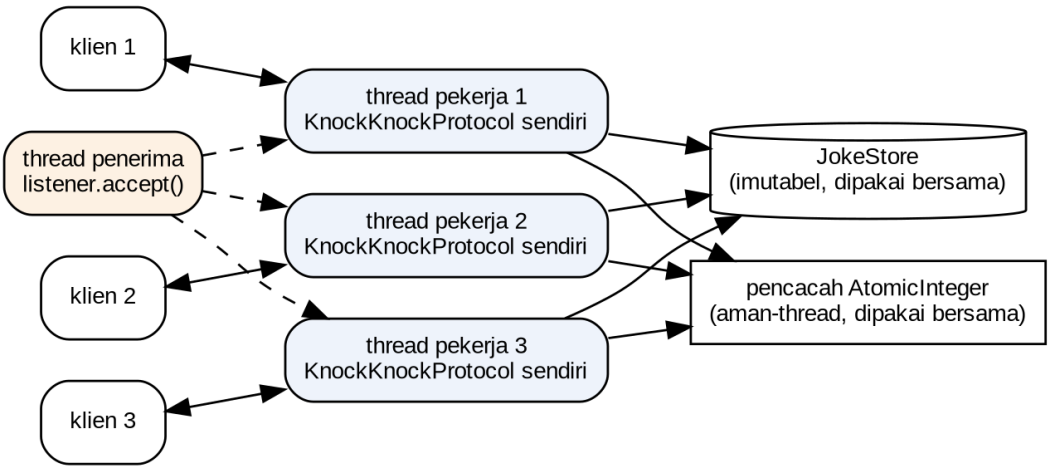
16.5.1 Soal 16.3 — Server untuk banyak klien, dan klien konsol

> **SOAL.** Build the server and the client of the application. The server listens on a port and must serve several clients at the same time, each in its own conversation. The client connects to the server, shows each line from the server to the user, and sends the user’s replies. Show that a second client is served while a first client is still in the middle of its conversation.
>
> *Dengan kata lain:* bangunlah server dan klien aplikasi itu. Server mendengarkan pada sebuah port dan harus melayani beberapa klien sekaligus, masing-masing dalam percakapannya sendiri. Klien menyambung ke server, menampilkan setiap baris dari server kepada pengguna, dan mengirim jawaban pengguna. Tunjukkan bahwa klien kedua dilayani selagi klien pertama masih berada di tengah percakapannya.

Dengan bahasa kita sendiri. Inilah kerangka berjalan dari Langkah 7: protokol Soal 16.2 dan penyimpan Soal 16.1, disatukan oleh soket sungguhan. *Thread* utama server hanya menunggu koneksi baru; setiap koneksi yang diterima mendapat *thread*-nya sendiri dan objek `KnockKnockProtocol`-nya sendiri, sehingga percakapan-percakapan tidak dapat saling mengganggu. Klien adalah sebuah gelung: baca satu baris, tampilkan, tanyakan kepada pengguna, kirim jawabannya, sampai `Bye.` atau sampai server memutus sambungan.

Rancangan. **Gambar 16.4** memperlihatkan *thread-thread* server. *Thread penerima* (*acceptor*) menjalankan `accept()` dalam gelung; `accept()` menunggu sampai ada klien yang menyambung lalu mengembalikan `Socket` baru khusus untuk klien itu, seperti diuraikan Bab 15. Penerima menyerahkan soket itu kepada *thread pekerja* dan segera kembali ke `accept()`. Setiap pekerja memiliki objek protokol yang baru. Satu-satunya objek yang dipakai bersama oleh semua pekerja

adalah penyimpan lelucon, yang imutabel sehingga aman dipakai bersama tanpa kunci apa pun (Bab 13), dan pencacah percakapan yang selesai dan yang terputus, yang berupa `AtomicInteger` sehingga aman diperbarui dari banyak *thread* (Bab 14).



Gambar 16.4 *Thread-thread* server *knock-knock*. Setiap klien memiliki *thread* pekerja dan objek protokolnya sendiri; hanya objek imutabel atau aman-*thread* yang dipakai bersama.

Pengguna klien dimodelkan sebagai *fungsi* dari baris server ke jawabannya: `Function<String, String>` di Java, objek apa pun yang dapat dipanggil di Python. Orang di depan papan ketik adalah salah satu fungsi semacam itu; `PoliteUser`, yang selalu menjawab dengan benar dan menginginkan sejumlah lelucon tertentu, adalah fungsi lain. Keputusan kecil ini membuat seluruh klien dapat diuji, karena sebuah uji dapat mengganti orang itu dengan `PoliteUser`.

Penyelesaian Java. **Kode 16.7** adalah server-nya. Metode `main`-nya menjalankannya pada port 4444, atau pada port yang diberikan di baris perintah.

Kode 16.7 `KnockKnockServer.java`: *thread* penerima, satu pekerja per klien, metode statis `converse` yang menjalankan satu percakapan, dan mode satu-per-satu yang sengaja dibuat keliru.

```java
// KnockKnockServer.java - Soal 16.3: server yang bercakap dengan banyak klien sekaligus.
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.net.ServerSocket;
import java.net.Socket;
import java.nio.charset.StandardCharsets;
import java.util.List;
import java.util.concurrent.CopyOnWriteArrayList;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.atomic.AtomicInteger;

public class KnockKnockServer {

    private final JokeStore jokes;                  // imutabel, jadi aman dipakai bersama
    private final boolean oneAtATime;               // true hanya untuk varian yang keliru
    private final ExecutorService workers = Executors.newCachedThreadPool();
    private final AtomicInteger finished = new AtomicInteger();
    private final AtomicInteger leftEarly = new AtomicInteger();
    private final List<String> log = new CopyOnWriteArrayList<>();
    private ServerSocket listener;

    public KnockKnockServer(JokeStore jokes) {
        this(jokes, false);
    }

    KnockKnockServer(JokeStore jokes, boolean oneAtATime) {
        this.jokes = jokes;
        this.oneAtATime = oneAtATime;
    }

    /** Membuka soket pendengar (port 0 berarti "port bebas mana saja") dan mengembalikan
port-nya. */
    public int start(int port) throws IOException {
        listener = new ServerSocket(port);
        Thread acceptor = new Thread(this::acceptLoop, "acceptor");
        acceptor.setDaemon(true);                   // tidak membuat program tetap hidup
        acceptor.start();
        return listener.getLocalPort();
    }
```

Kode 16.7 (Lanjutan) *KnockKnockServer.java: thread penerima, satu pekerja per klien, metode statis converse yang menjalankan satu percakapan, dan mode satu-per-satu yang sengaja dibuat keliru.*

```
    private void acceptLoop() {
        try {
            while (true) {
                Socket client = listener.accept();          // menunggu klien berikutnya
                if (oneAtATime) {
                    serve(client);                          // keliru: semua yang lain
menunggu
                } else {
                    workers.execute(() -> serve(client));   // satu thread per percakapan
                }
            }
        } catch (IOException e) {
            // stop() menutup soket pendengar, jadi accept() gagal: gelung berakhir
        }
    }

    private void serve(Socket client) {
        try (client;
             BufferedReader in = new BufferedReader(new InputStreamReader(
                     client.getInputStream(), StandardCharsets.UTF_8));
             PrintWriter out = new PrintWriter(new OutputStreamWriter(
                     client.getOutputStream(), StandardCharsets.UTF_8), true)) {
            if (converse(in, out, new KnockKnockProtocol(jokes))) {
                finished.incrementAndGet();
            } else {
                leftEarly.incrementAndGet();
                log.add("client left before the end");
            }
        } catch (IOException e) {
            leftEarly.incrementAndGet();
            log.add("connection lost: " + e.getClass().getSimpleName());
        }
    }

    /** Satu percakapan. Mengembalikan true jika mencapai "Bye.", false jika klien
menghilang. */
    static boolean converse(BufferedReader in, PrintWriter out, KnockKnockProtocol
protocol)
            throws IOException {
        out.println(protocol.processInput(null));           // server berbicara lebih dulu
        while (!protocol.finished()) {
            String line = in.readLine();
            if (line == null) {
                return false;                               // klien menutup koneksi
            }
            out.println(protocol.processInput(line));
        }
        return true;
    }
```

Kode 16.7 (Lanjutan) *KnockKnockServer.java: thread penerima, satu pekerja per klien, metode statis converse yang menjalankan satu percakapan, dan mode satu-per-satu yang sengaja dibuat keliru.*

```
    public void stop() throws IOException {
        listener.close();
        workers.shutdown();
    }

    int finishedCount() {
        return finished.get();
    }

    int leftEarlyCount() {
        return leftEarly.get();
    }

    List<String> log() {
        return List.copyOf(log);
    }

    public static void main(String[] args) throws Exception {
        int port = (args.length > 0) ? Integer.parseInt(args[0]) : 4444;
        KnockKnockServer server = new KnockKnockServer(JokeStore.classics());
        System.out.println("Knock-knock server listening on port " + server.start(port));
        System.out.println("Press Ctrl+C to stop it.");
        Thread.currentThread().join();                      // menunggu selamanya
    }
}
```

Bila dijalankan dari terminal, server mencetak dua baris lalu menunggu:

```
Knock-knock server listening on port 4444
Press Ctrl+C to stop it.
```

Kode 16.8 adalah kliennya. Dengan dua argumen, ia bercakap dengan server sungguhan untuk orang di depan papan ketik; tanpa argumen, ia menjalankan server di dalam program yang sama pada port yang bebas lalu menjalankan uji diri.

Kode 16.8 `KnockKnockClient.java`: gelung klien, `PoliteUser` yang berskrip, mode interaktif, dan uji diri yang juga memperlihatkan server satu-per-satu menelantarkan klien kedua.

```java
// KnockKnockClient.java - Soal 16.3: klien, pengguna berskrip, dan uji diri.
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.net.Socket;
import java.net.SocketTimeoutException;
import java.nio.charset.StandardCharsets;
import java.util.ArrayList;
import java.util.List;
import java.util.function.Function;

public class KnockKnockClient {

    /** Pengguna yang selalu menjawab dengan benar dan menginginkan sejumlah lelucon
tertentu. */
    static final class PoliteUser implements Function<String, String> {
        private int jokesWanted;

        PoliteUser(int jokesWanted) {
            this.jokesWanted = jokesWanted;
        }

        @Override
        public String apply(String serverLine) {
            if (serverLine.endsWith("Knock! Knock!")) {
                return "Who's there?";
            }
            if (serverLine.endsWith("(y/n)")) {
                jokesWanted--;
                return (jokesWanted > 0) ? "y" : "n";
            }
            return serverLine + " who?";                    // petunjuknya: "<clue> who?"
        }
    }
```

Kode 16.8 (Lanjutan) *KnockKnockClient.java: gelung klien, PoliteUser yang berskrip, mode interaktif, dan uji diri yang juga memperlihatkan server satu-per-satu menelantarkan klien kedua.*

```
    /** Bercakap dengan server melalui reader dan writer apa pun; mengembalikan
transkripnya. */
    static List<String> converse(BufferedReader in, PrintWriter out,
                                 Function<String, String> user) throws IOException {
        List<String> transcript = new ArrayList<>();
        String fromServer;
        while ((fromServer = in.readLine()) != null) {      // null: server memutus
sambungan
            transcript.add("S: " + fromServer);
            if (fromServer.equals("Bye.")) {
                break;
            }
            String reply = user.apply(fromServer);
            if (reply == null) {
                break;                                      // pengguna menyerah
            }
            transcript.add("C: " + reply);
            out.println(reply);
        }
        return transcript;
    }

    /** Membuka soket, bercakap, lalu menutup semuanya lagi. */
    static List<String> talkTo(String host, int port, Function<String, String> user)
            throws IOException {
        try (Socket socket = new Socket(host, port);
             BufferedReader in = new BufferedReader(new InputStreamReader(
                     socket.getInputStream(), StandardCharsets.UTF_8));
             PrintWriter out = new PrintWriter(new OutputStreamWriter(
                     socket.getOutputStream(), StandardCharsets.UTF_8), true)) {
            return converse(in, out, user);
        }
    }

    /** Menyambung, membaca baris pertama dengan batas waktu, dan membiarkan soket tetap
terbuka. */
    static String firstLineWithin(int port, int millis, List<Socket> keepOpen) throws
IOException {
        Socket s = new Socket("localhost", port);
        keepOpen.add(s);
        s.setSoTimeout(millis);                             // readLine menyerah setelah
millis
        BufferedReader in = new BufferedReader(new InputStreamReader(s.getInputStream()));
        try {
            return in.readLine();
        } catch (SocketTimeoutException e) {
            return "(nothing within " + millis + " ms)";
        }
    }
```

Kode 16.8 (Lanjutan) *KnockKnockClient.java: gelung klien, PoliteUser yang berskrip, mode interaktif, dan uji diri yang juga memperlihatkan server satu-per-satu menelantarkan klien kedua.*

```
    private static int passed = 0;

    private static void check(String name, Object expected, Object actual) {
        if (!java.util.Objects.equals(expected, actual)) {
            throw new AssertionError(name + ": expected " + expected + " but got " +
actual);
        }
        passed++;
        System.out.println("PASS " + name);
    }

    public static void main(String[] args) throws Exception {
        if (args.length == 2) {                          // java KnockKnockClient
localhost 4444
            BufferedReader keyboard = new BufferedReader(new InputStreamReader(System.in));
            Function<String, String> human = line -> {
                System.out.println("Server: " + line);
                System.out.print("You: ");
                try {
                    return keyboard.readLine();
                } catch (IOException e) {
                    return null;
                }
            };
            List<String> t = talkTo(args[0], Integer.parseInt(args[1]), human);
            System.out.println("Server: " + t.get(t.size() - 1).substring(3));
            return;
        }
        KnockKnockServer server = new KnockKnockServer(JokeStore.classics());
        int port = server.start(0);                      // port bebas mana saja
        List<String> t = talkTo("localhost", port, new PoliteUser(2));
        t.forEach(System.out::println);
        check("ends with Bye.", "S: Bye.", t.get(t.size() - 1));
        check("13 lines", 13, t.size());

        List<Socket> open = new ArrayList<>();
        firstLineWithin(port, 500, open);                // klien A menyambung lalu diam
        check("B served while A talks", "Knock! Knock!", firstLineWithin(port, 500, open));
        server.stop();
```

Kode 16.8 (Lanjutan) *KnockKnockClient.java: gelung klien, PoliteUser yang berskrip, mode interaktif, dan uji diri yang juga memperlihatkan server satu-per-satu menelantarkan klien kedua.*

```
        KnockKnockServer wrong = new KnockKnockServer(JokeStore.classics(), true);
        int wrongPort = wrong.start(0);
        firstLineWithin(wrongPort, 500, open);           // A lagi
        String b = firstLineWithin(wrongPort, 500, open);   // B harus menunggu A selesai
        System.out.println("one-at-a-time server, client B got: " + b);
        check("B starved by the wrong server", "(nothing within 500 ms)", b);
        for (Socket s : open) {
            s.close();
        }
        wrong.stop();
        System.out.println("All " + passed + " checks passed.");
    }
}
```

```
S: Knock! Knock!
C: Who's there?
S: Lettuce
C: Lettuce who?
S: Lettuce in, it's cold out here! Want another? (y/n)
C: y
S: Knock! Knock!
C: Who's there?
S: Harry
C: Harry who?
S: Harry up, the laundry is ready! Want another? (y/n)
C: n
S: Bye.
PASS ends with Bye.
PASS 13 lines
PASS B served while A talks
one-at-a-time server, client B got: (nothing within 500 ms)
PASS B starved by the wrong server
All 4 checks passed.
```

Untuk memakai program-program ini seperti yang dilakukan dua orang, bukalah dua terminal di folder `java`. Di terminal pertama, ketik `java KnockKnockServer`; di terminal kedua, `java KnockKnockClient localhost 4444`, lalu jawablah lelucon-leluconnya. Sebuah sesi tampak seperti ini (setelah setiap `You:` pengguna mengetik jawabannya, yang ditampilkan terminal pada baris yang sama):

```
Server: Knock! Knock!
You: Who's there?
Server: Lettuce
You: Lettuce who?
Server: Lettuce in, it's cold out here! Want another? (y/n)
You: n
Server: Bye.
```

Penyelesaian Python. **Kode 16.9** dan **Kode 16.10** memiliki struktur yang sama. `socket.makefile` pada Python membungkus soket menjadi objek berkas teks, sehingga membaca satu baris cukup dengan `reader.readline()` dan menulis satu baris dengan `writer.write(...)` yang diikuti `writer.flush()`.

Kode 16.9 `knock_knock_server.py`: server yang sama dengan pekerja `threading.Thread` dan sebuah kunci di sekeliling pencacahnya.

```
# knock_knock_server.py - Soal 16.3: server yang bercakap dengan banyak klien sekaligus.
import socket
import sys
import threading
from joke_store import classics
from knock_knock_protocol import KnockKnockProtocol


def send(writer, line):
    """Menulis satu baris dan langsung mendorongnya ke jaringan."""
    writer.write(line + "\n")
    writer.flush()


def converse(reader, writer, protocol):
    """Satu percakapan. Mengembalikan True jika mencapai "Bye.", False jika klien
menghilang."""
    send(writer, protocol.process_input(None))          # server berbicara lebih dulu
    while not protocol.finished():
        line = reader.readline()
        if line == "":
            return False                                # klien menutup koneksi
        send(writer, protocol.process_input(line.rstrip("\r\n")))
    return True


class KnockKnockServer:
    def __init__(self, jokes, one_at_a_time=False):
        self._jokes = jokes                             # imutabel, jadi aman dipakai
bersama
        self._one_at_a_time = one_at_a_time             # True hanya untuk varian yang
keliru
        self._lock = threading.Lock()                   # menjaga tiga field di bawah ini
        self._finished = 0
        self._left_early = 0
        self._log = []
        self._listener = None

    def start(self, port):
        """Membuka soket pendengar (port 0 berarti "port bebas mana saja") dan
mengembalikan port-nya."""
        self._listener = socket.create_server(("localhost", port))
        threading.Thread(target=self._accept_loop, daemon=True).start()
        return self._listener.getsockname()[1]
```

Kode 16.9 (Lanjutan) *knock_knock_server.py: server yang sama dengan pekerja threading.Thread dan sebuah kunci di sekeliling pencacahnya.*

```
    def _accept_loop(self):
        try:
            while True:
                client, _address = self._listener.accept()     # menunggu klien berikutnya
                if self._one_at_a_time:
                    self._serve(client)                         # keliru: semua yang lain
menunggu
                else:
                    threading.Thread(target=self._serve, args=(client,),
daemon=True).start()
        except OSError:
            pass                    # stop() menutup soket pendengar: gelung berakhir

    def _serve(self, client):
        try:
            with client, client.makefile("r", encoding="utf-8") as reader, \
                    client.makefile("w", encoding="utf-8") as writer:
                done = converse(reader, writer, KnockKnockProtocol(self._jokes))
            self._record(done, "client left before the end")
        except OSError as e:
            self._record(False, f"connection lost: {type(e).__name__}")

    def _record(self, done, message):
        with self._lock:
            if done:
                self._finished += 1
            else:
                self._left_early += 1
                self._log.append(message)

    def counts(self):
        with self._lock:
            return self._finished, self._left_early, list(self._log)

    def stop(self):
        try:
            self._listener.shutdown(socket.SHUT_RDWR)   # membangunkan accept() di Linux
        except OSError:
            pass
        self._listener.close()
```

Kode 16.9 (Lanjutan) *knock_knock_server.py: server yang sama dengan pekerja threading.Thread dan sebuah kunci di sekeliling pencacahnya.*

```
if __name__ == "__main__":
    server = KnockKnockServer(classics())
    print("Knock-knock server listening on port",
          server.start(int(sys.argv[1]) if len(sys.argv) > 1 else 4444))
    print("Press Ctrl+C to stop it.")
    try:
        threading.Event().wait()                        # menunggu selamanya
    except KeyboardInterrupt:
        server.stop()
```

Kode 16.10 `knock_knock_client.py`: klien, `PoliteUser` sebagai kelas yang dapat dipanggil, dan tiga uji termasuk klien kedua yang ditelantarkan.

```
# knock_knock_client.py - Soal 16.3: klien, pengguna berskrip, dan uji diri.
import socket
import sys
import unittest
from joke_store import classics
from knock_knock_server import KnockKnockServer, send


class PoliteUser:
    """Pengguna yang selalu menjawab dengan benar dan menginginkan sejumlah lelucon
tertentu."""

    def __init__(self, jokes_wanted):
        self._jokes_wanted = jokes_wanted

    def __call__(self, server_line):
        if server_line.endswith("Knock! Knock!"):
            return "Who's there?"
        if server_line.endswith("(y/n)"):
            self._jokes_wanted -= 1
            return "y" if self._jokes_wanted > 0 else "n"
        return server_line + " who?"                    # petunjuknya: "<clue> who?"


def converse(reader, writer, user):
    """Bercakap dengan server melalui reader dan writer apa pun; mengembalikan
transkripnya."""
    transcript = []
    while True:
        line = reader.readline()
        if line == "":
            break                                       # server memutus sambungan
        from_server = line.rstrip("\r\n")
        transcript.append("S: " + from_server)
        if from_server == "Bye.":
            break
        reply = user(from_server)
        if reply is None:
            break                                       # pengguna menyerah
        transcript.append("C: " + reply)
        send(writer, reply)
    return transcript
```

Kode 16.10 (Lanjutan) *knock_knock_client.py: klien, PoliteUser sebagai kelas yang dapat dipanggil, dan tiga uji termasuk klien kedua yang ditelantarkan.*

```
def talk_to(host, port, user):
    """Membuka soket, bercakap, lalu menutup semuanya lagi."""
    with socket.create_connection((host, port)) as s, \
            s.makefile("r", encoding="utf-8") as reader, \
            s.makefile("w", encoding="utf-8") as writer:
        return converse(reader, writer, user)


def first_line_within(port, seconds, keep_open):
    """Menyambung, membaca baris pertama dengan batas waktu, dan membiarkan soket tetap
terbuka."""
    s = socket.create_connection(("localhost", port))
    keep_open.append(s)
    s.settimeout(seconds)                               # readline menyerah setelah ini
    try:
        return s.makefile("r", encoding="utf-8").readline().rstrip("\r\n")
    except TimeoutError:
        return f"(nothing within {int(seconds * 1000)} ms)"


class TestClientAndServer(unittest.TestCase):
    def test_two_jokes(self):
        server = KnockKnockServer(classics())
        t = talk_to("localhost", server.start(0), PoliteUser(2))
        server.stop()
        self.assertEqual("S: Bye.", t[-1])
        self.assertEqual(13, len(t))

    def test_second_client_is_served_at_once(self):
        server, open_sockets = KnockKnockServer(classics()), []
        port = server.start(0)
        first_line_within(port, 0.5, open_sockets)      # klien A menyambung lalu diam
        self.assertEqual("Knock! Knock!", first_line_within(port, 0.5, open_sockets))
        for s in open_sockets:
            s.close()
        server.stop()
```

Kode 16.10 (Lanjutan) *knock_knock_client.py: klien, PoliteUser sebagai kelas yang dapat dipanggil, dan tiga uji termasuk klien kedua yang ditelantarkan.*

```
    def test_one_at_a_time_server_starves_the_second_client(self):
        wrong, open_sockets = KnockKnockServer(classics(), one_at_a_time=True), []
        port = wrong.start(0)
        first_line_within(port, 0.5, open_sockets)
        b = first_line_within(port, 0.5, open_sockets)
        print("\none-at-a-time server, client B got:", b, end=" ... ", file=sys.stderr)
        self.assertEqual("(nothing within 500 ms)", b)
        for s in open_sockets:
            s.close()
        wrong.stop()


def human(line):
    print("Server:", line)
    try:
        return input("You: ")
    except EOFError:
        return None


if __name__ == "__main__":
    if len(sys.argv) == 3:                              # python knock_knock_client.py
localhost 4444
        print("Server:", talk_to(sys.argv[1], int(sys.argv[2]), human)[-1][3:])
    else:
        demo = KnockKnockServer(classics())
        for line in talk_to("localhost", demo.start(0), PoliteUser(2)):
            print(line)
        demo.stop()
        sys.stdout.flush()
        unittest.main(verbosity=2)
```

```
S: Knock! Knock!
C: Who's there?
S: Lettuce
C: Lettuce who?
S: Lettuce in, it's cold out here! Want another? (y/n)
C: y
S: Knock! Knock!
C: Who's there?
S: Harry
C: Harry who?
S: Harry up, the laundry is ready! Want another? (y/n)
C: n
S: Bye.
test_one_at_a_time_server_starves_the_second_client
(__main__.TestClientAndServer.test_one_at_a_time_server_starves_the_second_client) ...
one-at-a-time server, client B got: (nothing within 500 ms) ... ok
test_second_client_is_served_at_once
(__main__.TestClientAndServer.test_second_client_is_served_at_once) ... ok
test_two_jokes (__main__.TestClientAndServer.test_two_jokes) ... ok

----------------------------------------------------------------------
Ran 3 tests in 0.511s

OK
```

Baris demi baris. Pada **Kode 16.7**, `start(int port)` membuat `ServerSocket`. Port 0 meminta sistem operasi memberikan port bebas mana saja, dan itulah yang dikehendaki uji: dua kali jalannya uji tidak pernah berebut port yang sama, dan `getLocalPort()` melaporkan port yang dipilih. Penerima adalah *thread daemon*, yaitu *thread* yang tidak membuat program Java tetap hidup dengan sendirinya, sehingga uji dapat berakhir tanpa menghentikannya secara manual.

`acceptLoop` memanggil `listener.accept()`, lalu `workers.execute(() -> serve(client))`: *lambda* itu adalah `Runnable` yang dijalankan kumpulan *thread* `Executors.newCachedThreadPool()` pada *thread* yang dibuatnya atau dipakainya ulang. Di `serve`, kepala `try (client; BufferedReader in = ...; PrintWriter out = ...)` menutup soket dan kedua *stream* apa pun yang terjadi; `client` boleh dicantumkan di situ karena nilainya secara efektif final. `true` pada `new PrintWriter(..., true)` menyalakan *autoflush*: setiap `println` langsung mendorong byte-nya ke jaringan. Soal 16.13 memperlihatkan apa yang terjadi tanpanya. `converse` bersifat `static` dan menerima *reader* serta *writer* alih-alih soket, sehingga Soal 16.4 dapat mengujinya dengan *string*. Ia berbicara lebih dulu (`processInput(null)`), lalu mengulang sampai protokol selesai, dan mengembalikan `false` jika `readLine()` mengembalikan `null`, yang merupakan cara Java melaporkan bahwa pihak lain menutup koneksi.

Pada **Kode 16.8**, `converse` adalah bayangan cerminnya: baca, catat, berhenti pada `Bye.`, tanyakan kepada pengguna, kirim. Ia juga berhenti ketika `readLine()` mengembalikan `null` (server memutus sambungan) dan ketika fungsi pengguna mengembalikan `null` (orangnya menekan Ctrl+D atau Ctrl+Z). `talkTo` membungkus `converse` dalam `try`-with-resources yang membuka dan menutup soket. `firstLineWithin` memanggil `setSoTimeout(millis)`, sesudah itu `readLine()` yang menunggu lebih dari `millis` milidetik akan melempar `SocketTimeoutException`; sebuah uji tidak boleh sampai dapat menunggu selamanya. Uji diri menjalankan percakapan dua lelucon, memeriksa bahwa panjangnya 13 baris dan berakhir dengan `Bye.`, lalu menyambungkan klien A yang diam dan memeriksa bahwa klien B tetap mendapat “Knock! Knock!” dalam 500 ms.

Server Python memakai `socket.create_server(("localhost", port))`, yang membuat soket, mengikatnya, dan mulai mendengarkan dalam satu panggilan, serta `threading.Thread(target=..., args=(client,), daemon=True)` untuk setiap pekerja. Karena Python tidak memiliki `AtomicInteger`, pencacah-pencacahnya berada di balik satu `threading.Lock`, dan `counts()` mengembalikan potret ketiganya yang konsisten di bawah kunci itu. `stop()` memanggil `shutdown` sebelum `close`, karena di Linux *thread* yang tertahan di `accept()` hanya dapat dibangunkan dengan andal oleh `shutdown`.

Penyelesaian keliru yang menggoda. Server terpendek memanggil `serve(client)` langsung di dalam gelung `accept`, dan itulah yang dilakukan `oneAtATime`. Server itu bekerja sempurna dengan satu klien, dan memang begitulah kebanyakan orang mengujinya. Dengan dua klien, klien kedua berhasil menyambung (sistem operasi menuntaskan koneksi dan memasukkannya ke antrean), tetapi tidak ada yang membaca darinya atau menulis kepadanya sampai percakapan pertama selesai. Uji diri memperlihatkan persis hal ini: terhadap server yang keliru, klien B tidak mendengar apa pun dalam 500 ms. Seorang pengguna *knock-knock* yang pergi membuat kopi di tengah sebuah lelucon akan menahan seluruh klub.

> **UJI DIRI.** Server pada **Kode 16.7** membuat *thread* baru untuk setiap klien. Apa yang terjadi jika sepuluh ribu klien menyambung sekaligus, dan bagaimana Anda membatasinya?
>
> *Jawaban.* `newCachedThreadPool` membuat sampai sepuluh ribu *thread*, masing-masing dengan tumpukan (*stack*) yang biasanya berukuran ratusan kilobyte atau lebih, sehingga memori dapat habis. Pakailah `Executors.newFixedThreadPool(200)`: paling banyak 200 percakapan berjalan bersamaan dan sisanya menunggu dalam antrean kumpulan itu. (Versi Java terbaru juga menyediakan *virtual thread*, `Executors.newVirtualThreadPerTaskExecutor()`, yang cukup murah untuk satu per klien bahkan dalam jumlah besar.) Di Python, pakailah `ThreadPoolExecutor(max_workers=200)` dan serahkan `self._serve` kepadanya.

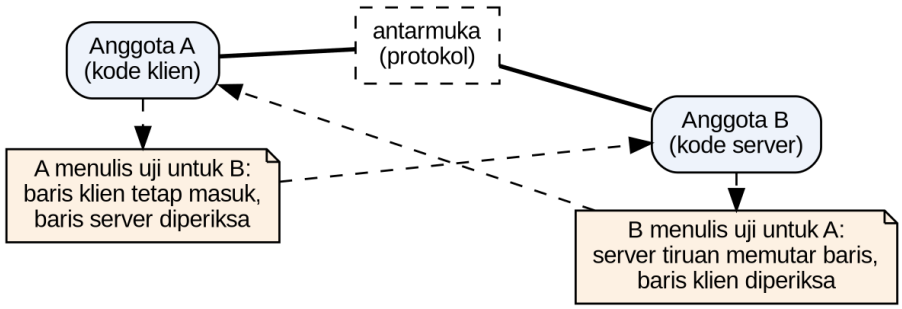
16.5.2 Soal 16.4 — Setiap pihak menguji pihak lain, dengan *stub*

> **SOAL.** Think about test cases across the interface between two members. Member A uses the functions of member B through an interface, so A designs test cases for B; B uses A's functions, so B designs test cases for A. Apply this to the client and the server of the Knock-Knock Club, using stubs so that the tests need no running network.
>
> *Dengan kata lain:* pikirkan kasus uji melintasi antarmuka antara dua anggota. Anggota A memakai fungsi anggota B melalui sebuah antarmuka, sehingga A merancang kasus uji untuk B; B memakai fungsi A, sehingga B merancang kasus uji untuk A. Terapkan hal ini pada klien dan server Knock-Knock Club, dengan memakai *stub* sehingga uji-ujinya tidak memerlukan jaringan yang berjalan.

Dengan bahasa kita sendiri. Orang yang *memakai* sebuah kontrak adalah orang terbaik untuk mengujinya, karena ia tahu apa yang ia andalkan. Di sini kontrak antara klien dan server adalah protokolnya. Pemilik klien menulis uji yang mengumpankan baris klien yang tetap ke metode `converse` milik server dan memeriksa balasannya; pemilik server menulis uji yang mengumpankan baris server yang tetap ke `converse` milik klien dan memeriksa apa yang diucapkan klien sebagai balasan. Dalam kedua kasus, pihak lawan digantikan oleh sebuah *stub*.

> **DEFINISI.** *Stub* adalah pengganti minimal bagi komponen yang menjadi tumpuan kode yang diuji. Ia memberikan jawaban tetap dan tidak memiliki logika sendiri. Uji dengan *stub* memeriksa satu komponen secara tersendiri, dengan cepat dan tanpa bagian sistem lainnya.

Rancangan. **Gambar 16.5** memperlihatkan susunannya. *Stub* di setiap sisi adalah metode bantu mungil yang sama, `replay(lines)`, yang mengubah daftar baris menjadi sebuah *reader*, seolah-olah pihak lawan mengirimkan persis baris-baris itu. Balasannya ditangkap dengan `StringWriter` (Python: `io.StringIO`). Karena `converse` pada Soal 16.3 menerima *reader* dan *writer* alih-alih soket, tidak ada jaringan yang terlibat sama sekali, dan setiap uji hanya memakan waktu mikrodetik.



Gambar 16.5 Uji silang melalui antarmuka. Anggota klien menulis uji untuk kode server; anggota server menulis uji untuk kode klien. Masing-masing memakai *stub* dari sisinya sendiri.

Penyelesaian Java. **Kode 16.11** memuat kedua kelompok uji, serta sebuah *mutan*: klien dengan satu kekeliruan yang disengaja, dipakai untuk memeriksa apakah uji-ujinya cukup kuat.

Kode 16.11 `ProtocolContractTest.java`: uji yang ditulis setiap anggota untuk anggota lain, dengan *stub* yang memutar ulang baris tetap, dan klien mutan yang diloloskan oleh uji yang lemah.

```java
// ProtocolContractTest.java - Soal 16.4: setiap pihak menguji pihak lain melalui protokol, dengan stub.
import java.io.BufferedReader;
import java.io.IOException;
import java.io.PrintWriter;
import java.io.StringReader;
import java.io.StringWriter;
import java.util.ArrayList;
import java.util.List;
import java.util.function.Function;

public class ProtocolContractTest {

    /** Stub untuk pihak mana pun: sekadar memutar ulang baris tetap, apa pun kata pihak lain. */
    static BufferedReader replay(String... lines) {
        return new BufferedReader(new StringReader(String.join("\n", lines) + "\n"));
    }

    /** Anggota KLIEN menguji SERVER: baris klien tetap masuk, baris server keluar. */
    static List<String> serverSays(String... clientLines) throws IOException {
        StringWriter sent = new StringWriter();
        boolean done = KnockKnockServer.converse(replay(clientLines), new PrintWriter(sent, true),
                                                 new KnockKnockProtocol(JokeStore.classics()));
        List<String> lines = new ArrayList<>(sent.toString().lines().toList());
        lines.add(done ? "(finished)" : "(client left)");
        return lines;
    }

    /** Anggota SERVER menguji KLIEN: baris server tetap masuk, baris klien keluar. */
    static List<String> clientSays(Function<String, String> user, String... serverLines)
            throws IOException {
        StringWriter sent = new StringWriter();
        KnockKnockClient.converse(replay(serverLines), new PrintWriter(sent, true), user);
        return sent.toString().lines().toList();
    }

    /** Klien mutan: hampir benar, tetapi mengucapkan "Who is there?". */
    static final class SloppyUser implements Function<String, String> {
        private final KnockKnockClient.PoliteUser polite = new KnockKnockClient.PoliteUser(1);

        @Override
        public String apply(String line) {
            String reply = polite.apply(line);
            return reply.equals("Who's there?") ? "Who is there?" : reply;
        }
    }
```

Kode 16.11 (Lanjutan) *ProtocolContractTest.java: uji yang ditulis setiap anggota untuk anggota lain, dengan stub yang memutar ulang baris tetap, dan klien mutan yang diloloskan oleh uji yang lemah.*

```
    /** Menggoda tetapi keliru: uji yang hanya memeriksa bahwa klien mengatakan sesuatu. */
    static boolean weakTest(Function<String, String> user) throws IOException {
        return !clientSays(user, "Knock! Knock!", "Bye.").isEmpty();
    }

    private static int passed = 0;

    private static void check(String name, Object expected, Object actual) {
        if (!java.util.Objects.equals(expected, actual)) {
            throw new AssertionError(name + ": expected " + expected + " but got " +
actual);
        }
        passed++;
        System.out.println("PASS " + name);
    }

    public static void main(String[] args) throws IOException {
        // Uji yang ditulis anggota klien untuk anggota server.
        check("server speaks first", "Knock! Knock!", serverSays("Who's there?").get(0));
        check("one whole joke", List.of("Knock! Knock!", "Lettuce",
                "Lettuce in, it's cold out here! Want another? (y/n)", "Bye.",
"(finished)"),
                serverSays("Who's there?", "Lettuce who?", "n"));
        check("case does not matter", "Lettuce", serverSays("WHO'S THERE?").get(1));
        check("client vanishes", "(client left)", serverSays("Who's there?").get(2));

        // Uji yang ditulis anggota server untuk anggota klien.
        String[] oneJoke = {"Knock! Knock!", "Lettuce",
                            "Lettuce in, it's cold out here! Want another? (y/n)", "Bye."};
        List<String> expected = List.of("Who's there?", "Lettuce who?", "n");
        check("client follows the protocol", expected,
              clientSays(new KnockKnockClient.PoliteUser(1), oneJoke));
        check("client stops at Bye.", List.of("Who's there?"),
              clientSays(new KnockKnockClient.PoliteUser(1), "Knock! Knock!", "Bye.",
"Knock! Knock!"));
        check("client survives a hang-up", List.of("Who's there?"),
              clientSays(new KnockKnockClient.PoliteUser(1), "Knock! Knock!"));

        // Mutan lolos dari uji yang lemah dan gagal pada uji yang kuat.
        System.out.println("weak test passes the sloppy client: " + weakTest(new
SloppyUser()));
        check("strong test catches the sloppy client", false,
              expected.equals(clientSays(new SloppyUser(), oneJoke)));
        System.out.println("All " + passed + " checks passed.");
    }
}
```

```
PASS server speaks first
PASS one whole joke
PASS case does not matter
PASS client vanishes
PASS client follows the protocol
PASS client stops at Bye.
PASS client survives a hang-up
weak test passes the sloppy client: true
PASS strong test catches the sloppy client
All 8 checks passed.
```

Penyelesaian Python. **Kode 16.12** menyusun uji yang sama dalam dua kelas `unittest`, satu untuk setiap anggota yang menulisnya, sehingga laporannya memperlihatkan sekilas siapa menjanjikan apa.

Kode 16.12 `protocol_contract_test.py`: uji silang dalam Python, satu kelas uji per anggota.

```
# protocol_contract_test.py - Soal 16.4: setiap pihak menguji pihak lain melalui protokol,
dengan stub.
import io
import sys
import unittest
from joke_store import classics
from knock_knock_protocol import KnockKnockProtocol
import knock_knock_server
import knock_knock_client
from knock_knock_client import PoliteUser


def replay(*lines):
    """Stub untuk pihak mana pun: sekadar memutar ulang baris tetap, apa pun kata pihak
lain."""
    return io.StringIO("\n".join(lines) + "\n")


def server_says(*client_lines):
    """Anggota KLIEN menguji SERVER: baris klien tetap masuk, baris server keluar."""
    sent = io.StringIO()
    done = knock_knock_server.converse(replay(*client_lines), sent,
KnockKnockProtocol(classics()))
    return sent.getvalue().splitlines() + ["(finished)" if done else "(client left)"]


def client_says(user, *server_lines):
    """Anggota SERVER menguji KLIEN: baris server tetap masuk, baris klien keluar."""
    sent = io.StringIO()
    knock_knock_client.converse(replay(*server_lines), sent, user)
    return sent.getvalue().splitlines()


class SloppyUser(PoliteUser):
    """Klien mutan: hampir benar, tetapi mengucapkan "Who is there?"."""

    def __call__(self, line):
        reply = super().__call__(line)
        return "Who is there?" if reply == "Who's there?" else reply


def weak_test(user):
    """Menggoda tetapi keliru: uji yang hanya memeriksa bahwa klien mengatakan sesuatu."""
    return len(client_says(user, "Knock! Knock!", "Bye.")) > 0


ONE_JOKE = ("Knock! Knock!", "Lettuce", "Lettuce in, it's cold out here! Want another?
(y/n)", "Bye.")
EXPECTED = ["Who's there?", "Lettuce who?", "n"]
```

Kode 16.12 (Lanjutan) *protocol_contract_test.py: uji silang dalam Python, satu kelas uji per anggota.*

```
class ClientMemberTestsTheServer(unittest.TestCase):
    def test_server_speaks_first(self):
        self.assertEqual("Knock! Knock!", server_says("Who's there?")[0])

    def test_one_whole_joke(self):
        self.assertEqual(["Knock! Knock!", "Lettuce",
                          "Lettuce in, it's cold out here! Want another? (y/n)", "Bye.",
"(finished)"],
                         server_says("Who's there?", "Lettuce who?", "n"))

    def test_case_does_not_matter(self):
        self.assertEqual("Lettuce", server_says("WHO'S THERE?")[1])

    def test_client_vanishes(self):
        self.assertEqual("(client left)", server_says("Who's there?")[2])


class ServerMemberTestsTheClient(unittest.TestCase):
    def test_client_follows_the_protocol(self):
        self.assertEqual(EXPECTED, client_says(PoliteUser(1), *ONE_JOKE))

    def test_client_stops_at_bye(self):
        self.assertEqual(["Who's there?"],
                         client_says(PoliteUser(1), "Knock! Knock!", "Bye.", "Knock!
Knock!"))

    def test_client_survives_a_hang_up(self):
        self.assertEqual(["Who's there?"], client_says(PoliteUser(1), "Knock! Knock!"))

    def test_strong_test_catches_the_sloppy_client(self):
        self.assertNotEqual(EXPECTED, client_says(SloppyUser(1), *ONE_JOKE))


if __name__ == "__main__":
    print("weak test passes the sloppy client:", weak_test(SloppyUser(1)))
    sys.stdout.flush()
    unittest.main(verbosity=2)
```

```
weak test passes the sloppy client: True
test_case_does_not_matter (__main__.ClientMemberTestsTheServer.test_case_does_not_matter)
... ok
test_client_vanishes (__main__.ClientMemberTestsTheServer.test_client_vanishes) ... ok
test_one_whole_joke (__main__.ClientMemberTestsTheServer.test_one_whole_joke) ... ok
test_server_speaks_first (__main__.ClientMemberTestsTheServer.test_server_speaks_first) ...
ok
test_client_follows_the_protocol
(__main__.ServerMemberTestsTheClient.test_client_follows_the_protocol) ... ok
test_client_stops_at_bye (__main__.ServerMemberTestsTheClient.test_client_stops_at_bye) ...
ok
test_client_survives_a_hang_up
(__main__.ServerMemberTestsTheClient.test_client_survives_a_hang_up) ... ok
test_strong_test_catches_the_sloppy_client
(__main__.ServerMemberTestsTheClient.test_strong_test_catches_the_sloppy_client) ... ok

----------------------------------------------------------------------
Ran 8 tests in 0.000s

OK
```

Baris demi baris. `replay` menggabungkan baris-baris dengan karakter baris baru, menambahkan satu baris baru di akhir, lalu membungkus teksnya dalam `StringReader` di dalam `BufferedReader`; bagi kode yang diuji, hal ini tidak dapat dibedakan dari *stream* masukan sebuah soket. `serverSays` menjalankan `KnockKnockServer.converse` yang sungguhan dengan *stub*

sebagai masukannya dan mengembalikan baris-baris server ditambah sebuah penanda, `(finished)` atau `(client left)`, yang diambil dari nilai kembalian `converse`. Keempat uji anggota klien memeriksa bahwa server berbicara lebih dulu, satu lelucon utuh, bahwa huruf besar-kecil tidak berpengaruh, dan bahwa klien yang berhenti setelah satu baris disadari. `clientSays` melakukan kebalikannya untuk anggota server: *stub* server "mengucapkan" baris-baris tetap, dan uji memeriksa apa yang ditulis klien. Ketiga uji anggota server mencakup lelucon yang normal, klien yang harus berhenti pada `Bye.` walaupun server masih berbicara, dan server yang memutus sambungan tanpa mengucapkan `Bye.`. Dua yang terakhir itu persis kasus-kasus yang paling kecil kemungkinannya terpikirkan oleh orang yang menulis klien, dan itulah alasan untuk membiarkan pihak lain yang menulis ujinya.

Penyelesaian keliru yang menggoda. `weakTest` hanya memeriksa bahwa klien mengucapkan *sesuatu*. Uji semacam ini biasanya ditulis terburu-buru agar sebuah laporan dapat menyebut "sudah diuji", dan uji itu meloloskan mutan `SloppyUser`, yang mengucapkan "Who is there?", balasan yang ditolak server sungguhan. Uji yang kuat membandingkan baris-baris klien dengan baris-baris yang diharapkan secara persis dan menggagalkan mutan itu. Merusak kode dengan sengaja lalu memeriksa apakah ada uji yang gagal disebut *mutation testing* (pengujian mutasi); itulah cara tercepat untuk mengetahui apakah uji-uji Anda sungguh menguji sesuatu [2].

> **UJI DIRI.** Tuliskan hasil yang diharapkan dari `serverSays("Who's there?", "Cabbage who?")` tanpa menjalankannya.
>
> *Jawaban.* `[Knock! Knock!, Lettuce, You're supposed to say "Lettuce who?"! Try again. Knock! Knock!, (client left)]`. Server menyapa, memberikan petunjuk, menolak petunjuk yang salah dan mengetuk lagi, lalu masukan yang diputar ulang habis, sehingga `converse` mengembalikan `false`.

16.5.3 Soal 16.5 — Banyak klien sekaligus: uji *multithread* dan terdistribusi

> **SOAL.** Develop and use a test plan that includes multithreaded and distributed testing. Start many clients at the same moment against one running server and check that every client receives exactly the conversation it should, and that the server counted every conversation. Show a server design that passes the single-client tests but fails this one.
>
> *Dengan kata lain:* susun dan pakailah rencana uji yang mencakup pengujian *multithread* dan terdistribusi. Jalankan banyak klien pada saat yang sama terhadap satu server yang sedang berjalan, lalu periksa bahwa setiap klien menerima persis percakapan yang semestinya dan bahwa server mencacah setiap percakapan. Tunjukkan sebuah rancangan server yang lolos uji satu klien tetapi gagal pada uji ini.

Dengan bahasa kita sendiri. Uji pada Soal 16.4 memeriksa setiap sisi secara tersendiri. Sekarang kita memeriksa sistem yang berjalan dalam kondisi yang menjadi alasan ia dibangun: banyak klien bercakap bersamaan melalui soket sungguhan. Dua pertanyaan membuat uji semacam itu dapat dipercaya. Apa jawaban yang benar untuk setiap klien? Dan bagaimana kita membuat klien-klien benar-benar tumpang-tindih alih-alih berjalan satu demi satu?

Rancangan. Jawaban yang benar berasal dari sebuah *orakel*, yaitu cara kedua yang independen untuk menghitung hasil yang diharapkan: untuk klien yang menginginkan n lelucon, kita menjalankan protokol baru dan sebuah `PoliteUser` bersama-sama *tanpa* jaringan, lalu mencatat transkripnya. Tumpang-tindih diperoleh dari sebuah *gerbang mulai*: setiap *thread* klien mula-mula menunggu pada `CountDownLatch` bernilai 1 (Python: sebuah `threading.Event`), dan uji baru membuka gerbang itu setelah kedua puluh tugas diserahkan, sehingga semuanya mulai dalam selisih beberapa mikrodetik. Setiap klien juga memiliki batas waktu dan batas 30 giliran, karena

server yang rusak harus menghasilkan uji yang gagal, bukan uji yang menggantung selamanya. Terakhir, server memperbarui pencacahnya sesaat *setelah* klien membaca `Bye.`, sehingga uji menunggu nilai cacahan itu dengan tenggat dua detik alih-alih langsung memeriksanya.

Penyelesaian Java. **Kode 16.13** menjalankan uji terhadap server yang benar dan terhadap sebuah "optimasi" yang menggoda: server yang membuat satu objek protokol dan memakainya bersama untuk semua koneksi.

Kode 16.13 `ConcurrentClientsTest.java`: dua puluh klien yang dilepas bersama oleh gerbang mulai, diperiksa terhadap orakel, dan server yang memakai satu objek protokol bersama.

```
// ConcurrentClientsTest.java - Soal 16.5: banyak klien sekaligus, dan server yang terlalu
banyak berbagi.
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.net.ServerSocket;
import java.net.Socket;
import java.nio.charset.StandardCharsets;
import java.util.ArrayList;
import java.util.List;
import java.util.concurrent.CountDownLatch;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.Future;
import java.util.function.Function;

public class ConcurrentClientsTest {

    static final int CLIENTS = 20;

    /** Orakel: transkrip yang HARUS dilihat klien yang menginginkan n lelucon, dihitung
tanpa jaringan. */
    static List<String> expected(int n) {
        KnockKnockProtocol p = new KnockKnockProtocol(JokeStore.classics());
        Function<String, String> user = new KnockKnockClient.PoliteUser(n);
        List<String> t = new ArrayList<>();
        String s = p.processInput(null);
        while (true) {
            t.add("S: " + s);
            if (s.equals("Bye.")) {
                return t;
            }
            String c = user.apply(s);
            t.add("C: " + c);
            s = p.processInput(c);
        }
    }
```

Kode 16.13 (Lanjutan) *ConcurrentClientsTest.java: dua puluh klien yang dilepas bersama oleh gerbang mulai, diperiksa terhadap orakel, dan server yang memakai satu objek protokol bersama.*

```
    /** Klien dengan batas waktu dan kesabaran terbatas, agar server yang rusak tidak dapat
menggantung uji. */
    static List<String> talkWithin(int port, int n) throws IOException {
        Function<String, String> polite = new KnockKnockClient.PoliteUser(n);
        int[] turns = {0};
        Function<String, String> patient = line -> (++turns[0] > 30) ? null :
polite.apply(line);
        try (Socket s = new Socket("localhost", port);
             BufferedReader in = new BufferedReader(new InputStreamReader(
                     s.getInputStream(), StandardCharsets.UTF_8));
             PrintWriter out = new PrintWriter(new OutputStreamWriter(
                     s.getOutputStream(), StandardCharsets.UTF_8), true)) {
            s.setSoTimeout(2000);
            return KnockKnockClient.converse(in, out, patient);
        } catch (IOException e) {
            return List.of("(" + e.getClass().getSimpleName() + ")");
        }
    }

    /** Memulai CLIENTS klien pada saat yang sama; mengembalikan berapa yang melihat
transkrip yang benar. */
    static int correctConversations(int port) throws Exception {
        ExecutorService pool = Executors.newFixedThreadPool(CLIENTS);
        CountDownLatch startGate = new CountDownLatch(1);
        List<Future<Boolean>> results = new ArrayList<>();
        for (int i = 0; i < CLIENTS; i++) {
            int n = i % 3 + 1;                              // 1, 2, atau 3 lelucon
            results.add(pool.submit(() -> {
                startGate.await();                          // semua menunggu di sini ...
                return talkWithin(port, n).equals(expected(n));
            }));
        }
        startGate.countDown();                              // ... lalu mulai bersama-sama
        int correct = 0;
        for (Future<Boolean> f : results) {
            if (f.get()) {
                correct++;
            }
        }
        pool.shutdown();
        return correct;
    }
```

Kode 16.13 (Lanjutan) *ConcurrentClientsTest.java: dua puluh klien yang dilepas bersama oleh gerbang mulai, diperiksa terhadap orakel, dan server yang memakai satu objek protokol bersama.*

```
    /** Menggoda tetapi keliru: SATU objek protokol dipakai bersama oleh setiap koneksi. */
    static int startSharedProtocolServer() throws IOException {
        ServerSocket listener = new ServerSocket(0);
        KnockKnockProtocol shared = new KnockKnockProtocol(JokeStore.classics());
        Thread acceptor = new Thread(() -> {
            try {
                while (true) {
                    Socket c = listener.accept();
                    new Thread(() -> {
                        try (c;
                             BufferedReader in = new BufferedReader(new
InputStreamReader(c.getInputStream()));
                             PrintWriter out = new PrintWriter(c.getOutputStream(), true))
{
                            KnockKnockServer.converse(in, out, shared);
                        } catch (IOException | RuntimeException e) {
                            // percakapan rusak; soket ditutup oleh try
                        }
                    }).start();
                }
            } catch (IOException e) {
                // listener ditutup
            }
        });
        acceptor.setDaemon(true);
        acceptor.start();
        return listener.getLocalPort();
    }

    static String firstLine(Socket s) throws IOException {
        return new BufferedReader(new InputStreamReader(s.getInputStream())).readLine();
    }

    private static int passed = 0;

    private static void check(String name, Object expected, Object actual) {
        if (!java.util.Objects.equals(expected, actual)) {
            throw new AssertionError(name + ": expected " + expected + " but got " +
actual);
        }
        passed++;
        System.out.println("PASS " + name);
    }
```

Kode 16.13 (Lanjutan) *ConcurrentClientsTest.java: dua puluh klien yang dilepas bersama oleh gerbang mulai, diperiksa terhadap orakel, dan server yang memakai satu objek protokol bersama.*

```
    public static void main(String[] args) throws Exception {
        KnockKnockServer server = new KnockKnockServer(JokeStore.classics());
        int port = server.start(0);
        int good = correctConversations(port);
        System.out.println("correct server: " + good + " of " + CLIENTS + " conversations
right");
        long deadline = System.currentTimeMillis() + 2000;  // server mencacah sesaat
kemudian
        while (server.finishedCount() < CLIENTS && System.currentTimeMillis() < deadline) {
            Thread.sleep(10);
        }
        check("every client got its own conversation", CLIENTS, good);
        check("server counted every conversation", CLIENTS, server.finishedCount());
        server.stop();

        int sharedPort = startSharedProtocolServer();
        try (Socket a = new Socket("localhost", sharedPort)) {
            System.out.println("shared protocol, client A hears: " + firstLine(a));
            try (Socket b = new Socket("localhost", sharedPort)) {     // B datang setelah
A
                System.out.println("shared protocol, client B hears: " + firstLine(b));
            }
        }
        int bad = correctConversations(sharedPort);
        System.out.println("shared protocol: all " + CLIENTS + " right? " + (bad ==
CLIENTS));
        check("sharing the protocol breaks conversations", true, bad < CLIENTS);
        System.out.println("All " + passed + " checks passed.");
    }
}
```

```
correct server: 20 of 20 conversations right
PASS every client got its own conversation
PASS server counted every conversation
shared protocol, client A hears: Knock! Knock!
shared protocol, client B hears: You're supposed to say "Who's there?"! Try again. Knock!
Knock!
shared protocol: all 20 right? false
PASS sharing the protocol breaks conversations
All 3 checks passed.
```

Penyelesaian Python. **Kode 16.14** memakai `ThreadPoolExecutor` untuk klien-klien dan `threading.Event` sebagai gerbang mulai.

Kode 16.14 `concurrent_clients_test.py`: uji terdistribusi yang sama dalam Python.

```
# concurrent_clients_test.py - Soal 16.5: banyak klien sekaligus, dan server yang terlalu
banyak berbagi.
import socket
import sys
import threading
import time
import unittest
from concurrent.futures import ThreadPoolExecutor
from joke_store import classics
from knock_knock_protocol import KnockKnockProtocol
from knock_knock_server import KnockKnockServer, converse as server_converse
from knock_knock_client import PoliteUser, converse as client_converse

CLIENTS = 20


def expected(n):
    """Orakel: transkrip yang HARUS dilihat klien yang menginginkan n lelucon, dihitung
tanpa jaringan."""
    p, user, t = KnockKnockProtocol(classics()), PoliteUser(n), []
    s = p.process_input(None)
    while True:
        t.append("S: " + s)
        if s == "Bye.":
            return t
        c = user(s)
        t.append("C: " + c)
        s = p.process_input(c)


def talk_within(port, n):
    """Klien dengan batas waktu dan kesabaran terbatas, agar server yang rusak tidak dapat
menggantung uji."""
    polite, turns = PoliteUser(n), [0]

    def patient(line):
        turns[0] += 1
        return None if turns[0] > 30 else polite(line)

    try:
        with socket.create_connection(("localhost", port), timeout=2) as s, \
                s.makefile("r", encoding="utf-8") as reader, \
                s.makefile("w", encoding="utf-8") as writer:
            return client_converse(reader, writer, patient)
    except OSError as e:
        return [f"({type(e).__name__})"]
```

Kode 16.14 (Lanjutan) *concurrent_clients_test.py: uji terdistribusi yang sama dalam Python.*

```
def correct_conversations(port):
    """Memulai CLIENTS klien pada saat yang sama; mengembalikan berapa yang melihat
transkrip yang benar."""
    start_gate = threading.Event()

    def one_client(n):
        start_gate.wait()                               # semua menunggu di sini ...
        return talk_within(port, n) == expected(n)

    with ThreadPoolExecutor(max_workers=CLIENTS) as pool:
        futures = [pool.submit(one_client, i % 3 + 1) for i in range(CLIENTS)]
        start_gate.set()                                # ... lalu mulai bersama-sama
        return sum(1 for f in futures if f.result())


def start_shared_protocol_server():
    """Menggoda tetapi keliru: SATU objek protokol dipakai bersama oleh setiap koneksi."""
    listener = socket.create_server(("localhost", 0))
    shared = KnockKnockProtocol(classics())

    def serve(c):
        try:
            with c, c.makefile("r") as reader, c.makefile("w") as writer:
                server_converse(reader, writer, shared)
        except (OSError, RuntimeError):
            pass                                        # percakapan rusak

    def accept_loop():
        while True:
            c, _ = listener.accept()
            threading.Thread(target=serve, args=(c,), daemon=True).start()

    threading.Thread(target=accept_loop, daemon=True).start()
    return listener.getsockname()[1]


def first_line(s):
    return s.makefile("r").readline().rstrip("\n")
```

Kode 16.14 (Lanjutan) *concurrent_clients_test.py: uji terdistribusi yang sama dalam Python.*

```
class TestConcurrentClients(unittest.TestCase):
    def test_every_client_gets_its_own_conversation(self):
        server = KnockKnockServer(classics())
        good = correct_conversations(server.start(0))
        deadline = time.monotonic() + 2                 # server mencacah sesaat kemudian
        while server.counts()[0] < CLIENTS and time.monotonic() < deadline:
            time.sleep(0.01)
        server.stop()
        self.assertEqual(CLIENTS, good)
        self.assertEqual(CLIENTS, server.counts()[0])

    def test_sharing_the_protocol_breaks_conversations(self):
        self.assertLess(correct_conversations(start_shared_protocol_server()), CLIENTS)


if __name__ == "__main__":
    port = start_shared_protocol_server()
    with socket.create_connection(("localhost", port)) as a:
        print("shared protocol, client A hears:", first_line(a))
        with socket.create_connection(("localhost", port)) as b:     # B datang setelah A
            print("shared protocol, client B hears:", first_line(b))
    sys.stdout.flush()
    unittest.main(verbosity=2)
```

```
shared protocol, client A hears: Knock! Knock!
shared protocol, client B hears: You're supposed to say "Who's there?"! Try again. Knock!
Knock!
test_every_client_gets_its_own_conversation
(__main__.TestConcurrentClients.test_every_client_gets_its_own_conversation) ... ok
test_sharing_the_protocol_breaks_conversations
(__main__.TestConcurrentClients.test_sharing_the_protocol_breaks_conversations) ... ok

----------------------------------------------------------------------
Ran 2 tests in 0.059s

OK
```

Baris demi baris. `expected(n)` adalah orakelnya: ia memanggil `processInput` dan `PoliteUser` berselang-seling sampai `Bye.`. `talkWithin` membungkus pengguna dalam fungsi kedua, `patient`, yang mencacah giliran dalam larik satu elemen (sebuah *lambda* tidak boleh menugasi variabel lokal biasa, tetapi boleh mengubah elemen larik) dan menyerah setelah 30. `correctConversations` menyerahkan dua puluh tugas ke kumpulan *thread* berukuran tetap; masing-masing memanggil `startGate.await()` dan menunggu di situ. Panggilan `startGate.countDown()` menurunkan cacahan dari 1 ke 0 dan melepaskan semuanya bersama-sama. Setiap `Future<Boolean>` menyimpan putusan satu klien, dan `f.get()` menunggunya. Gelung setelah `correctConversations` di `main` memeriksa `finishedCount()` setiap 10 ms selama paling lama dua detik: beginilah cara menguji sesuatu yang *pada akhirnya* menjadi benar, keadaan yang lazim dalam sistem terdistribusi.

Server yang keliru biasa saja kecuali pada satu baris: `KnockKnockProtocol shared = new KnockKnockProtocol(...)` terletak *di luar* gelung `accept`. Demonstrasi sebelum uji beban sepenuhnya deterministik. Klien A menyambung dan membaca baris pertamanya: protokol bersama berpindah dari `WAITING` ke `SENT_KNOCK` dan A mendengar “Knock! Knock!”. Baru sesudah itu klien B menyambung; server memanggil `processInput(null)` pada objek yang sama, yang kini berada dalam keadaan `SENT_KNOCK`, dan baris paling pertama yang diterima B adalah “You’re supposed to say …”. Dalam uji beban, percakapan-percakapan berselang-seling secara sembarang, dan tidak semua dari dua puluh percakapan itu benar. Program Python mengulang demonstrasi itu sebelum menjalankan ujinya.

Penyelesaian keliru yang menggoda. Memakai bersama objek protokol terasa seperti menghemat memori, dan uji dengan satu klien tidak dapat membedakannya. *Bug*-nya sebenarnya bukan tentang *thread*: kedua klien pada demonstrasi itu dilayani satu demi satu dan tetap saling mengganggu, karena keadaan sebuah percakapan tinggal di dalam objek yang dipakai dua percakapan. Kaidah dari Bab 14 layak dinyatakan ulang dalam bentuk ini: *keadaan per koneksi adalah milik koneksi itu*; apa yang dipakai bersama harus imutabel (penyimpan lelucon) atau dilindungi (pencacah).

> **UJI DIRI.** Seseorang mengusulkan untuk memperbaiki server berprotokol-bersama dengan mendeklarasikan `processInput` sebagai `synchronized`. Apakah uji beban akan lolos?
>
> *Jawaban.* Tidak. `synchronized` membuat setiap panggilan berjalan sendirian, sehingga *field* tidak dapat lagi rusak di tengah panggilan, tetapi panggilan dari klien-klien yang berbeda tetap berselang-seling *di antara* panggilan: ketukan A yang disusul sapaan B adalah persis kegagalan deterministik tadi, dan sama sekali tidak melibatkan panggilan yang serentak. Keamanan *thread* pada setiap metode tidak membuat percakapan bersama menjadi benar; hanya satu objek protokol per koneksi yang dapat melakukannya.

16.6 Soal-Soal tentang Ketangguhan, Antarmuka Grafis, dan Dokumentasi

16.6.1 Soal 16.6 — Ketika server atau klien mati

> **SOAL.** Take care of failures on both sides. When the server is down, the client should know it; when a client is down, the server should know it. Handle connection errors as exceptions, both when the server is down and when a client is down, and show that the server keeps serving other clients after one of them has vanished.
>
> *Dengan kata lain:* tangani kegagalan di kedua sisi. Ketika server mati, klien harus mengetahuinya; ketika sebuah klien mati, server harus mengetahuinya. Tangani galat koneksi sebagai eksepsi, baik ketika server mati maupun ketika klien mati, dan tunjukkan bahwa server tetap melayani klien lain setelah salah satunya menghilang.

Dengan bahasa kita sendiri. Program jaringan menghadapi kegagalan yang tidak pernah dialami program tunggal, dan masing-masing tampak berbeda dari dalam. Kita akan mendaftarkannya, memutuskan apa yang diamati setiap sisi dalam setiap kasus, dan menulis klien yang satu-satunya cara berakhirnya adalah sebuah kalimat yang jelas bagi pengguna, tidak pernah berupa jejak tumpukan (*stack trace*).

Rancangan. **Tabel 16.6** adalah rancangannya. Kolom-kolom tengahnya adalah bagian yang memerlukan pengetahuan tentang soket: apa yang sebenarnya *diamati* program, yang berbeda di Java dan di Python.

Tabel 16.6 Apa yang diamati setiap sisi ketika sisi lain gagal, di Java dan di Python, dan bagaimana Knock-Knock Club bereaksi.

Situasi	Yang diamati Java	Yang diamati Python	Reaksi kita
server tidak berjalan	`ConnectException` dari `new Socket` atau `connect`	`ConnectionRefusedError`	"Cannot reach the server: is it running?"
server memutus di tengah percakapan	`readLine()` mengembalikan `null`	`readline()` mengembalikan `""`	"The server closed the connection after *k* line(s)."
server hidup tetapi diam	`SocketTimeoutException` setelah `setSoTimeout`	`TimeoutError` setelah `settimeout`	"The server did not answer within 500 ms."
klien pergi dengan wajar atau *crash*	`readLine()` server mengembalikan `null`	`readline()` server mengembalikan `""`	catat "client left before the end", tetap melayani
koneksi di-*reset* (data belum terbaca, kabel dicabut)	`SocketException: Connection reset`	`ConnectionResetError`	catat "connection lost", tetap melayani

Dua kaidah rancangan menyusul. Pertama, *setiap panggilan yang menahan mendapat batas waktu*: `connect` dengan batas waktu dan `setSoTimeout` untuk pembacaan di Java, `create_connection(..., timeout=...)` di Python. Tanpa batas, server yang diam membekukan klien selamanya, dan tidak pernah ada eksepsi yang dilempar untuk ditangani. Kedua, *akhir stream adalah hasil yang wajar, bukan galat*: `null` di Java dan `""` di Python harus diperiksa pada setiap pembacaan.

Penyelesaian Java. **Kode 16.15** memuat klien yang aman, klien yang naif, dua metode bantu yang sengaja menimbulkan kegagalan (port yang tidak didengarkan siapa pun, dan server yang berulah), serta sebuah skenario ketika sebuah klien pergi di tengah jalan dan klien kedua dilayani secara normal.

Kode 16.15 `FailureDetection.java`: klien yang mengubah setiap kegagalan menjadi satu kalimat, klien yang naif, server-server yang sengaja berulah, dan server yang menyadari klien yang telah pergi.

```
// FailureDetection.java - Soal 16.6: klien menyadari server yang mati, server menyadari
klien yang mati.
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.net.ConnectException;
import java.net.InetSocketAddress;
import java.net.ServerSocket;
import java.net.Socket;
import java.net.SocketTimeoutException;
import java.nio.charset.StandardCharsets;
import java.util.List;
import java.util.function.Function;

public class FailureDetection {

    /** Pintu masuk klien yang aman: setiap kegagalan jaringan menjadi satu kalimat bagi
pengguna. */
    static String safeTalk(int port, int limitMillis, Function<String, String> user) {
        try (Socket s = new Socket()) {
            s.connect(new InetSocketAddress("localhost", port), limitMillis);  //
menyambung, dengan batas waktu
            s.setSoTimeout(limitMillis);                                       // setiap
pembacaan, dengan batas waktu
            BufferedReader in = new BufferedReader(new InputStreamReader(
                    s.getInputStream(), StandardCharsets.UTF_8));
            PrintWriter out = new PrintWriter(new OutputStreamWriter(
                    s.getOutputStream(), StandardCharsets.UTF_8), true);
            List<String> t = KnockKnockClient.converse(in, out, user);
            if (t.isEmpty() || !t.get(t.size() - 1).equals("S: Bye.")) {
                return "The server closed the connection after " + t.size() + " line(s).";
            }
            return "OK, " + t.size() + " lines.";
        } catch (ConnectException e) {
            return "Cannot reach the server: is it running?";
        } catch (SocketTimeoutException e) {
            return "The server did not answer within " + limitMillis + " ms.";
        } catch (IOException e) {
            return "Connection error: " + e.getClass().getSimpleName();
        }
    }
```

Kode 16.15 (Lanjutan) *FailureDetection.java: klien yang mengubah setiap kegagalan menjadi satu kalimat, klien yang naif, server-server yang sengaja berulah, dan server yang menyadari klien yang telah pergi.*

```
    /** Menggoda tetapi keliru: klien yang menganggap server selalu ada. */
    static void naiveTalk(int port) throws IOException {
        Socket s = new Socket("localhost", port);
        BufferedReader in = new BufferedReader(new InputStreamReader(s.getInputStream()));
        PrintWriter out = new PrintWriter(s.getOutputStream(), true);
        Function<String, String> user = new KnockKnockClient.PoliteUser(1);
        while (true) {
            String line = in.readLine();
            if (line.equals("Bye.")) {                      // line bernilai null setelah
sambungan diputus
                break;
            }
            out.println(user.apply(line));
        }
        s.close();
    }

    /** Port yang tidak didengarkan siapa pun: kita membukanya, mencatat nomornya, lalu
menutupnya lagi. */
    static int portWithNobody() throws IOException {
        try (ServerSocket probe = new ServerSocket(0)) {
            return probe.getLocalPort();
        }
    }

    /** Server yang berulah: mengucapkan barisnya, mendengar satu balasan, lalu memutus
(atau diam saja). */
    static int badServer(List<String> lines, boolean hangUp) throws IOException {
        ServerSocket listener = new ServerSocket(0);
        Thread t = new Thread(() -> {
            try (listener; Socket c = listener.accept()) {
                PrintWriter out = new PrintWriter(c.getOutputStream(), true);
                lines.forEach(out::println);
                if (hangUp) {
                    new BufferedReader(new
InputStreamReader(c.getInputStream())).readLine();
                } else {
                    Thread.sleep(5000);                     // hidup, tetapi diam
                }
            } catch (IOException | InterruptedException e) {
                // tidak ada yang perlu dilakukan: server ini memang dibuat berulah
            }
        });
        t.setDaemon(true);
        t.start();
        return listener.getLocalPort();
    }

    private static int passed = 0;
```

Kode 16.15 (Lanjutan) *FailureDetection.java: klien yang mengubah setiap kegagalan menjadi satu kalimat, klien yang naif, server-server yang sengaja berulah, dan server yang menyadari klien yang telah pergi.*

```
    private static void check(String name, Object expected, Object actual) {
        if (!java.util.Objects.equals(expected, actual)) {
            throw new AssertionError(name + ": expected " + expected + " but got " +
actual);
        }
        passed++;
        System.out.println("PASS " + name);
    }

    public static void main(String[] args) throws Exception {
        String down = safeTalk(portWithNobody(), 500, new KnockKnockClient.PoliteUser(1));
        String crash = safeTalk(badServer(List.of("Knock! Knock!"), true), 500,
                                new KnockKnockClient.PoliteUser(1));
        String silent = safeTalk(badServer(List.of(), false), 500, new
KnockKnockClient.PoliteUser(1));
        System.out.println("server down    -> " + down);
        System.out.println("server crashes -> " + crash);
        System.out.println("server silent  -> " + silent);
        try {
            naiveTalk(badServer(List.of("Knock! Knock!"), true));
        } catch (NullPointerException e) {
            System.out.println("naive client   -> NullPointerException");
        }

        KnockKnockServer server = new KnockKnockServer(JokeStore.classics());
        int port = server.start(0);
        try (Socket quitter = new Socket("localhost", port)) {
            new BufferedReader(new InputStreamReader(quitter.getInputStream())).readLine();
        }                                                    // pergi setelah baris pertama
        String next = safeTalk(port, 500, new KnockKnockClient.PoliteUser(1));
        long deadline = System.currentTimeMillis() + 2000;
        while (server.finishedCount() + server.leftEarlyCount() < 2
                && System.currentTimeMillis() < deadline) {
            Thread.sleep(10);
        }
        System.out.println("next client    -> " + next);
        System.out.println("server log     -> " + server.log());
        server.stop();

        check("server down", "Cannot reach the server: is it running?", down);
        check("server crashes", "The server closed the connection after 2 line(s).",
crash);
        check("server silent", "The server did not answer within 500 ms.", silent);
        check("server survives a client that leaves", "OK, 7 lines.", next);
        check("server noticed", List.of("client left before the end"), server.log());
        System.out.println("All " + passed + " checks passed.");
    }
}
```

```
server down    -> Cannot reach the server: is it running?
server crashes -> The server closed the connection after 2 line(s).
server silent  -> The server did not answer within 500 ms.
naive client   -> NullPointerException
next client    -> OK, 7 lines.
server log     -> [client left before the end]
PASS server down
PASS server crashes
PASS server silent
PASS server survives a client that leaves
PASS server noticed
All 5 checks passed.
```

Penyelesaian Python. **Kode 16.16** mengikuti rencana yang sama.

Kode 16.16 `failure_detection.py`: skenario-skenario kegagalan yang sama dalam Python.

```
# failure_detection.py - Soal 16.6: klien menyadari server yang mati, server menyadari
klien yang mati.
import socket
import sys
import threading
import time
import unittest
from joke_store import classics
from knock_knock_server import KnockKnockServer, send
from knock_knock_client import PoliteUser, converse


def safe_talk(port, limit_seconds, user):
    """Pintu masuk klien yang aman: setiap kegagalan jaringan menjadi satu kalimat bagi
pengguna."""
    try:
        with socket.create_connection(("localhost", port), timeout=limit_seconds) as s, \
                s.makefile("r", encoding="utf-8") as reader, \
                s.makefile("w", encoding="utf-8") as writer:
            t = converse(reader, writer, user)          # batas waktu berlaku untuk setiap
pembacaan
    except ConnectionRefusedError:
        return "Cannot reach the server: is it running?"
    except TimeoutError:
        return f"The server did not answer within {int(limit_seconds * 1000)} ms."
    except OSError as e:
        return f"Connection error: {type(e).__name__}"
    if not t or t[-1] != "S: Bye.":
        return f"The server closed the connection after {len(t)} line(s)."
    return f"OK, {len(t)} lines."


def naive_talk(port):
    """Menggoda tetapi keliru: klien yang menganggap server selalu ada."""
    s = socket.create_connection(("localhost", port))
    reader, writer, user = s.makefile("r"), s.makefile("w"), PoliteUser(1)
    while True:
        line = reader.readline().rstrip("\n")           # "" setelah sambungan diputus,
selamanya
        if line == "Bye.":
            break
        send(writer, user(line))                        # menjawab " who?" kepada kesunyian


def port_with_nobody():
    """Port yang tidak didengarkan siapa pun: kita membukanya, mencatat nomornya, lalu
menutupnya lagi."""
    with socket.create_server(("localhost", 0)) as probe:
        return probe.getsockname()[1]
```

Kode 16.16 (Lanjutan) *failure_detection.py: skenario-skenario kegagalan yang sama dalam Python.*

```
def bad_server(lines, hang_up):
    """Server yang berulah: mengucapkan barisnya, mendengar satu balasan, lalu memutus
(atau diam saja)."""
    listener = socket.create_server(("localhost", 0))

    def run():
        with listener, listener.accept()[0] as c, c.makefile("r") as reader,
c.makefile("w") as writer:
            for line in lines:
                send(writer, line)
            if hang_up:
                reader.readline()
            else:
                time.sleep(5)                           # hidup, tetapi diam

    threading.Thread(target=run, daemon=True).start()
    return listener.getsockname()[1]


def client_down_scenario():
    """Seorang klien pergi setelah satu baris; klien kedua tetap harus dilayani."""
    server = KnockKnockServer(classics())
    port = server.start(0)
    with socket.create_connection(("localhost", port)) as quitter:
        quitter.makefile("r").readline()                # pergi setelah baris pertama
    nxt = safe_talk(port, 0.5, PoliteUser(1))
    deadline = time.monotonic() + 2
    while sum(server.counts()[:2]) < 2 and time.monotonic() < deadline:
        time.sleep(0.01)
    server.stop()
    return nxt, server.counts()[2]


class TestFailures(unittest.TestCase):
    def test_server_down(self):
        self.assertEqual("Cannot reach the server: is it running?",
                         safe_talk(port_with_nobody(), 0.5, PoliteUser(1)))

    def test_server_crashes(self):
        self.assertEqual("The server closed the connection after 2 line(s).",
                         safe_talk(bad_server(["Knock! Knock!"], True), 0.5,
PoliteUser(1)))

    def test_server_silent(self):
        self.assertEqual("The server did not answer within 500 ms.",
                         safe_talk(bad_server([], False), 0.5, PoliteUser(1)))
```

Kode 16.16 (Lanjutan) *failure_detection.py: skenario-skenario kegagalan yang sama dalam Python.*

```
    def test_naive_client_crashes(self):
        with self.assertRaises(OSError):                # BrokenPipeError atau
ConnectionResetError
            naive_talk(bad_server(["Knock! Knock!"], True))

    def test_server_survives_a_client_that_leaves(self):
        nxt, log = client_down_scenario()
        self.assertEqual("OK, 7 lines.", nxt)
        self.assertEqual(["client left before the end"], log)


if __name__ == "__main__":
    print("server down    ->", safe_talk(port_with_nobody(), 0.5, PoliteUser(1)))
    print("server crashes ->", safe_talk(bad_server(["Knock! Knock!"], True), 0.5,
PoliteUser(1)))
    print("server silent  ->", safe_talk(bad_server([], False), 0.5, PoliteUser(1)))
    try:
        naive_talk(bad_server(["Knock! Knock!"], True))
    except OSError as e:
        print("naive client   -> crashed with an OSError:", isinstance(e, OSError))
    nxt, log = client_down_scenario()
    print("next client    ->", nxt)
    print("server log     ->", log)
    sys.stdout.flush()
    unittest.main(verbosity=2)
```

```
server down    -> Cannot reach the server: is it running?
server crashes -> The server closed the connection after 2 line(s).
server silent  -> The server did not answer within 500 ms.
naive client   -> crashed with an OSError: True
next client    -> OK, 7 lines.
server log     -> ['client left before the end']
test_naive_client_crashes (__main__.TestFailures.test_naive_client_crashes) ...
/usr/lib/python3.11/socket.py:789: ResourceWarning: unclosed <socket.socket fd=7, family=2,
type=1, proto=6, laddr=('127.0.0.1', 46800)>
  self._sock = None
ResourceWarning: Enable tracemalloc to get the object allocation traceback
ok
test_server_crashes (__main__.TestFailures.test_server_crashes) ... ok
test_server_down (__main__.TestFailures.test_server_down) ... ok
test_server_silent (__main__.TestFailures.test_server_silent) ... ok
test_server_survives_a_client_that_leaves
(__main__.TestFailures.test_server_survives_a_client_that_leaves) ... ok

----------------------------------------------------------------------
Ran 5 tests in 0.521s

OK
```

Baris demi baris. `safeTalk` membuat `Socket` yang belum tersambung lalu memanggil `connect(address, limitMillis)`, sehingga bahkan upaya menyambung pun memiliki batas waktu (`new Socket(host, port)` biasa dapat menunggu lebih dari satu menit untuk mesin yang tidak terjangkau). `setSoTimeout(limitMillis)` kemudian membatasi setiap pembacaan. Ketiga klausa `catch` tersusun dari yang khusus ke yang umum, sebagaimana diwajibkan Java: `ConnectException` dan `SocketTimeoutException` keduanya subkelas `IOException`, dan klausa umum yang ditulis lebih dulu akan membuat klausa-klausa khusus tidak terjangkau, yang merupakan galat kompilasi. Setelah kembali secara normal, transkrip itu sendiri memberi tahu apakah percakapan berakhir dengan semestinya: jika baris terakhirnya bukan `S: Bye.`, server memutus sambungan terlalu dini. `portWithNobody` membuka soket server pada port bebas, mengingat nomornya, lalu menutupnya lagi: upaya menyambung ke port itu akan ditolak.

`badServer` menerima satu klien, mengucapkan baris-barisnya, lalu membaca satu balasan dan memutus sambungan, atau diam selama lima detik. Membaca satu balasan sebelum memutus membuat skenario ini deterministik: server baru menutup koneksi setelah baris klien tiba, sehingga klien melihat akhir *stream* yang bersih.

Skenario terakhir memakai server sungguhan. `quitter` membaca satu baris lalu ditutup oleh blok `try`-nya, yang mengirimkan akhir *stream* kepada server; `readLine()` milik pekerja mengembalikan `null`, `converse` mengembalikan `false`, dan server mencatat "client left before the end". Klien kedua, yang dimulai sesudahnya, menuntaskan percakapan satu lelucon yang normal (7 baris), yang menunjukkan bahwa satu klien yang telah pergi tidak mengganggu klien lain. Catatan server diperiksa agar berisi tepat satu entri.

Di Python, `socket.create_connection((host, port), timeout=0.5)` menerapkan batas waktu pada penyambungan *dan* pada setiap operasi soket selanjutnya. Kelas-kelas eksepsinya berbeda, `ConnectionRefusedError` dan `TimeoutError`, tetapi struktur `safe_talk` sama.

Penyelesaian keliru yang menggoda. `naiveTalk` adalah klien yang pertama kali ditulis kebanyakan orang: `while (true) { String line = in.readLine(); if (line.equals("Bye.")) break; ... }`. Ketika server menghilang, `readLine()` mengembalikan `null`, dan `line.equals(...)` melempar `NullPointerException`, seperti yang diperlihatkan keluarannya. Versi Python gagal dengan cara lain yang lebih berbahaya: `readline()` terus-menerus mengembalikan `""` setelah akhir *stream*, sehingga gelungnya terus menjawab " who?" kepada kesunyian sampai upaya menulis ke soket yang mati memunculkan `BrokenPipeError` atau `ConnectionResetError`. Dalam kedua kasus, pengguna melihat *crash* alih-alih sebuah kalimat, dan di kedua bahasa klien naif itu juga membiarkan soketnya terbuka, yang bahkan dilaporkan `unittest` dalam keluaran Python sebagai `ResourceWarning`. Periksalah akhir *stream* pada setiap pembacaan, dan tutuplah soket dengan `try`-with-resources atau `with`.

> **UJI DIRI.** Kabel jaringan sebuah klien tercabut di tengah lelucon. Komputer klien tidak mengirim apa-apa, bahkan akhir *stream* pun tidak. Apa yang diamati server pada **Kode 16.7**, dan bagaimana Anda membuatnya menyadari hal itu?
>
> *Jawaban.* Tidak ada sama sekali: pekerja menunggu di `readLine()` selamanya, karena tidak ada paket yang datang untuk mengabarkan bahwa koneksi telah lenyap. Berikan juga batas waktu baca pada soket-soket server (misalnya `client.setSoTimeout(60_000)` di `serve`), dan perlakukan `SocketTimeoutException` sebagai "klien hilang". Protokol yang harus mendeteksi kegagalan senyap dengan cepat menambahkan *heartbeat* (detak), yaitu pesan kecil seperti `PING` yang dikirim setiap pihak beberapa detik sekali; detak yang tidak datang berarti pihak lain telah pergi. Latihan 16.7 meminta rancangan ini.

16.6.2 Soal 16.7 — Jendela yang tidak pernah membeku

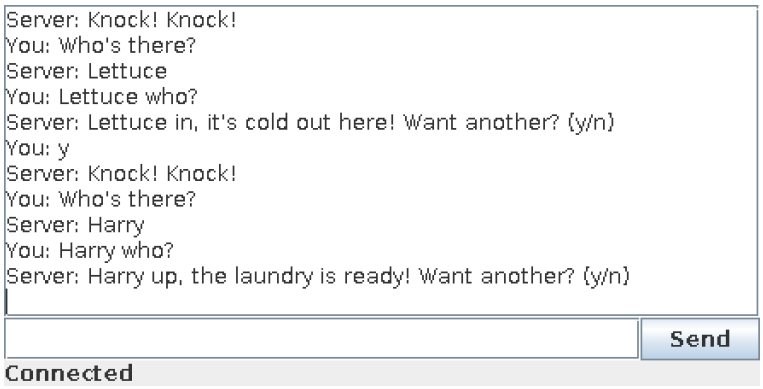
> **SOAL.** Give the client a graphical user interface. The window shows the conversation, lets the user type and send each reply, and shows the connection status, including "server down". The window must stay responsive while the program waits for the network, and the logic must be testable without opening a window.
>
> *Dengan kata lain:* berikan antarmuka grafis kepada klien. Jendelanya menampilkan percakapan, membiarkan pengguna mengetik dan mengirim setiap jawaban, serta menampilkan status koneksi, termasuk "server mati". Jendela itu harus tetap tanggap selagi program menunggu jaringan, dan logikanya harus dapat diuji tanpa membuka jendela.

Dengan bahasa kita sendiri. Bab 10 mengajarkan satu kaidah tentang antarmuka grafis yang menjadi sangat penting bila ada jaringan: *thread* yang menggambar jendela dan menangani klik

(*event dispatch thread* pada Swing, gelung utama pada Tkinter) tidak boleh menunggu apa pun. Pembacaan dari jaringan dapat menunggu berdetik-detik. Jadi jaringan harus tinggal di *thread*-nya sendiri, dan kedua *thread* harus berkomunikasi dengan aman. Bab 11 memberi kita strukturnya: *view* yang hanya menampilkan, dan *controller* (pengendali) yang mengerjakan tugasnya.

Rancangan. Pengendali menjalankan seluruh percakapan pada *thread* latar bernama `network`. Ia memerlukan tiga hal, dan masing-masing diberikan sebagai objek, dan itulah yang membuatnya dapat diuji. Sebuah `View` (antarmuka dengan `showLine`, `setStatus`, dan `setInputEnabled`) adalah *tempat* ia menampilkan sesuatu. Sebuah `Executor` bernama `ui` adalah *cara* ia menampilkannya: setiap perubahan pada *view* dibungkus dalam `Runnable` kecil dan diserahkan kepada `ui`, yang pada program sungguhan adalah `SwingUtilities::invokeLater`, metode yang menjalankan kode pada *event dispatch thread*. Terakhir, sebuah `Function<String, String>` menyediakan jawaban pengguna: di jendela, fungsi itu menunggu pada sebuah `BlockingQueue` yang diisi tombol **Send**; di uji, fungsi itu adalah `PoliteUser`. Dalam uji diri, `ui` cukup berupa `Runnable::run` (langsung jalankan) dan *view*-nya adalah `RecordingView` yang menyimpan setiap panggilan dalam sebuah *list*, sehingga uji dapat memeriksa persis apa yang akan dilihat pengguna.

Gambar 16.6 adalah tangkapan layar sungguhan jendela Swing dari **Kode 16.17** di tengah percakapan yang digerakkan pengendali: area percakapan, kolom masukan dengan tombol **Send**-nya, dan baris status di bagian bawah. Jendela Tkinter pada **Kode 16.18** memiliki tiga bagian yang sama.



Gambar 16.6 Jendela Knock-Knock Club (Swing) selama percakapan. Baris status berbunyi “Connected”; kolom masukan aktif karena server sedang menunggu jawaban.

Penyelesaian Java. Kode 16.17 memuat antarmuka, pengendali, *view* Swing, *view* perekam, dan sebuah pengukuran yang memperlihatkan mengapa jaringan tidak boleh berjalan pada *thread* antarmuka pengguna. Jalankan dengan `java KnockKnockGui` untuk membuka jendela (dengan server yang sudah berjalan), atau dengan `java KnockKnockGui --selftest` untuk uji tanpa layar.

Kode 16.17 `KnockKnockGui.java`: MVC dengan antarmuka `View`, pengendali yang menjalankan jaringan pada *thread*-nya sendiri, *view* Swing, dan uji diri tanpa layar.

```
// KnockKnockGui.java - Soal 16.7: klien Swing yang tetap responsif, dengan uji diri tanpa layar.
import java.awt.BorderLayout;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.net.ConnectException;
import java.net.Socket;
import java.nio.charset.StandardCharsets;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import java.util.concurrent.BlockingQueue;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executor;
import java.util.concurrent.Executors;
import java.util.concurrent.LinkedBlockingQueue;
import java.util.concurrent.TimeUnit;
import java.util.function.Function;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import javax.swing.JTextField;
import javax.swing.SwingUtilities;

public class KnockKnockGui {

    /** Yang dibutuhkan pengendali dari view apa pun: huruf V dari MVC, sebagai antarmuka. */
    interface View {
        void showLine(String line);

        void setStatus(String status);

        void setInputEnabled(boolean enabled);
    }

    /** Pengendali: bercakap dengan server pada thread latar, tidak pernah pada thread UI. */
    static final class Controller {
        private final View view;
        private final Executor ui;                          // menjalankan kode pada thread UI
        private final Function<String, String> replies;     // sumber jawaban pengguna
```

Kode 16.17 (Lanjutan) *KnockKnockGui.java: MVC dengan antarmuka View, pengendali yang menjalankan jaringan pada thread-nya sendiri, view Swing, dan uji diri tanpa layar.*

```
        Controller(View view, Executor ui, Function<String, String> replies) {
            this.view = view;
            this.ui = ui;
            this.replies = replies;
        }

        /** Memulai percakapan dan segera kembali; thread jaringan yang menunggu. */
        Thread connect(int port) {
            Thread network = new Thread(() -> talk(port), "network");
            network.setDaemon(true);
            network.start();
            return network;
        }

        private void talk(int port) {
            ui.execute(() -> view.setStatus("Connecting..."));
            try (Socket s = new Socket("localhost", port);
                 BufferedReader in = new BufferedReader(new InputStreamReader(
                         s.getInputStream(), StandardCharsets.UTF_8));
                 PrintWriter out = new PrintWriter(new OutputStreamWriter(
                         s.getOutputStream(), StandardCharsets.UTF_8), true)) {
                ui.execute(() -> view.setStatus("Connected"));
                List<String> t = KnockKnockClient.converse(in, out, line -> {
                    ui.execute(() -> {
                        view.showLine("Server: " + line);
                        view.setInputEnabled(true);
                    });
                    String reply = replies.apply(line);             // mungkin menunggu
pengguna
                    ui.execute(() -> {
                        view.setInputEnabled(false);
                        view.showLine("You: " + reply);
                    });
                    return reply;
                });
                String last = t.get(t.size() - 1);
                ui.execute(() -> {
                    if (last.equals("S: Bye.")) {
                        view.showLine("Server: Bye.");
                        view.setStatus("Conversation over");
                    } else {
                        view.setStatus("The server closed the connection");
                    }
                });
            } catch (ConnectException e) {
                ui.execute(() -> view.setStatus("Cannot reach the server: is it
running?"));
            } catch (IOException e) {
```

Kode 16.17 (Lanjutan) *KnockKnockGui.java: MVC dengan antarmuka View, pengendali yang menjalankan jaringan pada thread-nya sendiri, view Swing, dan uji diri tanpa layar.*

```
                ui.execute(() -> view.setStatus("Connection lost: " +
e.getClass().getSimpleName()));
            }
        }
    }

    /** Jendela sungguhan. Setiap metode dipanggil pada thread Swing (EDT). */
    static final class SwingView implements View {
        private final JTextArea conversation = new JTextArea(12, 40);
        private final JTextField input = new JTextField(30);
        private final JButton send = new JButton("Send");
        private final JLabel status = new JLabel("Not connected");

        SwingView(BlockingQueue<String> answers) {
            conversation.setEditable(false);
            send.addActionListener(e -> {                  // berjalan pada EDT: harus
cepat
                answers.add(input.getText());              // serahkan jawaban ke thread
jaringan
                input.setText("");
            });
            input.addActionListener(e -> send.doClick());  // Enter bekerja seperti tombol
            JPanel bottom = new JPanel(new BorderLayout());
            bottom.add(input, BorderLayout.CENTER);
            bottom.add(send, BorderLayout.EAST);
            bottom.add(status, BorderLayout.SOUTH);
            JFrame frame = new JFrame("Knock-Knock Club");
            frame.add(new JScrollPane(conversation), BorderLayout.CENTER);
            frame.add(bottom, BorderLayout.SOUTH);
            frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            frame.pack();
            frame.setVisible(true);
            setInputEnabled(false);
        }

        @Override
        public void showLine(String line) {
            conversation.append(line + "\n");
        }

        @Override
        public void setStatus(String text) {
            status.setText(text);
        }
```

Kode 16.17 (Lanjutan) *KnockKnockGui.java: MVC dengan antarmuka View, pengendali yang menjalankan jaringan pada thread-nya sendiri, view Swing, dan uji diri tanpa layar.*

```
        @Override
        public void setInputEnabled(boolean enabled) {
            input.setEnabled(enabled);
            send.setEnabled(enabled);
        }
    }

    /** View untuk pengujian: hanya mengingat apa yang diminta untuk ditampilkan. */
    static final class RecordingView implements View {
        final List<String> calls = Collections.synchronizedList(new ArrayList<>());

        @Override
        public void showLine(String line) {
            calls.add(line);
        }

        @Override
        public void setStatus(String status) {
            calls.add("[status] " + status);
        }

        @Override
        public void setInputEnabled(boolean enabled) {
            // tidak dicatat: hanya akan memperpanjang uji
        }
    }

    /** Berapa lama sebuah "repaint" menunggu bila panggilan jaringan berjalan pada thread
UI itu sendiri. */
    static long repaintDelay(boolean networkOnUiThread, int port) throws Exception {
        ExecutorService fakeEdt = Executors.newSingleThreadExecutor();
        Runnable blockingCall = () -> FailureDetection.safeTalk(port, 500,
                new KnockKnockClient.PoliteUser(1));        // menunggu 500 ms untuk server
yang diam
        if (networkOnUiThread) {
            fakeEdt.execute(blockingCall);                  // keliru: thread UI sibuk
menunggu
        } else {
            new Thread(blockingCall).start();               // benar: thread tersendiri
yang menunggu
        }
        long clicked = System.nanoTime();
        fakeEdt.submit(() -> { }).get();                    // repaint yang ditunggu
pengguna
        long waited = (System.nanoTime() - clicked) / 1_000_000;
        fakeEdt.shutdown();
        fakeEdt.awaitTermination(2, TimeUnit.SECONDS);
        return waited;
    }
```

Kode 16.17 (Lanjutan) *KnockKnockGui.java: MVC dengan antarmuka View, pengendali yang menjalankan jaringan pada thread-nya sendiri, view Swing, dan uji diri tanpa layar.*

```
    private static int passed = 0;

    private static void check(String name, Object expected, Object actual) {
        if (!java.util.Objects.equals(expected, actual)) {
            throw new AssertionError(name + ": expected " + expected + " but got " +
actual);
        }
        passed++;
        System.out.println("PASS " + name);
    }

    public static void main(String[] args) throws Exception {
        if (args.length > 0 && args[0].equals("--selftest")) {
            KnockKnockServer server = new KnockKnockServer(JokeStore.classics());
            int port = server.start(0);
            RecordingView view = new RecordingView();
            new Controller(view, Runnable::run, new
KnockKnockClient.PoliteUser(1)).connect(port).join();
            view.calls.forEach(System.out::println);
            server.stop();
            check("ten view calls", 10, view.calls.size());
            check("last status", "[status] Conversation over", view.calls.get(9));

            RecordingView lonely = new RecordingView();
            new Controller(lonely, Runnable::run, new KnockKnockClient.PoliteUser(1))
                    .connect(FailureDetection.portWithNobody()).join();
            check("server down is shown", "[status] Cannot reach the server: is it
running?",
                  lonely.calls.get(1));

            int silent = FailureDetection.badServer(List.of(), false);
            long wrong = repaintDelay(true, silent);
            long right = repaintDelay(false, FailureDetection.badServer(List.of(), false));
            System.out.println("repaint waited at least 400 ms, network on UI thread: " +
(wrong >= 400));
            System.out.println("repaint waited at least 400 ms, network on own thread: " +
(right >= 400));
            check("UI thread blocked by the wrong design", true, wrong >= 400);
            check("UI thread free in the right design", true, right < 400);
            System.out.println("All " + passed + " checks passed.");
            return;
        }
        int port = (args.length > 0) ? Integer.parseInt(args[0]) : 4444;
        BlockingQueue<String> answers = new LinkedBlockingQueue<>();
        SwingUtilities.invokeLater(() -> {
            View view = new SwingView(answers);
            Function<String, String> fromUser = line -> {
                try {
                    return answers.take();                  // thread jaringan menunggu di
sini
                } catch (InterruptedException e) {
                    return null;
```

Kode 16.17 (Lanjutan) *KnockKnockGui.java: MVC dengan antarmuka View, pengendali yang menjalankan jaringan pada thread-nya sendiri, view Swing, dan uji diri tanpa layar.*

```
                }
            };
            new Controller(view, SwingUtilities::invokeLater, fromUser).connect(port);
        });
    }
}
```

```
[status] Connecting...
[status] Connected
Server: Knock! Knock!
You: Who's there?
Server: Lettuce
You: Lettuce who?
Server: Lettuce in, it's cold out here! Want another? (y/n)
You: n
Server: Bye.
[status] Conversation over
PASS ten view calls
PASS last status
PASS server down is shown
repaint waited at least 400 ms, network on UI thread: true
repaint waited at least 400 ms, network on own thread: false
PASS UI thread blocked by the wrong design
PASS UI thread free in the right design
All 5 checks passed.
```

Penyelesaian Python. **Kode 16.18** memiliki bagian-bagian yang sama. Tkinter sama sekali tidak aman-*thread*, sehingga *thread* jaringan tidak pernah memanggil *view* secara langsung: ia memasukkan fungsi-fungsi kecil ke dalam sebuah `queue.Queue`, dan *thread* Tk mengambilnya setiap 50 ms di `drain`. Modul ini baru mengimpor `tkinter` di dalam kode jendela, sehingga uji diri tetap berjalan bahkan pada mesin tanpa Tk.

Kode 16.18 `knock_knock_gui.py`: rancangan yang sama dengan Tkinter; *thread* jaringan mengirim pembaruan *view* ke sebuah antrean yang dikosongkan *thread* Tk.

```
# knock_knock_gui.py - Soal 16.7: klien Tkinter yang tetap responsif, dengan uji diri tanpa layar.
import queue
import socket
import sys
import threading
import time
import unittest
from concurrent.futures import ThreadPoolExecutor
from joke_store import classics
from knock_knock_server import KnockKnockServer
from knock_knock_client import PoliteUser, converse
from failure_detection import safe_talk, port_with_nobody, bad_server


class Controller:
    """Bercakap dengan server pada thread latar, tidak pernah pada thread UI."""

    def __init__(self, view, ui, replies):
        self._view = view           # apa saja yang memiliki show_line, set_status, set_input_enabled
        self._ui = ui               # ui(f) menjalankan fungsi f pada thread UI
        self._replies = replies     # sumber jawaban pengguna

    def connect(self, port):
        """Memulai percakapan dan segera kembali; thread jaringan yang menunggu."""
        network = threading.Thread(target=self._talk, args=(port,), daemon=True)
        network.start()
        return network

    def _answer(self, line):
        def before():
            self._view.show_line("Server: " + line)
            self._view.set_input_enabled(True)
        self._ui(before)
        reply = self._replies(line)                     # mungkin menunggu pengguna

        def after():
            self._view.set_input_enabled(False)
            self._view.show_line(f"You: {reply}")
        self._ui(after)
        return reply
```

Kode 16.18 (Lanjutan) *knock_knock_gui.py: rancangan yang sama dengan Tkinter; thread jaringan mengirim pembaruan view ke sebuah antrean yang dikosongkan thread Tk.*

```
    def _talk(self, port):
        self._ui(lambda: self._view.set_status("Connecting..."))
        try:
            with socket.create_connection(("localhost", port)) as s, \
                    s.makefile("r", encoding="utf-8") as reader, \
                    s.makefile("w", encoding="utf-8") as writer:
                self._ui(lambda: self._view.set_status("Connected"))
                t = converse(reader, writer, self._answer)
        except ConnectionRefusedError:
            self._ui(lambda: self._view.set_status("Cannot reach the server: is it
running?"))
            return
        except OSError as e:
            name = type(e).__name__
            self._ui(lambda: self._view.set_status("Connection lost: " + name))
            return

        def finish():
            if t and t[-1] == "S: Bye.":
                self._view.show_line("Server: Bye.")
                self._view.set_status("Conversation over")
            else:
                self._view.set_status("The server closed the connection")
        self._ui(finish)


class TkView:
    """Jendela sungguhan. Setiap metode berjalan pada thread Tk."""

    def __init__(self, root, answers):
        import tkinter as tk                          # diimpor hanya bila jendela
diperlukan
        root.title("Knock-Knock Club")
        self._conversation = tk.Text(root, width=48, height=12, state="disabled")
        self._conversation.pack(fill="both", expand=True)
        bottom = tk.Frame(root)
        bottom.pack(fill="x")
        self._input = tk.Entry(bottom, width=36)
        self._input.pack(side="left", fill="x", expand=True)
        self._send = tk.Button(bottom, text="Send", command=self._on_send)
        self._send.pack(side="right")
        self._status = tk.Label(root, text="Not connected", anchor="w")
        self._status.pack(fill="x")
        self._input.bind("<Return>", lambda event: self._on_send())
        self._answers = answers
        self.set_input_enabled(False)
```

Kode 16.18 (Lanjutan) *knock_knock_gui.py: rancangan yang sama dengan Tkinter; thread jaringan mengirim pembaruan view ke sebuah antrean yang dikosongkan thread Tk.*

```
    def _on_send(self):                                 # berjalan pada thread Tk: harus
cepat
        self._answers.put(self._input.get())            # serahkan jawaban ke thread
jaringan
        self._input.delete(0, "end")

    def show_line(self, line):
        self._conversation.configure(state="normal")
        self._conversation.insert("end", line + "\n")
        self._conversation.configure(state="disabled")

    def set_status(self, text):
        self._status.configure(text=text)

    def set_input_enabled(self, enabled):
        state = "normal" if enabled else "disabled"
        self._input.configure(state=state)
        self._send.configure(state=state)


class RecordingView:
    """View untuk pengujian: hanya mengingat apa yang diminta untuk ditampilkan."""

    def __init__(self):
        self.calls = []

    def show_line(self, line):
        self.calls.append(line)

    def set_status(self, status):
        self.calls.append("[status] " + status)

    def set_input_enabled(self, enabled):
        pass                                            # tidak dicatat


def repaint_delay(network_on_ui_thread, port):
    """Berapa lama sebuah "repaint" menunggu bila panggilan jaringan berjalan pada thread
UI itu sendiri."""
    fake_ui_thread = ThreadPoolExecutor(max_workers=1)
    blocking_call = lambda: safe_talk(port, 0.5, PoliteUser(1))   # menunggu 0,5 s untuk
server yang diam
    if network_on_ui_thread:
        fake_ui_thread.submit(blocking_call)            # keliru: thread UI sibuk menunggu
    else:
        threading.Thread(target=blocking_call).start()  # benar: thread tersendiri yang
menunggu
    clicked = time.monotonic()
    fake_ui_thread.submit(lambda: None).result()        # repaint yang ditunggu pengguna
    waited = time.monotonic() - clicked
    fake_ui_thread.shutdown()
```

Kode 16.18 (Lanjutan) *knock_knock_gui.py: rancangan yang sama dengan Tkinter; thread jaringan mengirim pembaruan view ke sebuah antrean yang dikosongkan thread Tk.*

```
    return waited


class TestGui(unittest.TestCase):
    def test_one_joke_through_the_controller(self):
        server = KnockKnockServer(classics())
        view = RecordingView()
        Controller(view, lambda f: f(), PoliteUser(1)).connect(server.start(0)).join()
        server.stop()
        self.assertEqual(10, len(view.calls))
        self.assertEqual("[status] Conversation over", view.calls[9])

    def test_server_down_is_shown(self):
        view = RecordingView()
        Controller(view, lambda f: f(), PoliteUser(1)).connect(port_with_nobody()).join()
        self.assertEqual("[status] Cannot reach the server: is it running?", view.calls[1])

    def test_ui_thread_blocked_only_by_the_wrong_design(self):
        self.assertGreaterEqual(repaint_delay(True, bad_server([], False)), 0.4)
        self.assertLess(repaint_delay(False, bad_server([], False)), 0.4)


def run_window(port):
    import tkinter as tk
    root, answers = tk.Tk(), queue.Queue()
    view, jobs = TkView(root, answers), queue.Queue()

    def drain():                                        # Tk tidak aman-thread: thread Tk
        while not jobs.empty():                         # menjalankan setiap tugas yang
dikirim thread jaringan
            jobs.get()()
        root.after(50, drain)

    Controller(view, jobs.put, lambda line: answers.get()).connect(port)
    drain()
    root.mainloop()
```

Kode 16.18 (Lanjutan) *knock_knock_gui.py: rancangan yang sama dengan Tkinter; thread jaringan mengirim pembaruan view ke sebuah antrean yang dikosongkan thread Tk.*

```
if __name__ == "__main__":
    if len(sys.argv) > 1 and sys.argv[1] == "--selftest":
        server = KnockKnockServer(classics())
        view = RecordingView()
        Controller(view, lambda f: f(), PoliteUser(1)).connect(server.start(0)).join()
        server.stop()
        print("\n".join(view.calls))
        sys.stdout.flush()
        unittest.main(argv=sys.argv[:1], verbosity=2)
    else:
        run_window(int(sys.argv[1]) if len(sys.argv) > 1 else 4444)
```

```
[status] Connecting...
[status] Connected
Server: Knock! Knock!
You: Who's there?
Server: Lettuce
You: Lettuce who?
Server: Lettuce in, it's cold out here! Want another? (y/n)
You: n
Server: Bye.
[status] Conversation over
test_one_joke_through_the_controller
(__main__.TestGui.test_one_joke_through_the_controller) ... ok
test_server_down_is_shown (__main__.TestGui.test_server_down_is_shown) ... ok
test_ui_thread_blocked_only_by_the_wrong_design
(__main__.TestGui.test_ui_thread_blocked_only_by_the_wrong_design) ... ok

----------------------------------------------------------------------
Ran 3 tests in 0.503s

OK
```

Baris demi baris. `connect` membuat *thread* `network` lalu langsung kembali, sehingga penangan tombol yang memanggilnya selesai dalam hitungan mikrodetik. Di `talk`, setiap interaksi dengan *view* berbentuk `ui.execute(() -> ...)`. Fungsi pengguna yang diberikan kepada `KnockKnockClient.converse` melakukan tiga hal: mengirim "tampilkan baris server dan aktifkan masukan", lalu memanggil `replies.apply(line)`, yang di jendela *menunggu* pada antrean sampai pengguna menekan **Send**, dan kemudian mengirim "nonaktifkan masukan dan tampilkan ucapan pengguna". Penantian itu terjadi pada *thread* jaringan, tidak pernah pada *event dispatch thread*. Klausa-klausa `catch` mengubah kegagalan menjadi pesan status, dengan memakai ulang penalaran Soal 16.6. Di `SwingView`, *action listener* tombol **Send** mengerjakan sesedikit mungkin: memasukkan teks ke antrean lalu mengosongkan kolomnya. `input.addActionListener(e -> send.doClick())` membuat tombol Enter berperilaku seperti tombol **Send**. `main` membangun jendela di dalam `SwingUtilities.invokeLater`, karena bahkan pembuatan komponen Swing pun harus berlangsung pada *event dispatch thread*.

Uji diri mencatat sepuluh panggilan *view*: dua status, tujuh baris percakapan satu lelucon, `Bye.` yang ditampilkan di akhir, dan status penutup. Pengendali kedua, yang diarahkan ke port yang tidak didengarkan siapa pun, menampilkan "Cannot reach the server: is it running?" sebagai panggilan keduanya.

Pengukuran `repaintDelay` menirukan *event dispatch thread* dengan *executor* satu *thread*. Pada rancangan yang keliru, panggilan jaringan yang menahan diserahkan ke *thread* itu juga; "repaint" yang diserahkan tepat sesudahnya terpaksa menunggu sampai panggilan itu menyerah setelah 500 ms. Pada rancangan yang benar, panggilan itu berjalan pada *thread*-nya sendiri dan *repaint* langsung berjalan. Perbandingan yang dicetak berupa nilai *boolean* ("at least 400 ms: true/false") alih-alih waktu mentah, sehingga keluarannya sama di setiap mesin.

Penyelesaian keliru yang menggoda. Percobaan pertama yang wajar menaruh kode soket langsung di *action listener* tombol **Send**: baca baris server, tambahkan ke layar, selesai. Cara itu bahkan berjalan bila server cepat. Namun selama *listener* menunggu jaringan, *event dispatch thread* tidak dapat berbuat apa pun: tidak dapat menggambar ulang, menerima klik, atau menutup jendela. Dengan server yang lambat atau diam, jendela menjadi kelabu dan sistem operasi menawarkan untuk mematikannya. `repaintDelay(true, ...)` adalah rancangan itu dalam versi mini, dan ia menahan "repaint" selama setengah detik penuh.

> **UJI DIRI.** Di `talk`, mengapa kodenya menulis `String last = t.get(t.size() - 1);` sebelum `ui.execute(...)`, alih-alih memanggil `t.get(...)` di dalam *lambda*?
>
> *Jawaban.* Keduanya sama-sama berjalan di sini, tetapi menghitung `last` lebih dulu membuat *lambda* tetap kecil dan membuat data yang dibawanya ke *thread* lain menjadi eksplisit dan imutabel (sebuah `String`). *Lambda* hanya boleh memakai variabel lokal yang secara efektif final; hanya menyerahkan nilai imutabel kepada *thread* lain adalah kebiasaan paling aman, dan penalaran yang sama pula yang membuat penyimpan lelucon imutabel.

16.6.3 Soal 16.8 — Kriteria kinerja, dokumentasi produk, dan laporan

> **SOAL.** Describe the design from specification through design to implementation, and write the product documentation: a user-level guide, performance criteria, installation instructions and troubleshooting. Support the performance criteria with measurements. Outline the project report, including each member's job description and share of the work.
>
> *Dengan kata lain:* uraikan rancangan dari spesifikasi melalui rancangan hingga implementasi, dan tulislah dokumentasi produk: panduan tingkat pengguna, kriteria kinerja, petunjuk instalasi, dan pemecahan masalah. Dukunglah kriteria kinerja dengan pengukuran. Buatlah kerangka laporan proyek, termasuk uraian tugas dan bagian kerja setiap anggota.

Dengan bahasa kita sendiri. Dokumentasi adalah bagian proyek yang paling menggoda untuk ditulis paling akhir dan terburu-buru. Satu-satunya bagian yang tidak boleh dikarang-karang adalah kriteria kinerja: janji semacam "lima puluh pengguna sekaligus" hanya berharga bila sudah diukur. Jadi kita lebih dulu menulis program pengukur yang kecil, lalu dokumen-dokumen di sekitar hasilnya.

Rancangan. *Kriteria kinerja* adalah janji terukur yang memuat angka dan kondisi: "dengan 50 klien yang mulai pada saat yang sama, setiap percakapan satu lelucon selesai dalam 1000 ms pada sebuah laptop". Dua besaran menggambarkan server yang sedang dibebani. *Latensi* satu percakapan adalah waktu dari menyambung sampai mendengar `Bye.`; kita melaporkan median, yang dikalahkan oleh separuh percakapan, dan percakapan yang paling lambat. *Throughput* (laju penyelesaian) adalah

$$X = \frac{n}{t} \tag{16.2}$$

percakapan per detik, dengan n percakapan selesai dalam t detik waktu dinding. Persamaan (16.2) dan latensi dihubungkan oleh sebuah hukum antrean yang sederhana. Jika rata-rata ada L percakapan yang berlangsung bersamaan dan masing-masing memakan W detik, maka

$$L = X \cdot W \tag{16.3}$$

yang dikenal sebagai *hukum Little* [3]. Persamaan (16.3) berguna untuk memeriksa kewajaran setiap pengukuran: dengan *throughput* 900 percakapan per detik dan latensi median sekitar 5 ms, pada saat yang khas ada sekitar $900 \times 0.005 = 4.5$ percakapan yang sedang berlangsung, dan itu cocok dengan mesin berprosesor dua inti yang sekaligus menjalankan lima puluh *thread* klien.

Pengukuran berubah-ubah dari satu kali jalan ke kali berikutnya: sistem operasi menjadwalkan pekerjaan lain, kompilator *just-in-time* Java mengoptimalkan kode selagi berjalan, dan pembuatan *thread* memakan waktu. Dua kebiasaan membuat angka-angka itu bermakna: satu putaran *pemanasan* yang tidak dihitung, dan *pengulangan*, yaitu beberapa putaran yang hasilnya digabungkan.

Penyelesaian Java. **Kode 16.19** mengukur satu, sepuluh, dan lima puluh klien serentak, masing-masing lima putaran, setelah pemanasan dengan dua puluh klien.

Kode 16.19 `PerformanceProbe.java`: latensi dan *throughput* server *knock-knock* dengan 1, 10, dan 50 klien serentak.

```
// PerformanceProbe.java - Soal 16.8: mengukur angka-angka yang dijanjikan dokumentasi.
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import java.util.concurrent.CountDownLatch;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.Future;

public class PerformanceProbe {

    static final int ROUNDS = 5;

    /** Menjalankan n klien pada saat yang sama; mengembalikan waktu setiap klien dalam
milidetik, terurut. */
    static List<Double> latencies(int port, int n) throws Exception {
        ExecutorService pool = Executors.newFixedThreadPool(n);
        CountDownLatch startGate = new CountDownLatch(1);
        List<Future<Double>> futures = new ArrayList<>();
        for (int i = 0; i < n; i++) {
            futures.add(pool.submit(() -> {
                startGate.await();
                long t0 = System.nanoTime();
                List<String> t = KnockKnockClient.talkTo("localhost", port,
                                                         new
KnockKnockClient.PoliteUser(1));
                if (!t.get(t.size() - 1).equals("S: Bye.")) {
                    throw new IllegalStateException("conversation failed");
                }
                return (System.nanoTime() - t0) / 1e6;
            }));
        }
        startGate.countDown();
        List<Double> result = new ArrayList<>();
        for (Future<Double> f : futures) {
            result.add(f.get());
        }
        pool.shutdown();
        Collections.sort(result);
        return result;
    }
```

Kode 16.19 (Lanjutan) *PerformanceProbe.java: latensi dan throughput server knock-knock dengan 1, 10, dan 50 klien serentak.*

```
    public static void main(String[] args) throws Exception {
        KnockKnockServer server = new KnockKnockServer(JokeStore.classics());
        int port = server.start(0);
        latencies(port, 20);                                // pemanasan: biarkan JIT
mengompilasi kodenya
        System.out.println(" clients   median ms   slowest ms   conversations/s");
        double slowestAt50 = 0;
        for (int n : new int[] {1, 10, 50}) {
            List<Double> ms = new ArrayList<>();
            long t0 = System.nanoTime();
            for (int round = 0; round < ROUNDS; round++) {  // satu pengukuran tidak cukup
                ms.addAll(latencies(port, n));
            }
            double wallSeconds = (System.nanoTime() - t0) / 1e9;
            Collections.sort(ms);
            double median = ms.get(ms.size() / 2);
            double slowest = ms.get(ms.size() - 1);
            System.out.printf("%8d %11.1f %12.1f %17.0f%n", n, median, slowest,
                              n * ROUNDS / wallSeconds);
            slowestAt50 = slowest;
        }
        server.stop();
        System.out.println("criterion 'every one of 50 clients within 1000 ms' met: " +
(slowestAt50 < 1000));
    }
}
```

```
 clients   median ms   slowest ms   conversations/s
       1         1.7          2.1               488
      10         3.1         27.3               571
      50         4.7         59.9               905
criterion 'every one of 50 clients within 1000 ms' met: true
```

Penyelesaian Python. **Kode 16.20** adalah pengukuran yang sama. `time.perf_counter()` adalah jam Python untuk mengukur selang waktu yang pendek.

Kode 16.20 `performance_probe.py`: pengukuran yang sama dalam Python.

```
# performance_probe.py - Soal 16.8: mengukur angka-angka yang dijanjikan dokumentasi.
import threading
import time
from concurrent.futures import ThreadPoolExecutor
from joke_store import classics
from knock_knock_server import KnockKnockServer
from knock_knock_client import PoliteUser, talk_to

ROUNDS = 5


def latencies(port, n):
    """Menjalankan n klien pada saat yang sama; mengembalikan waktu setiap klien dalam
milidetik, terurut."""
    start_gate = threading.Event()

    def one_client():
        start_gate.wait()
        t0 = time.perf_counter()
        t = talk_to("localhost", port, PoliteUser(1))
        if t[-1] != "S: Bye.":
            raise RuntimeError("conversation failed")
        return (time.perf_counter() - t0) * 1000

    with ThreadPoolExecutor(max_workers=n) as pool:
        futures = [pool.submit(one_client) for _ in range(n)]
        start_gate.set()
        return sorted(f.result() for f in futures)


if __name__ == "__main__":
    server = KnockKnockServer(classics())
    port = server.start(0)
    latencies(port, 20)                                  # pemanasan
    print(" clients   median ms   slowest ms   conversations/s")
    for n in (1, 10, 50):
        ms, t0 = [], time.perf_counter()
        for _ in range(ROUNDS):                          # satu pengukuran tidak cukup
            ms += latencies(port, n)
        wall_seconds = time.perf_counter() - t0
        ms.sort()
        median, slowest = ms[len(ms) // 2], ms[-1]
        print(f"{n:8d} {median:11.1f} {slowest:12.1f} {n * ROUNDS / wall_seconds:17.0f}")
    server.stop()
    print("criterion 'every one of 50 clients within 1000 ms' met:", slowest < 1000)
```

```
 clients   median ms   slowest ms   conversations/s
       1         4.2          8.9              149
      10         5.3         28.8              304
      50        16.0         42.1              701
criterion 'every one of 50 clients within 1000 ms' met: True
```

Angka Anda akan berbeda dari angka-angka ini, demikian pula angka dari kali jalan kedua di mesin yang sama; yang tidak boleh berbeda adalah baris terakhirnya. Perhatikan pada keluaran sebenarnya bahwa *throughput naik* dari 1 ke 50 klien di kedua bahasa: satu klien menanggung seluruh biaya koneksi baru dan *thread* server baru untuk setiap percakapan dan membiarkan prosesor menganggur selagi ia menunggu jaringan, sedangkan lima puluh klien serentak membuat kedua inti prosesor mesin pembangun tetap sibuk. Efek semacam itu tidak dapat ditebak; efek itu harus diukur.

Baris demi baris. `latencies` memakai ulang gerbang mulai dari Soal 16.5, mencatat `System.nanoTime()` sebelum dan sesudah setiap percakapan, dan mengembalikan waktu-waktu yang terurut dalam milidetik (nanodetik dibagi 10^6). Gelung utama menggabungkan lima putaran

untuk setiap banyaknya klien, mengurutkannya, mengambil elemen tengah sebagai median dan elemen terakhir sebagai yang paling lambat, lalu menghitung Persamaan (16.2) dengan n sama dengan banyaknya percakapan dalam kelima putaran. `printf("%8d %11.1f ...")` mencetak kolom-kolom selebar tetap: `%8d` adalah bilangan bulat dalam 8 karakter dan `%11.1f` bilangan dengan satu angka desimal dalam 11 karakter. Baris terakhir menyatakan kriterianya dan apakah kali jalan ini memenuhinya.

Dokumentasinya. Dengan hasil pengukuran di tangan, dokumentasi produk adalah seperangkat dokumen pendek yang konkret. **Tabel 16.7** adalah kerangka yang dipakai Knock-Knock Club; proyek sungguhan mengganti isinya tetapi mempertahankan judul-judulnya.

Tabel 16.7 Dokumentasi produk Knock-Knock Club, satu baris per dokumen.

Dokumen	Isi untuk Knock-Knock Club
Spesifikasi	tujuan, cerita pengguna pada Soal 16.1, **Tabel 16.5**
Rancangan	**Gambar 16.1**, **Gambar 16.4**, kontrak `JokeStore`, protokol, dan `View`
Panduan pengguna	"Jalankan jendela, tunggu sampai *Connected*, jawab baris server di kotak masukan lalu tekan Send atau Enter; jawab `n` untuk keluar."
Kriteria kinerja	"50 percakapan satu lelucon yang serentak, masing-masing dalam 1000 ms; diukur dengan `PerformanceProbe`."
Instalasi	pasang JDK 17+ atau Python 3.10+; salin foldernya; jalankan `javac *.java` sekali; jalankan server, lalu klien-kliennya
Pemecahan masalah	**Tabel 16.8**
Rencana uji	uji kontrak (16.1), uji protokol (16.2), uji silang dengan *stub* (16.4), uji konkurensi (16.5), uji kegagalan (16.6), uji GUI tanpa layar (16.7), pengukur kinerja (16.8)

Tabel pemecahan masalah mendaftarkan gejala yang benar-benar dilihat pengguna, dengan kata-kata mereka, beserta penyebab dan obatnya. **Tabel 16.8** memuat entri-entri yang kemungkinan besar muncul dari skenario kegagalan pada Soal 16.6 dan dari pemakaian soket sehari-hari.

Tabel 16.8 Pemecahan masalah Knock-Knock Club: gejala seperti yang dilihat pengguna, penyebab, dan obatnya.

Gejala	Kemungkinan penyebab	Yang harus dilakukan
"Cannot reach the server: is it running?"	server belum dijalankan, host atau port salah, *firewall*	jalankan `KnockKnockServer`; periksa host dan port; izinkan port itu di *firewall*
server mencetak `Address already in use` (Java `BindException`, Python `OSError`)	program lain, atau salinan server yang lama, memakai port itu	hentikan salinan lama, atau jalankan server pada port lain: `java KnockKnockServer 5555`
"The server did not answer within 500 ms."	server kelebihan beban, macet, atau berada di seberang jaringan yang lambat	coba lagi; periksa konsol server; naikkan batas waktu untuk jaringan yang lambat
"The server closed the connection"	server dihentikan atau mengalami *crash*	jalankan ulang server; laporkan keluaran konsolnya
jendela tetap kelabu atau tidak menanggapi	kode jaringan berjalan pada *thread* antarmuka pengguna	kesalahan pemrograman: lihat Soal 16.7

Laporannya. Laporan proyek yang dapat dibaca dan diperiksa orang lain memiliki kerangka yang tetap. (1) *Pendahuluan*: tujuan dan hal-hal yang bukan tujuan. (2) *Spesifikasi*: cerita pengguna dan kontrak-kontraknya. (3) *Rancangan*: diagram-diagram, tabel protokol, dan alasan keputusan-keputusan utama (mengapa satu *thread* per klien, mengapa penyimpan yang imutabel). (4) *Implementasi*: struktur kode, dengan pseudokode atau cuplikan pendek untuk bagian-bagian yang

menarik. (5) *Pengujian*: rencana uji dan keluaran uji yang sebenarnya. (6) *Evaluasi*: kinerja terukur dibandingkan dengan kriteria, dan apa yang belum berjalan. (7) *Kesimpulan*: apa yang dipelajari tim dan apa yang akan dikerjakan secara berbeda. (8) *Kontribusi*: untuk setiap anggota, bagian yang dimiliki, pekerjaan yang dilakukan, dan persentase yang disepakati dari keseluruhan, misalnya "anggota 1, 33%: `JokeStore` dan kedua implementasinya, uji kontrak, panduan instalasi". Menulis kontribusi sambil proyek berjalan, seperti saran Langkah 10, membuatnya akurat dan tidak diperdebatkan.

Penyelesaian keliru yang menggoda. Kriteria yang menggoda adalah satu pengukuran yang ditulis sebagai fakta: "server menangani 50 pengguna dalam 2 ms", diambil dari satu kali jalan yang kebetulan mujur. Keluaran di atas memperlihatkan mengapa hal itu menyesatkan: dari satu kali jalan ke kali berikutnya, percakapan paling lambat di antara kelima puluh percakapan bervariasi antara sekitar 15 dan 70 ms di mesin yang sama. Sebuah kriteria memerlukan kondisi ("50 klien serentak"), batas dengan cadangan ("dalam 1000 ms"), dan metode yang dapat diulang siapa pun ("diukur dengan `PerformanceProbe`, lima putaran setelah pemanasan").

> **UJI DIRI.** Misalkan sebuah pengukuran memperlihatkan *throughput* 1000 percakapan per detik sementara rata-rata ada 20 percakapan yang berlangsung. Berapa latensi rata-rata yang diramalkan Persamaan (16.3)?
>
> *Jawaban.* $W = L/X = 20/1000 = 0.02$ detik, yaitu 20 ms per percakapan.

16.7 Soal Baru yang Dikerjakan: Iterator dan Nilai Imutabel

Enam soal yang tersisa adalah soal baru, masing-masing berlatar studi kasus yang berjalan sepanjang buku ini, Profitina Laundry. Dua soal pertama melatih Bab 13.

16.7.1 Soal 16.9 — Jadwal penjemputan dengan iteratornya sendiri

> **SOAL.** Kurir-kurir Profitina Laundry menjemput pesanan pada jam-jam tertentu. Tulislah kelas `PickupSchedule` yang menyimpan penjemputan (pengenal pesanan dan jamnya) dalam lariknya sendiri, selalu terurut menurut jam, dengan penjemputan pada jam yang sama tetap dalam urutan saat ditambahkan. Buatlah kelas itu dapat dipakai dalam gelung for-each, berikan kepadanya sebuah tampilan *malas* (*lazy*) atas penjemputan di antara dua jam, dan buatlah iteratornya *fail fast* (gagal seketika) bila jadwal diubah di tengah sebuah gelung.

Dengan bahasa kita sendiri. Gelung for-each di Java dan gelung `for` di Python bekerja dengan objek apa pun yang dapat menghasilkan sebuah *iterator*: objek yang menyerahkan elemen satu per satu dan mengingat sudah sampai di mana. Kita akan menulis objek itu sendiri. "Malas" berarti tampilan itu tidak bekerja sampai ada yang beriterasi: ia tidak membangun *list* baru, melainkan mencari setiap penjemputan yang cocok ketika gelung meminta elemen berikutnya. "Fail fast" berarti gelung atas jadwal yang diubah seseorang di tengah jalan langsung berhenti dengan eksepsi, alih-alih diam-diam melompati atau mengulang elemen.

Rancangan. Jadwal menyimpan larik `slots`, cacahan `size`, dan sebuah *cacahan modifikasi* `modCount` yang bertambah satu pada setiap perubahan. Iteratornya adalah *kelas dalam* (*inner class*), `Cursor`, sehingga setiap objek iterator dapat melihat *field* jadwalnya; ia menyimpan posisinya sendiri `next` dan salinan `modCount` yang diambil ketika gelung dimulai. Setiap panggilan `next()` membandingkan kedua cacahan itu. Tampilan malas `between(from, to)` mengembalikan sebuah `Iterable` yang iteratornya membungkus iterator biasa dan melewati penjemputan yang tidak cocok. Penambahan menjaga larik tetap terurut dengan langkah penyisipan dari *insertion*

sort: geser penjemputan yang lebih lambat satu tempat ke kanan, lalu letakkan yang baru di celahnya.

Penyelesaian Java. **Kode 16.21** memuat jadwal, kursor, tampilan malas, dan rancangan keliru yang koleksinya menjadi iteratornya sendiri.

Kode 16.21 `PickupSchedule.java`: larik terurut dengan iterator berupa kelas dalam, tampilan tersaring yang malas, dan pemeriksaan *fail-fast*.

```
// PickupSchedule.java - Soal 16.9: koleksi dengan iterator sendiri, tampilan malas, dan
pemeriksaan fail-fast.
import java.util.ArrayList;
import java.util.Arrays;
import java.util.ConcurrentModificationException;
import java.util.Iterator;
import java.util.List;
import java.util.NoSuchElementException;

public class PickupSchedule implements Iterable<PickupSchedule.Pickup> {

    /** Satu penjemputan kurir: pesanan mana, pada jam berapa. */
    record Pickup(String orderId, int hour) {
    }

    private Pickup[] slots = new Pickup[2];
    private int size = 0;
    private int modCount = 0;                       // berapa kali jadwal telah berubah

    /** Menambahkan penjemputan, menjaga jadwal terurut menurut jam (jam yang sama
mempertahankan urutannya). */
    public void add(Pickup p) {
        if (size == slots.length) {
            slots = Arrays.copyOf(slots, 2 * size);             // tumbuh, seperti di Bab 8
        }
        int i = size;
        while (i > 0 && slots[i - 1].hour() > p.hour()) {       // geser penjemputan yang
lebih lambat ke kanan
            slots[i] = slots[i - 1];
            i--;
        }
        slots[i] = p;
        size++;
        modCount++;
    }

    /** Menghapus penjemputan satu pesanan; mengembalikan false jika tidak ada. */
    public boolean removeOrder(String orderId) {
        for (int i = 0; i < size; i++) {
            if (slots[i].orderId().equals(orderId)) {
                System.arraycopy(slots, i + 1, slots, i, size - i - 1);
                slots[--size] = null;
                modCount++;
                return true;
            }
        }
        return false;
    }
```

Kode 16.21 (Lanjutan) *PickupSchedule.java: larik terurut dengan iterator berupa kelas dalam, tampilan tersaring yang malas, dan pemeriksaan fail-fast.*

```
    @Override
    public Iterator<Pickup> iterator() {
        return new Cursor();                        // posisi BARU untuk setiap gelung
    }

    /** Iterator: kelas dalam, sehingga dapat melihat slots, size, dan modCount. */
    private final class Cursor implements Iterator<Pickup> {
        private int next = 0;
        private final int expectedModCount = modCount;

        @Override
        public boolean hasNext() {
            return next < size;
        }

        @Override
        public Pickup next() {
            if (modCount != expectedModCount) {
                throw new ConcurrentModificationException("schedule changed during
iteration");
            }
            if (!hasNext()) {
                throw new NoSuchElementException();
            }
            return slots[next++];
        }
    }

    /** Tampilan malas: penjemputan dengan from <= hour < to, dicari hanya selagi ada yang
beriterasi. */
    public Iterable<Pickup> between(int from, int to) {
        return () -> new Iterator<>() {
            private final Iterator<Pickup> all = iterator();
            private Pickup ahead = advance();               // kecocokan berikutnya, atau
null

            private Pickup advance() {
                while (all.hasNext()) {
                    Pickup p = all.next();
                    if (p.hour() >= from && p.hour() < to) {
                        return p;
                    }
                }
                return null;
            }
```

Kode 16.21 (Lanjutan) *PickupSchedule.java: larik terurut dengan iterator berupa kelas dalam, tampilan tersaring yang malas, dan pemeriksaan fail-fast.*

```
            @Override
            public boolean hasNext() {
                return ahead != null;
            }

            @Override
            public Pickup next() {
                if (ahead == null) {
                    throw new NoSuchElementException();
                }
                Pickup p = ahead;
                ahead = advance();
                return p;
            }
        };
    }

    /** Menggoda tetapi keliru: koleksi ADALAH iteratornya sendiri, sehingga hanya punya
satu posisi. */
    static final class SelfIteratingSchedule implements Iterable<String>, Iterator<String>
{
        private final List<String> ids;
        private int next = 0;

        SelfIteratingSchedule(List<String> ids) {
            this.ids = ids;
        }

        @Override
        public Iterator<String> iterator() {
            return this;                              // setiap gelung memakai "next" yang
sama
        }

        @Override
        public boolean hasNext() {
            return next < ids.size();
        }

        @Override
        public String next() {
            return ids.get(next++);
        }
    }

    private static int passed = 0;
```

Kode 16.21 (Lanjutan) *PickupSchedule.java: larik terurut dengan iterator berupa kelas dalam, tampilan tersaring yang malas, dan pemeriksaan fail-fast.*

```
    private static void check(String name, Object expected, Object actual) {
        if (!java.util.Objects.equals(expected, actual)) {
            throw new AssertionError(name + ": expected " + expected + " but got " +
actual);
        }
        passed++;
        System.out.println("PASS " + name);
    }

    public static void main(String[] args) {
        PickupSchedule s = new PickupSchedule();
        s.add(new Pickup("A1", 14));
        s.add(new Pickup("A2", 9));
        s.add(new Pickup("A3", 17));
        s.add(new Pickup("A4", 11));
        s.add(new Pickup("A5", 9));
        List<String> all = new ArrayList<>();
        for (Pickup p : s) {                          // gelung for-each memanggil iterator()
            all.add(p.orderId() + "@" + p.hour());
        }
        System.out.println("whole day : " + all);
        List<String> morning = new ArrayList<>();
        for (Pickup p : s.between(8, 12)) {
            morning.add(p.orderId());
        }
        System.out.println("morning   : " + morning);

        int pairs = 0;
        for (Pickup a : s) {
            for (Pickup b : s) {
                pairs++;
            }
        }
        SelfIteratingSchedule wrong = new SelfIteratingSchedule(List.of("A1", "A2", "A3",
"A4", "A5"));
        int wrongPairs = 0;
        for (String a : wrong) {
            for (String b : wrong) {
                wrongPairs++;
            }
        }
        System.out.println("pairs     : " + pairs + " (self-iterating version: " +
wrongPairs + ")");
```

Kode 16.21 (Lanjutan) *PickupSchedule.java: larik terurut dengan iterator berupa kelas dalam, tampilan tersaring yang malas, dan pemeriksaan fail-fast.*

```
        String failure = "none";
        try {
            for (Pickup p : s) {
                if (p.hour() == 11) {
                    s.removeOrder(p.orderId());     // mengubah jadwal di dalam gelung
                }
            }
        } catch (ConcurrentModificationException e) {
            failure = e.getMessage();
        }
        System.out.println("remove in loop: " + failure);

        check("sorted by hour, stable", "[A2@9, A5@9, A4@11, A1@14, A3@17]",
all.toString());
        check("lazy view", List.of("A2", "A5", "A4"), morning);
        check("nested loops", 25, pairs);
        check("self-iterating loses pairs", 4, wrongPairs);
        check("fail-fast", "schedule changed during iteration", failure);
        Iterator<Pickup> it = new PickupSchedule().iterator();
        check("empty schedule", false, it.hasNext());
        try {
            it.next();
            throw new AssertionError("no exception");
        } catch (NoSuchElementException e) {
            check("next past the end", "NoSuchElementException",
e.getClass().getSimpleName());
        }
        System.out.println("All " + passed + " checks passed.");
    }
}
```

```
whole day : [A2@9, A5@9, A4@11, A1@14, A3@17]
morning   : [A2, A5, A4]
pairs     : 25 (self-iterating version: 4)
remove in loop: schedule changed during iteration
PASS sorted by hour, stable
PASS lazy view
PASS nested loops
PASS self-iterating loses pairs
PASS fail-fast
PASS empty schedule
PASS next past the end
All 7 checks passed.
```

Penyelesaian Python. Di Python, `__iter__` harus mengembalikan sebuah iterator, dan iterator adalah objek apa pun yang memiliki metode `__next__` yang memunculkan `StopIteration` di akhir. **Kode 16.22** menulis kursornya dengan tangan untuk memperlihatkan protokolnya, dan menulis tampilan malasnya sebagai *generator*, yaitu fungsi yang memuat `yield`, yang kelas iteratornya dituliskan Python sendiri.

Kode 16.22 `pickup_schedule.py`: jadwal yang sama dengan kelas iterator eksplisit dan generator untuk tampilan malas.

```
# pickup_schedule.py - Soal 16.9: koleksi dengan iterator sendiri, tampilan malas, dan
pemeriksaan fail-fast.
import sys
import unittest
from dataclasses import dataclass


@dataclass(frozen=True)
class Pickup:
    """Satu penjemputan kurir: pesanan mana, pada jam berapa."""
    order_id: str
    hour: int


class PickupSchedule:
    def __init__(self):
        self._slots = []
        self._mod_count = 0                     # berapa kali jadwal telah berubah

    def add(self, pickup):
        """Menambahkan penjemputan, menjaga jadwal terurut menurut jam (jam yang sama
mempertahankan urutannya)."""
        i = len(self._slots)
        while i > 0 and self._slots[i - 1].hour > pickup.hour:
            i -= 1
        self._slots.insert(i, pickup)
        self._mod_count += 1

    def remove_order(self, order_id):
        """Menghapus penjemputan satu pesanan; mengembalikan False jika tidak ada."""
        for i, p in enumerate(self._slots):
            if p.order_id == order_id:
                del self._slots[i]
                self._mod_count += 1
                return True
        return False

    def __len__(self):
        return len(self._slots)

    def __iter__(self):
        return _Cursor(self)                    # posisi BARU untuk setiap gelung
```

Kode 16.22 (Lanjutan) *pickup_schedule.py: jadwal yang sama dengan kelas iterator eksplisit dan generator untuk tampilan malas.*

```
    def between(self, start, end):
        """Tampilan malas: generator mencari penjemputan dengan start <= hour < end
        hanya selagi ada yang beriterasi."""
        for p in self:
            if start <= p.hour < end:
                yield p


class _Cursor:
    """Iterator, ditulis dengan tangan: __iter__ dan __next__."""

    def __init__(self, schedule):
        self._schedule = schedule
        self._next = 0
        self._expected = schedule._mod_count

    def __iter__(self):
        return self

    def __next__(self):
        if self._schedule._mod_count != self._expected:
            raise RuntimeError("schedule changed during iteration")
        if self._next >= len(self._schedule._slots):
            raise StopIteration                # cara Python mengatakan "sudah habis"
        p = self._schedule._slots[self._next]
        self._next += 1
        return p


class SelfIteratingSchedule:
    """Menggoda tetapi keliru: koleksi ADALAH iteratornya sendiri, sehingga hanya punya
satu posisi."""

    def __init__(self, ids):
        self._ids = list(ids)
        self._next = 0

    def __iter__(self):
        return self                            # setiap gelung memakai _next yang sama

    def __next__(self):
        if self._next >= len(self._ids):
            raise StopIteration
        self._next += 1
        return self._ids[self._next - 1]
```

Kode 16.22 (Lanjutan) *pickup_schedule.py: jadwal yang sama dengan kelas iterator eksplisit dan generator untuk tampilan malas.*

```
def sample():
    s = PickupSchedule()
    for order_id, hour in (("A1", 14), ("A2", 9), ("A3", 17), ("A4", 11), ("A5", 9)):
        s.add(Pickup(order_id, hour))
    return s


def count_pairs(collection):
    return sum(1 for _a in collection for _b in collection)


class TestSchedule(unittest.TestCase):
    def test_sorted_by_hour_and_stable(self):
        self.assertEqual(["A2@9", "A5@9", "A4@11", "A1@14", "A3@17"],
                         [f"{p.order_id}@{p.hour}" for p in sample()])

    def test_lazy_view(self):
        self.assertEqual(["A2", "A5", "A4"], [p.order_id for p in sample().between(8, 12)])

    def test_nested_loops(self):
        self.assertEqual(25, count_pairs(sample()))
        self.assertEqual(4, count_pairs(SelfIteratingSchedule(["A1", "A2", "A3", "A4",
"A5"])))

    def test_fail_fast(self):
        s = sample()
        with self.assertRaises(RuntimeError):
            for p in s:
                if p.hour == 11:
                    s.remove_order(p.order_id)

    def test_next_past_the_end(self):
        with self.assertRaises(StopIteration):
            next(iter(PickupSchedule()))


if __name__ == "__main__":
    s = sample()
    print("whole day :", [f"{p.order_id}@{p.hour}" for p in s])
    print("morning   :", [p.order_id for p in s.between(8, 12)])
    print("pairs     :", count_pairs(s), "(self-iterating version:",
          str(count_pairs(SelfIteratingSchedule(["A1", "A2", "A3", "A4", "A5"]))) + ")")
    view = s.between(8, 12)                      # objek generator habis terpakai setelah
satu gelung
    print("one generator, two loops:", len(list(view)), "then", len(list(view)))
    sys.stdout.flush()
    unittest.main(verbosity=2)
```

```
whole day : ['A2@9', 'A5@9', 'A4@11', 'A1@14', 'A3@17']
morning   : ['A2', 'A5', 'A4']
pairs     : 25 (self-iterating version: 4)
one generator, two loops: 3 then 0
test_fail_fast (__main__.TestSchedule.test_fail_fast) ... ok
test_lazy_view (__main__.TestSchedule.test_lazy_view) ... ok
test_nested_loops (__main__.TestSchedule.test_nested_loops) ... ok
test_next_past_the_end (__main__.TestSchedule.test_next_past_the_end) ... ok
test_sorted_by_hour_and_stable (__main__.TestSchedule.test_sorted_by_hour_and_stable) ...
ok

----------------------------------------------------------------------
Ran 5 tests in 0.000s

OK
```

Baris demi baris. Di `add`, gelung `while` menggeser setiap penjemputan dengan jam yang *lebih lambat* satu tempat ke kanan; perbandingannya `>` dan bukan `>=`, dan itulah yang menjaga jam-jam yang sama tetap dalam urutan kedatangannya: A2 dan A5 sama-sama pada pukul 9 dan tetap dalam urutan itu. Ketika larik penuh, `Arrays.copyOf` menggandakannya, sesuai aturan pertumbuhan Soal 8.17. `iterator()` selalu mengembalikan `new Cursor()`, sehingga setiap gelung mendapat posisinya sendiri; itulah yang membuat gelung bersarang menghitung $5 \times 5 = 25$ pasangan. `Cursor` mencatat `expectedModCount` ketika dibuat; `next()` melempar `ConcurrentModificationException` bila `modCount` jadwal telah berubah sejak saat itu, dan hal itu terjadi pada keluaran ketika gelung menghapus A4. Iterator anonim dari `between` melihat satu elemen *ke depan*: `ahead` selalu menyimpan kecocokan berikutnya atau `null`, sehingga `hasNext()` cukup berupa `ahead != null`, dan `next()` mengembalikan `ahead` lalu bergerak maju. Tampilan itu tidak pernah menyalin jadwal, dan bila jadwal berubah di antara dua gelung, gelung berikutnya melihat perubahan itu.

Di Python, `_Cursor.__next__` memunculkan `StopIteration`, yang ditangkap diam-diam oleh pernyataan `for` untuk mengakhiri gelung. Generator `between` memuat `for p in self: if ...: yield p`; memanggilnya sama sekali tidak menjalankan kode, hanya membuat objek generator, dan setiap `next` menjalankan badannya sampai `yield` berikutnya. Baris terakhir yang dicetak sebelum uji memperlihatkan satu perbedaan penting: objek generator *adalah* iterator, sehingga hanya dapat dipakai untuk tepat satu gelung. `len(list(view))` bernilai 3 pada kali pertama dan 0 pada kali kedua. `s.between(8, 12)` membuat generator baru setiap kali dipanggil, dan itulah sebabnya gelung-gelung milik jadwal sendiri tidak pernah terkena masalah ini.

Penyelesaian keliru yang menggoda. `SelfIteratingSchedule` mengimplementasikan `Iterable` dan `Iterator` sekaligus, dan `iterator()` mengembalikan `this`. Barisnya lebih sedikit, dan gelung tunggal berjalan dengan baik. Namun setiap gelung atasnya memakai satu posisi bersama. Pada gelung bersarang, gelung luar mengambil A1, gelung dalam menelusuri A2 sampai A5 dan menghabiskan posisi bersama itu, dan ketika gelung dalam berakhir, gelung luar tidak menemukan apa-apa lagi: 4 pasangan alih-alih 25. Untuk koleksi berisi n elemen, gelung bersarang mengunjungi

$$P_{\text{sendiri}} = n^2, \qquad P_{\text{bersama}} = n - 1 \tag{16.4}$$

pasangan, masing-masing dengan satu iterator per gelung dan dengan satu iterator bersama; Persamaan (16.4) dengan $n = 5$ menghasilkan angka 25 dan 4 pada keluaran. Sebuah koleksi harus menyerahkan iterator *baru* untuk setiap gelung.

> **UJI DIRI.** Pada gelung *fail-fast*, kode menghapus A4, penjemputan ketiga dari lima. Apa yang terjadi jika gelung itu menghapus penjemputan pukul 14 (A1, yang keempat)?
>
> *Jawaban.* Setelah A1 dihapus, `size` bernilai 4 dan `next` milik kursor juga 4, sehingga `hasNext()` mengembalikan `false` dan gelung berakhir secara normal, tanpa eksepsi dan tanpa mengunjungi A3. Iterator `ArrayList` milik Java sendiri berperilaku persis sama untuk elemen kedua dari belakang. Itulah sebabnya pemeriksaan ini disebut "upaya terbaik": iterator *fail-fast* menangkap sebagian besar kekeliruan, tetapi kode yang benar tidak boleh mengandalkan eksepsinya. Untuk menghapus di dalam gelung, pakailah `remove` milik iterator itu sendiri atau `removeIf` (Latihan 16.2).

16.7.2 Soal 16.10 — Uang sebagai objek nilai yang imutabel

> **SOAL.** Tulislah kelas `Money` untuk Profitina Laundry: jumlah dalam satuan utuh dan kode mata uang tiga huruf. Objek `Money` tidak boleh berubah setelah dibuat; menjumlahkan dua nilai atau

> menghargai suatu berat harus menghasilkan objek baru. Dua objek `Money` dengan jumlah dan mata uang yang sama harus sama dan harus berfungsi sebagai kunci yang sama di sebuah *hash map*. Penjumlahan mata uang yang berbeda harus ditolak.

Dengan bahasa kita sendiri. Sebagian objek memiliki identitas: dua pelanggan bernama Sari adalah dua orang. Sebagian lain hanyalah nilai: dua lembar uang Rp20.000 dapat saling dipertukarkan. *Objek nilai* adalah kelas jenis kedua, dan tiga hal membuatnya berperilaku seperti bilangan: ia imutabel, `equals` membandingkan nilai dan bukan identitas objek, dan `hashCode` sejalan dengan `equals`. Butir terakhir itulah tempat sebagian besar kelas nilai buatan sendiri gagal, tanpa suara.

> **DEFINISI.** *Kontrak hash*: jika `a.equals(b)` bernilai benar, maka `a.hashCode() == b.hashCode()` juga harus benar. *Hash map* menemukan sebuah kunci dengan terlebih dahulu menghitung kode *hash*-nya lalu hanya mencari di ember (*bucket*) itu; kunci yang kode *hash*-nya tidak sejalan dengan `equals` dicari di ember yang salah.

Rancangan. Kelasnya `final` sehingga tidak ada subkelas yang dapat menambahkan *field* yang dapat berubah; kedua *field*-nya `private final`; konstruktornya privat dan *factory* statis `of` memvalidasi mata uang (tiga huruf kapital) dan jumlahnya (tidak negatif), sehingga *setiap* `Money` yang ada pasti sah. `plus` dan `times` mengembalikan objek baru. `equals` dan `hashCode` memakai dua *field* yang persis sama. Jumlah uang berupa satuan utuh yang disimpan dalam `long`, tidak pernah `double`, dengan alasan yang sama seperti pada Soal 8.18: bilangan titik mengambang biner tidak dapat menyatakan kebanyakan pecahan desimal secara tepat.

Penyelesaian Java. **Kode 16.23** memperlihatkan kelas itu, objek nilai yang sama sebagai *record*, dan dua varian yang rusak.

Kode 16.23 `Money.java`: objek nilai imutabel dengan pembuatan yang tervalidasi, kesamaan nilai, kode *hash* yang konsisten, pengurutan, dan dua varian yang rusak.

```java
// Money.java - Soal 16.10: objek nilai imutabel dengan equals, hashCode, dan compareTo.
import java.util.HashMap;
import java.util.HashSet;
import java.util.List;
import java.util.Locale;
import java.util.Map;
import java.util.Objects;
import java.util.Set;
import java.util.TreeMap;

public final class Money implements Comparable<Money> {    // final: tidak ada subkelas yang dapat menambahkan setter

    private final String currency;                  // "IDR"
    private final long amount;                      // satuan utuh: rupiah tidak memakai sen dalam keseharian

    private Money(String currency, long amount) {
        this.currency = currency;
        this.amount = amount;
    }

    /** Satu-satunya pintu masuk: setiap Money yang ada telah lolos pemeriksaan ini. */
    public static Money of(long amount, String currency) {
        if (currency == null || !currency.matches("[A-Z]{3}")) {
            throw new IllegalArgumentException("currency must be three capital letters: " + currency);
        }
        if (amount < 0) {
            throw new IllegalArgumentException("amount must not be negative: " + amount);
        }
        return new Money(currency, amount);
    }

    public static Money rupiah(long amount) {
        return of(amount, "IDR");
    }

    public long amount() {
        return amount;
    }

    public String currency() {
        return currency;
    }
```

Kode 16.23 (Lanjutan) *Money.java: objek nilai imutabel dengan pembuatan yang tervalidasi, kesamaan nilai, kode hash yang konsisten, pengurutan, dan dua varian yang rusak.*

```
    /** Mengembalikan Money BARU; objek ini tidak pernah berubah. */
    public Money plus(Money other) {
        requireSameCurrency(other);
        return new Money(currency, Math.addExact(amount, other.amount));   // luapan
melempar eksepsi
    }

    /** Harga kg kilogram dengan harga per kilogram ini, dibulatkan ke satuan utuh. */
    public Money times(double kg) {
        return new Money(currency, Math.round(amount * kg));
    }

    private void requireSameCurrency(Money other) {
        if (!currency.equals(other.currency)) {
            throw new IllegalArgumentException(currency + " and " + other.currency + " do
not mix");
        }
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) {
            return true;
        }
        if (!(o instanceof Money)) {
            return false;
        }
        Money m = (Money) o;
        return amount == m.amount && currency.equals(m.currency);
    }

    @Override
    public int hashCode() {
        return Objects.hash(currency, amount);      // dibangun dari field yang persis sama
dengan yang dipakai equals
    }

    @Override
    public int compareTo(Money other) {
        requireSameCurrency(other);
        return Long.compare(amount, other.amount);
    }

    @Override
    public String toString() {
        return String.format(Locale.ROOT, "%s %,d", currency, amount);
    }
```

Kode 16.23 (Lanjutan) *Money.java: objek nilai imutabel dengan pembuatan yang tervalidasi, kesamaan nilai, kode hash yang konsisten, pengurutan, dan dua varian yang rusak.*

```
    /** Objek nilai yang sama sebagai record: Java menuliskan equals, hashCode, dan
toString untuk kita. */
    record MoneyRecord(String currency, long amount) {
        MoneyRecord {
            if (amount < 0) {
                throw new IllegalArgumentException("amount must not be negative: " +
amount);
            }
        }
    }

    /** Menggoda tetapi keliru (1): equals tanpa hashCode. */
    static final class NoHashMoney {
        final long amount;

        NoHashMoney(long amount) {
            this.amount = amount;
        }

        @Override
        public boolean equals(Object o) {
            return o instanceof NoHashMoney && ((NoHashMoney) o).amount == amount;
        }
    }

    /** Menggoda tetapi keliru (2): kunci yang dapat berubah setelah masuk ke himpunan. */
    static final class MutableMoney {
        long amount;

        MutableMoney(long amount) {
            this.amount = amount;
        }

        @Override
        public boolean equals(Object o) {
            return o instanceof MutableMoney && ((MutableMoney) o).amount == amount;
        }

        @Override
        public int hashCode() {
            return Long.hashCode(amount);
        }
    }

    private static int passed = 0;
```

Kode 16.23 (Lanjutan) *Money.java: objek nilai imutabel dengan pembuatan yang tervalidasi, kesamaan nilai, kode hash yang konsisten, pengurutan, dan dua varian yang rusak.*

```
    private static void check(String name, Object expected, Object actual) {
        if (!Objects.equals(expected, actual)) {
            throw new AssertionError(name + ": expected " + expected + " but got " +
actual);
        }
        passed++;
        System.out.println("PASS " + name);
    }

    public static void main(String[] args) {
        Money a = Money.rupiah(24_500);
        Money b = Money.rupiah(7_000).times(3.5);           // 3,5 kg cucian
        System.out.println("a = " + a + ", b = " + b);
        System.out.println("a == b: " + (a == b) + ", a.equals(b): " + a.equals(b)
                           + ", same hash: " + (a.hashCode() == b.hashCode()));

        Map<Money, Integer> ordersPerPrice = new HashMap<>();
        for (long price : new long[] {21_000, 24_500, 21_000, 70_000, 21_000}) {
            ordersPerPrice.merge(Money.rupiah(price), 1, Integer::sum);
        }
        System.out.println("orders per price: " + new TreeMap<>(ordersPerPrice));

        Set<NoHashMoney> noHash = new HashSet<>(List.of(new NoHashMoney(5000)));
        noHash.add(new NoHashMoney(5000));
        System.out.println("equals without hashCode, set size: " + noHash.size());

        MutableMoney key = new MutableMoney(5000);
        Set<MutableMoney> mutable = new HashSet<>();
        mutable.add(key);
        key.amount = 9000;                                  // diubah selagi berada di
dalam himpunan
        System.out.println("mutable key: contains " + mutable.contains(key) + ", size " +
mutable.size());

        check("value equality", a, b);
        check("hash agrees with equals", a.hashCode(), b.hashCode());
        check("plus makes a new object", "IDR 45,500",
a.plus(Money.rupiah(21_000)).toString());
        check("a is unchanged", "IDR 24,500", a.toString());
        check("three orders at 21,000", 3, ordersPerPrice.get(Money.rupiah(21_000)));
        check("record equality", new MoneyRecord("IDR", 5000), new MoneyRecord("IDR",
5000));
        try {
            a.plus(Money.of(10, "USD"));
            throw new AssertionError("no exception");
        } catch (IllegalArgumentException e) {
            check("currencies do not mix", "IDR and USD do not mix", e.getMessage());
        }
        check("the ghost key cannot be found", false, mutable.contains(key));
        System.out.println("All " + passed + " checks passed.");
    }
```

Kode 16.23 (Lanjutan) *Money.java: objek nilai imutabel dengan pembuatan yang tervalidasi, kesamaan nilai, kode hash yang konsisten, pengurutan, dan dua varian yang rusak.*

```
}
```

```
a = IDR 24,500, b = IDR 24,500
a == b: false, a.equals(b): true, same hash: true
orders per price: {IDR 21,000=3, IDR 24,500=1, IDR 70,000=1}
equals without hashCode, set size: 2
mutable key: contains false, size 1
PASS value equality
PASS hash agrees with equals
PASS plus makes a new object
PASS a is unchanged
PASS three orders at 21,000
PASS record equality
PASS currencies do not mix
PASS the ghost key cannot be found
All 8 checks passed.
```

Penyelesaian Python. **Kode 16.24** memakai *dataclass* beku, yang menuliskan `__eq__` dan `__hash__` dari *field*-nya dan melarang penugasan.

Kode 16.24 `money.py`: objek nilai yang sama sebagai *dataclass* beku, dan kedua varian yang rusak.

```python
# money.py - Soal 16.10: objek nilai imutabel dengan kesamaan, hash, dan urutan.
import dataclasses
import re
import sys
import unittest
from collections import Counter
from dataclasses import dataclass


@dataclass(frozen=True)                         # frozen: penugasan memunculkan galat
class Money:
    amount: int                                 # satuan utuh: rupiah tidak memakai sen
dalam keseharian
    currency: str = "IDR"

    def __post_init__(self):                    # setiap Money yang ada telah lolos
pemeriksaan ini
        if not re.fullmatch(r"[A-Z]{3}", self.currency or ""):
            raise ValueError(f"currency must be three capital letters: {self.currency}")
        if self.amount < 0:
            raise ValueError(f"amount must not be negative: {self.amount}")

    def __add__(self, other):
        """Mengembalikan Money BARU; objek ini tidak pernah berubah."""
        self._require_same_currency(other)
        return Money(self.amount + other.amount, self.currency)

    def times(self, kg):
        """Harga kg kilogram dengan harga per kilogram ini, dibulatkan ke satuan utuh."""
        return Money(round(self.amount * kg), self.currency)

    def __lt__(self, other):
        self._require_same_currency(other)
        return self.amount < other.amount

    def _require_same_currency(self, other):
        if self.currency != other.currency:
            raise ValueError(f"{self.currency} and {other.currency} do not mix")

    def __str__(self):
        return f"{self.currency} {self.amount:,}"


class NoHashMoney:
    """Menggoda tetapi keliru (1): __eq__ tanpa __hash__."""
```

Kode 16.24 (Lanjutan) *money.py: objek nilai yang sama sebagai dataclass beku, dan kedua varian yang rusak.*

```
    def __init__(self, amount):
        self.amount = amount

    def __eq__(self, other):
        return isinstance(other, NoHashMoney) and other.amount == self.amount


@dataclass(unsafe_hash=True)                    # dapat di-hash DAN dapat diubah: mencari perkara
class MutableMoney:
    """Menggoda tetapi keliru (2): kunci yang dapat berubah setelah masuk ke himpunan."""
    amount: int


class TestMoney(unittest.TestCase):
    def test_value_equality_and_hash(self):
        a, b = Money(24_500), Money(7_000).times(3.5)
        self.assertEqual(a, b)
        self.assertEqual(hash(a), hash(b))

    def test_plus_makes_a_new_object(self):
        a = Money(24_500)
        self.assertEqual("IDR 45,500", str(a + Money(21_000)))
        self.assertEqual("IDR 24,500", str(a))

    def test_cannot_change(self):
        with self.assertRaises(dataclasses.FrozenInstanceError):
            Money(1).amount = 2

    def test_currencies_do_not_mix(self):
        with self.assertRaises(ValueError):
            Money(24_500) + Money(10, "USD")

    def test_counting_by_price(self):
        counts = Counter(Money(p) for p in (21_000, 24_500, 21_000, 70_000, 21_000))
        self.assertEqual(3, counts[Money(21_000)])

    def test_ghost_key(self):
        key = MutableMoney(5000)
        keys = {key}
        key.amount = 9000
        self.assertNotIn(key, keys)
```

Kode 16.24 (Lanjutan) *money.py: objek nilai yang sama sebagai dataclass beku, dan kedua varian yang rusak.*

```
if __name__ == "__main__":
    a, b = Money(24_500), Money(7_000).times(3.5)       # 3,5 kg cucian
    print(f"a = {a}, b = {b}")
    print(f"a is b: {a is b}, a == b: {a == b}, same hash: {hash(a) == hash(b)}")
    counts = Counter(Money(p) for p in (21_000, 24_500, 21_000, 70_000, 21_000))
    print("orders per price:", {str(m): n for m, n in sorted(counts.items())})
    try:
        {NoHashMoney(5000)}
    except TypeError as e:
        print("__eq__ without __hash__:", e)
    key = MutableMoney(5000)
    keys = {key}
    key.amount = 9000                                   # diubah selagi berada di dalam
himpunan
    print(f"mutable key: contains {key in keys}, size {len(keys)}")
    sys.stdout.flush()
    unittest.main(verbosity=2)
```

```
a = IDR 24,500, b = IDR 24,500
a is b: False, a == b: True, same hash: True
orders per price: {'IDR 21,000': 3, 'IDR 24,500': 1, 'IDR 70,000': 1}
__eq__ without __hash__: unhashable type: 'NoHashMoney'
mutable key: contains False, size 1
test_cannot_change (__main__.TestMoney.test_cannot_change) ... ok
test_counting_by_price (__main__.TestMoney.test_counting_by_price) ... ok
test_currencies_do_not_mix (__main__.TestMoney.test_currencies_do_not_mix) ... ok
test_ghost_key (__main__.TestMoney.test_ghost_key) ... ok
test_plus_makes_a_new_object (__main__.TestMoney.test_plus_makes_a_new_object) ... ok
test_value_equality_and_hash (__main__.TestMoney.test_value_equality_and_hash) ... ok

----------------------------------------------------------------------
Ran 6 tests in 0.000s

OK
```

Baris demi baris. `Money.rupiah(7_000).times(3.5)` menghitung $7000 \times 3.5 = 24{,}500$ dan membangun objek *baru*; `a == b` bernilai `false` karena keduanya dua objek, sedangkan `a.equals(b)` bernilai `true` karena keduanya menyimpan nilai yang sama, dan kode *hash* keduanya sama karena `Objects.hash(currency, amount)` dihitung dari *field* yang sama dengan yang dibandingkan `equals`. `Math.addExact` melempar `ArithmeticException` alih-alih diam-diam berputar balik bila sebuah jumlah melampaui jangkauan `long`. `ordersPerPrice.merge(key, 1, Integer::sum)` mencacah pesanan per harga: ia menaruh 1 untuk kunci baru dan menambahkan 1 untuk kunci yang sudah ada, dan itu hanya berhasil karena harga yang sama adalah kunci yang sama. `String.format(Locale.ROOT, "%s %,d", ...)` mencetak jumlah dengan pemisah ribuan tanpa bergantung pada pengaturan bahasa komputer. *Record* `MoneyRecord` memperlihatkan seberapa banyak hal ini kini dituliskan Java untuk Anda: sebuah *konstruktor ringkas* untuk validasi, dan tidak ada lagi yang lain. Baris-baris yang dicetak juga memperlihatkan bahwa `compareTo` dipakai oleh `TreeMap`, yang mengurutkan harga dari 21.000 sampai 70.000.

Di Python, `__post_init__` berjalan setelah `__init__` yang dibangkitkan dan merupakan tempat untuk validasi. `round(self.amount * kg)` berbeda dari `Math.round` milik Java dalam satu hal: Python membulatkan setengah yang tepat ke bilangan *genap* terdekat (`round(2.5)` bernilai 2), sedangkan Java membulatkannya ke atas. Untuk uang, tuliskan aturan pembulatan dalam spesifikasi dan ujilah. `Counter` adalah *dictionary* yang mencacah; ia bergantung pada `__hash__` persis seperti `HashMap` bergantung pada `hashCode`. Penugasan ke *field* *dataclass* beku memunculkan `FrozenInstanceError`, dan sebuah uji memeriksanya.

Penyelesaian keliru yang menggoda. Ada dua, dan keluarannya memperlihatkan keduanya. `NoHashMoney` meng-*override* `equals` tetapi tidak `hashCode`; di Java, dua objek yang sama lalu memiliki kode *hash* identitas yang berbeda, jatuh ke ember yang berbeda, dan sebuah `HashSet` dengan senang hati menampung dua elemen yang "sama" (`set size: 2`). `javac -Xlint` bahkan memberi peringatan tentang kelas ini. Python menolak kekeliruan itu mentah-mentah: kelas yang mendefinisikan `__eq__` tanpa `__hash__` mendapat `__hash__ = None` dan sama sekali tidak dapat dimasukkan ke sebuah *set* (`unhashable type`). `MutableMoney` memiliki kedua metode itu, tetapi jumlahnya dapat berubah; setelah diubah selagi berada di dalam *set*, *set* itu mencari di ember milik kode *hash* yang *baru*, tidak menemukannya, dan melaporkan `contains false` sementara ukurannya masih 1: elemen hantu yang tidak dapat ditemukan maupun dihapus. Kunci *hash map* dan elemen *hash set* harus imutabel, dan inilah argumen praktis terkuat bagi objek nilai.

> **UJI DIRI.** Apakah `Money.rupiah(5000).equals(Money.of(5000, "USD"))` bernilai benar? Apakah `Money.rupiah(0)` diperbolehkan? Apa yang dilakukan `Money.rupiah(-1)`?
>
> *Jawaban.* Salah: mata uangnya berbeda, dan `equals` membandingkan kedua *field*. `rupiah(0)` diperbolehkan, karena hanya jumlah negatif yang ditolak. `rupiah(-1)` melempar `IllegalArgumentException("amount must not be negative: -1")`, sehingga tidak pernah ada `Money` yang tidak sah.

16.8 Soal Baru yang Dikerjakan: *Thread*

16.8.1 Soal 16.11 — Pencacah bersama, *race* yang diukur, dan penawarnya

> **SOAL.** Empat mesin cuci berjalan sebagai empat *thread*, dan masing-masing menambahkan 1 ke sebuah pencacah bersama untuk setiap muatan yang dicucinya. Ukurlah berapa pembaruan yang hilang pada pencacah yang tidak disinkronkan, tunjukkan cara membuat kehilangan itu terjadi setiap kali, dan perbaiki pencacah itu dengan tiga cara berbeda di Java dan satu cara di Python.

Dengan bahasa kita sendiri. Bab 14 menjelaskan *race condition* (kondisi balapan): `count++` tampak seperti satu langkah, padahal tiga langkah, yaitu membaca nilai, menambah satu, dan menulisnya kembali, dan dua *thread* dapat berselang-seling di antara langkah-langkah itu. **Gambar 16.7** memperlihatkan kecelakaan terkecil yang mungkin. Cacahan akhir yang diharapkan adalah

$$N = T \cdot k \tag{16.5}$$

untuk T *thread* yang masing-masing menambah k kali, dan setiap selang-seling seperti pada gambar membuat cacahan sebenarnya c lebih kecil, sehingga banyaknya *pembaruan yang hilang* adalah

$$L_{\text{hilang}} = N - c. \tag{16.6}$$

langkah	mesin 1	count di memori	mesin 2
1	baca count: 41	41	
2		41	baca count: 41
3	tulis 41 + 1 = 42	42	
4		42	tulis 41 + 1 = 42

Dua muatan dicuci, tetapi count hanya naik satu: satu pembaruan hilang.

Gambar 16.7 Pembaruan yang hilang. Kedua mesin membaca 41 sebelum salah satunya menulis, sehingga keduanya menulis 42: dua muatan dicuci, dan pencacah hanya naik satu.

Rancangan. Lima pencacah mengimplementasikan satu antarmuka, `Counter`, sehingga satu metode `run` dapat menguji semuanya. `UnsafeCounter` adalah *bug*-nya; `SlowUnsafeCounter` adalah *bug* yang sama dengan celah antara baca dan tulis dibiarkan terbuka selama 1 ms, yang mengubah kecelakaan langka menjadi kecelakaan pasti; `SynchronizedCounter`, `LockedCounter`, dan `AtomicCounter` adalah tiga penawarnya. Dengan $T = 4$ dan $k = 1{,}000{,}000$, Persamaan (16.5) menghasilkan $N = 4{,}000{,}000$.

Penyelesaian Java. Kode 16.25:

Kode 16.25 `WashCounter.java`: pencacah yang tidak aman, *race* yang sama dibuat pasti, dan tiga pencacah yang aman-*thread*, semuanya dijalankan oleh empat *thread* yang dilepas bersama.

```java
// WashCounter.java - Soal 16.11: pencacah bersama, race yang diukur, dan tiga cara memperbaikinya.
import java.util.concurrent.CountDownLatch;
import java.util.concurrent.atomic.AtomicLong;

public class WashCounter {

    static final int THREADS = 4;
    static final int LOADS_PER_THREAD = 1_000_000;

    interface Counter {
        void addLoad();

        long total();
    }

    /**
     * Menggoda tetapi keliru: count++ adalah baca, tambah, tulis, dan dua thread dapat berselang-seling.
     * volatile membuat setiap baca dan tulis menuju memori utama, tetapi TIDAK membuat ++ atomik.
     */
    static final class UnsafeCounter implements Counter {
        private volatile long count = 0;

        public void addLoad() {
            count++;
        }

        public long total() {
            return count;
        }
    }

    /** Bug yang sama dengan celah antara baca dan tulis dibiarkan terbuka selama 1 ms. */
    static final class SlowUnsafeCounter implements Counter {
        private long count = 0;

        public void addLoad() {
            long seen = count;                      // 1. baca
            try {
                Thread.sleep(1);                    //    ... mesin lain berjalan sekarang ...
            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
            }
            count = seen + 1;                       // 2. tulis kembali: dapat membatalkan kerja mesin lain
        }
```

Kode 16.25 (Lanjutan) *WashCounter.java: pencacah yang tidak aman, race yang sama dibuat pasti, dan tiga pencacah yang aman-thread, semuanya dijalankan oleh empat thread yang dilepas bersama.*

```
        public long total() {
            return count;
        }
    }

    /** Perbaikan 1: metode synchronized, sehingga hanya satu thread pada satu waktu berada
di dalam. */
    static final class SynchronizedCounter implements Counter {
        private long count = 0;

        public synchronized void addLoad() {
            count++;
        }

        public synchronized long total() {
            return count;
        }
    }

    /** Perbaikan 2: blok synchronized pada objek kunci privat yang tidak terlihat oleh
siapa pun. */
    static final class LockedCounter implements Counter {
        private final Object lock = new Object();
        private long count = 0;

        public void addLoad() {
            synchronized (lock) {
                count++;
            }
        }

        public long total() {
            synchronized (lock) {
                return count;
            }
        }
    }

    /** Perbaikan 3: variabel atomik, yang penambahannya dikerjakan perangkat keras dalam
satu langkah tak terbagi. */
    static final class AtomicCounter implements Counter {
        private final AtomicLong count = new AtomicLong();

        public void addLoad() {
            count.incrementAndGet();
        }
```

Kode 16.25 (Lanjutan) *WashCounter.java: pencacah yang tidak aman, race yang sama dibuat pasti, dan tiga pencacah yang aman-thread, semuanya dijalankan oleh empat thread yang dilepas bersama.*

```
        public long total() {
            return count.get();
        }
    }

    /** Memulai THREADS mesin cuci yang semuanya menambah satu pencacah; menunggu mereka
dengan join(). */
    static long run(Counter counter, int loadsPerThread) throws InterruptedException {
        Thread[] machines = new Thread[THREADS];
        CountDownLatch startGate = new CountDownLatch(1);
        for (int i = 0; i < THREADS; i++) {
            machines[i] = new Thread(() -> {
                try {
                    startGate.await();              // semua mesin mulai pada saat yang
sama
                } catch (InterruptedException e) {
                    return;
                }
                for (int k = 0; k < loadsPerThread; k++) {
                    counter.addLoad();
                }
            });
            machines[i].start();
        }
        startGate.countDown();
        for (Thread m : machines) {
            m.join();                               // tunggu sampai mesin ini selesai
        }
        return counter.total();
    }

    private static int passed = 0;

    private static void check(String name, Object expected, Object actual) {
        if (!java.util.Objects.equals(expected, actual)) {
            throw new AssertionError(name + ": expected " + expected + " but got " +
actual);
        }
        passed++;
        System.out.println("PASS " + name);
    }
```

Kode 16.25 (Lanjutan) *WashCounter.java: pencacah yang tidak aman, race yang sama dibuat pasti, dan tiga pencacah yang aman-thread, semuanya dijalankan oleh empat thread yang dilepas bersama.*

```
    public static void main(String[] args) throws InterruptedException {
        long expected = (long) THREADS * LOADS_PER_THREAD;
        System.out.println("expected total: " + expected);
        for (int trial = 1; trial <= 3; trial++) {
            long got = run(new UnsafeCounter(), LOADS_PER_THREAD);
            System.out.println("unsafe, trial " + trial + ": " + got + " (lost " +
(expected - got) + ")");
        }
        long slow = run(new SlowUnsafeCounter(), 50);
        System.out.println("slow unsafe: " + slow + " of " + THREADS * 50);
        Counter[] fixed = {new SynchronizedCounter(), new LockedCounter(), new
AtomicCounter()};
        for (Counter c : fixed) {
            long t0 = System.nanoTime();
            long got = run(c, LOADS_PER_THREAD);
            long ms = (System.nanoTime() - t0) / 1_000_000;
            System.out.println(c.getClass().getSimpleName() + ": " + got + " in " + ms + "
ms");
            check(c.getClass().getSimpleName() + " is exact", expected, got);
        }
        check("the widened window always loses updates", true, slow < THREADS * 50);
        System.out.println("All " + passed + " checks passed.");
    }
}
```

```
expected total: 4000000
unsafe, trial 1: 3992166 (lost 7834)
unsafe, trial 2: 3948364 (lost 51636)
unsafe, trial 3: 3939457 (lost 60543)
slow unsafe: 50 of 200
SynchronizedCounter: 4000000 in 63 ms
PASS SynchronizedCounter is exact
LockedCounter: 4000000 in 61 ms
PASS LockedCounter is exact
AtomicCounter: 4000000 in 43 ms
PASS AtomicCounter is exact
PASS the widened window always loses updates
All 4 checks passed.
```

Penyelesaian Python. **Kode 16.26** memuat pencacah yang tidak aman, yang lambat, dan kuncinya.

Kode 16.26 `wash_counter.py`: percobaan yang sama dalam Python.

```
# wash_counter.py - Soal 16.11: pencacah bersama, race yang diukur, dan kunci yang
memperbaikinya.
import sys
import threading
import time
import unittest

THREADS = 4
LOADS_PER_THREAD = 200_000


class UnsafeCounter:
    """Menggoda tetapi keliru: self.count += 1 adalah baca, tambah, tulis, dan thread dapat
berselang-seling."""

    def __init__(self):
        self.count = 0

    def add_load(self):
        self.count += 1

    def total(self):
        return self.count


class SlowUnsafeCounter(UnsafeCounter):
    """Bug yang sama dengan celah antara baca dan tulis dibiarkan terbuka selama 1 ms."""

    def add_load(self):
        seen = self.count                       # 1. baca
        time.sleep(0.001)                       #    ... mesin lain berjalan sekarang ...
        self.count = seen + 1                   # 2. tulis kembali: dapat membatalkan kerja
mesin lain


class LockedCounter:
    """Perbaikannya: kunci, sehingga hanya satu thread pada satu waktu berada di antara
baca dan tulis."""

    def __init__(self):
        self._lock = threading.Lock()
        self._count = 0

    def add_load(self):
        with self._lock:                        # diambil di sini, dilepas di akhir blok
            self._count += 1
```

Kode 16.26 (Lanjutan) *wash_counter.py: percobaan yang sama dalam Python.*

```
    def total(self):
        with self._lock:
            return self._count


def run(counter, loads_per_thread):
    """Memulai THREADS mesin cuci yang semuanya menambah satu pencacah; menunggu mereka
dengan join()."""
    start_gate = threading.Event()

    def machine():
        start_gate.wait()                       # semua mesin mulai pada saat yang sama
        for _ in range(loads_per_thread):
            counter.add_load()

    machines = [threading.Thread(target=machine) for _ in range(THREADS)]
    for m in machines:
        m.start()
    start_gate.set()
    for m in machines:
        m.join()                                # tunggu sampai mesin ini selesai
    return counter.total()


class TestCounters(unittest.TestCase):
    def test_locked_counter_is_exact(self):
        self.assertEqual(THREADS * 1000, run(LockedCounter(), 1000))

    def test_widened_window_always_loses_updates(self):
        self.assertLess(run(SlowUnsafeCounter(), 50), THREADS * 50)


if __name__ == "__main__":
    sys.setswitchinterval(1e-6)                 # sering-sering berganti thread: race lebih
cepat muncul
    expected = THREADS * LOADS_PER_THREAD
    print("expected total:", expected)
    for trial in (1, 2, 3):
        got = run(UnsafeCounter(), LOADS_PER_THREAD)
        print(f"unsafe, trial {trial}: {got} (lost {expected - got})")
    print("slow unsafe:", run(SlowUnsafeCounter(), 50), "of", THREADS * 50)
    t0 = time.perf_counter()
    got = run(LockedCounter(), LOADS_PER_THREAD)
    print(f"LockedCounter: {got} in {(time.perf_counter() - t0) * 1000:.0f} ms")
    sys.stdout.flush()
    unittest.main(verbosity=2)
```

```
expected total: 800000
unsafe, trial 1: 800000 (lost 0)
unsafe, trial 2: 800000 (lost 0)
unsafe, trial 3: 800000 (lost 0)
slow unsafe: 50 of 200
LockedCounter: 800000 in 189 ms
test_locked_counter_is_exact (__main__.TestCounters.test_locked_counter_is_exact) ... ok
test_widened_window_always_loses_updates
(__main__.TestCounters.test_widened_window_always_loses_updates) ... ok

----------------------------------------------------------------------
Ran 2 tests in 0.080s

OK
```

Baris demi baris. `run` membuat empat *thread*, membiarkan semuanya menunggu di gerbang mulai, membukanya, lalu memanggil `join()` pada masing-masing, yang menunggu sampai *thread*

itu selesai; baru sesudah itu `total()` dibaca. Pada keluaran Java, pencacah yang tidak aman kehilangan antara beberapa ribu sampai enam puluh ribu dari empat juta pembaruan dalam tiga percobaan: menurut Persamaan (16.6), percobaan pertama kehilangan $4{,}000{,}000 - 3{,}992{,}166 = 7{,}834$. Pada kali jalan lain atau mesin lain Anda mungkin melihat lebih banyak, lebih sedikit, atau sesekali tidak ada sama sekali: ketika sistem operasi kebetulan menjalankan *thread-thread* itu satu demi satu, tidak ada yang berselang-seling. Justru itulah yang membuat *race* begitu berbahaya. *Field*-nya dideklarasikan `volatile`, yang memaksa setiap baca dan tulis menuju memori utama dan mencegah kompilator menyimpan cacahan di register; banyak orang percaya bahwa `volatile` memperbaiki *race*, dan keluarannya memperlihatkan bahwa tidak, karena tiga langkah itu tetap tiga langkah.

`SlowUnsafeCounter` membaca cacahan, tidur 1 ms, lalu menulis nilai lama ditambah satu. Selama tidur itu, ketiga *thread* lain membaca nilai yang sama, sehingga pada setiap putaran keempatnya menulis hasil yang sama: tersisa 50 dari 200, setiap kali. Melebarkan celah *race* dengan sengaja adalah teknik pengujian yang nyata: teknik ini mengubah "gagal sebulan sekali" menjadi "gagal pada setiap kali uji dijalankan", sehingga perbaikannya dapat dibuktikan.

Penawarnya. `synchronized` pada sebuah metode berarti sebuah *thread* harus memiliki *kunci monitor* objek itu untuk masuk, sehingga paling banyak satu *thread* pada satu waktu yang menjalankan `addLoad` atau `total` pada objek itu. `LockedCounter` mengunci sebuah objek privat, yang merupakan mekanisme yang sama dengan satu kelebihan: tidak ada kode di luar kelas yang dapat mengambil kunci yang sama lalu mengganggu. `AtomicCounter` memakai `AtomicLong.incrementAndGet()`, yang dikerjakan prosesor sebagai satu operasi tak terbagi (gelung *compare-and-swap*), dan biasanya yang tercepat di antara ketiganya. Ketiganya mencapai tepat 4.000.000; waktu yang dicetak sesudahnya berubah-ubah antarjalan dan tidak perlu dibaca terlalu teliti.

Keluaran Python menyimpan sebuah kejutan, dan kejutan itu nyata: pencacah yang tidak aman *tidak kehilangan apa pun* dalam tiga percobaan, walaupun `sys.setswitchinterval(1e-6)` meminta interpreter berganti *thread* sesering mungkin. Sebabnya adalah detail implementasi CPython: interpreter hanya berganti *thread* pada titik-titik tertentu dalam *bytecode*, dan pada versi-versi sekarang baca dan tulis dari `self.count += 1` kebetulan tidak dipisahkan oleh titik semacam itu. Itu keberuntungan, bukan kebenaran: bahasanya tidak menjanjikannya, interpreter yang lebih lama dan lebih baru berperilaku berbeda, dan *build free-threaded* dari Python 3.13 ke atas, yang berjalan tanpa *global interpreter lock*, berselang-seling dengan bebas. Pencacah yang lambat membuktikan bahwa kodenya tetap salah: 50 dari 200, setiap kali. `with self._lock:` mengambil kunci di awal blok dan melepasnya di akhir, bahkan bila sebuah eksepsi muncul, dan membuat pencacah itu tepat.

Penyelesaian keliru yang menggoda. Kesimpulan yang menggoda dari keluaran Python adalah "`+=` bersifat atomik di Python, jadi tidak perlu kunci". Uji yang lolos seribu kali tidak membuktikan bahwa tidak ada *race*; uji itu mungkin hanya membuktikan bahwa mesin ini, interpreter ini, dan beban ini tidak pernah menghasilkan selang-seling yang buruk. Penalaran yang benar bersifat struktural: jika dua *thread* membaca dan menulis variabel yang sama dan sedikitnya satu di antaranya menulis, akses-akses itu harus dilindungi, entah sebuah uji pernah gagal atau tidak.

> **UJI DIRI.** Pada `SlowUnsafeCounter`, dengan 4 *thread* dan masing-masing 50 muatan, hasilnya 50. Jelaskan mengapa hasilnya *tepat* 50, dan berapa hasilnya dengan 2 *thread*.
>
> *Jawaban.* Keempat *thread* mulai bersama, membaca nilai yang sama, tidur, lalu menulis nilai itu ditambah satu, sehingga setiap putaran hanya menambahkan 1 alih-alih 4; setelah 50

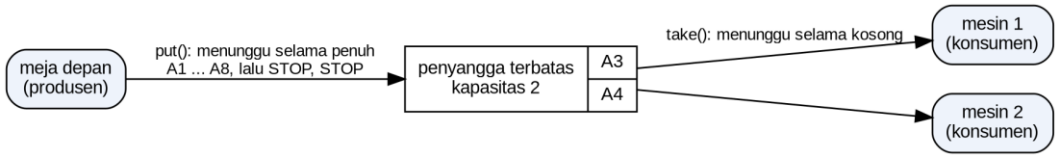
putaran cacahannya 50. (Putaran-putaran itu tetap sejajar karena setiap tidur berlangsung sama lamanya.) Dengan 2 *thread*, hasilnya juga sekitar 50, kini dari 100 yang diharapkan.

16.8.2 Soal 16.12 — Lini cuci produsen–konsumen

SOAL. Meja depan Profitina Laundry menerima pesanan dan menaruhnya di sebuah rak yang paling banyak memuat dua pesanan; dua mesin cuci mengambil pesanan dari rak dan mencucinya. Meja depan harus menunggu ketika rak penuh dan mesin harus menunggu ketika rak kosong. Tulislah rak itu dengan tangan menggunakan `wait` dan `notifyAll` (Python: sebuah `Condition`), lalu dengan antrean pemblokir dari pustaka. Hentikan mesin-mesin dengan rapi ketika pesanan hari itu habis, dan periksa bahwa setiap pesanan dicuci tepat satu kali.

Dengan bahasa kita sendiri. Inilah soal *produsen–konsumen* dari Bab 14 [4]: satu *thread* atau lebih menghasilkan butir, *thread* lain menghabiskannya, dan sebuah *penyangga terbatas* (*bounded buffer*) di antara mereka meredam perbedaan kecepatan sambil menjaga memori tetap terbatas. Kesulitannya adalah menunggu dengan benar: *thread* yang mendapati penyangga penuh atau kosong harus tidur tanpa memegang kunci, dan harus dibangunkan ketika keadaannya berubah.

Rancangan. **Gambar 16.8** memperlihatkan lininya. Untuk menghentikan para konsumen, produsen menaruh satu *pil racun* (*poison pill*), yaitu nilai khusus `STOP`, di rak untuk setiap mesin setelah pesanan sungguhan yang terakhir; mesin yang mengambil pil itu selesai bekerja. Penyangga buatan tangan adalah sebuah *monitor*: semua metodenya `synchronized` pada objek penyangga, penyangga yang penuh atau kosong membuat pemanggilnya menjalankan `wait()`, yang melepas kunci selama tidur, dan setiap perubahan memanggil `notifyAll()`.



Gambar 16.8 Lini cuci: satu produsen, penyangga terbatas berkapasitas 2, dan dua konsumen. Dua pil `STOP`, satu per mesin, mengakhiri hari kerja.

Penyelesaian Java. **Kode 16.27**:

Kode 16.27 `WashLine.java`: penyangga terbatas buatan tangan, lini yang sama dengan `ArrayBlockingQueue`, pil racun, dan penyangga yang memakai `if` alih-alih `while`.

```java
// WashLine.java - Soal 16.12: produsen-konsumen dengan penyangga terbatas, buatan tangan
dan dengan BlockingQueue.
import java.util.ArrayDeque;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Deque;
import java.util.HashSet;
import java.util.List;
import java.util.concurrent.ArrayBlockingQueue;
import java.util.concurrent.BlockingQueue;

public class WashLine {

    static final String STOP = "STOP";              // "pil racun": tidak ada pesanan lagi
    static final int MACHINES = 2;

    interface Buffer {
        void put(String order) throws InterruptedException;

        String take() throws InterruptedException;
    }

    /** Penyangga terbatas buatan tangan: monitor dengan wait() dan notifyAll(). */
    static final class OrderBuffer implements Buffer {
        private final Deque<String> items = new ArrayDeque<>();
        private final int capacity;
        private int mostEverWaiting = 0;

        OrderBuffer(int capacity) {
            this.capacity = capacity;
        }

        @Override
        public synchronized void put(String order) throws InterruptedException {
            while (items.size() == capacity) {
                wait();                             // penuh: meja depan menunggu tempat
kosong
            }
            items.addLast(order);
            mostEverWaiting = Math.max(mostEverWaiting, items.size());
            notifyAll();                            // bangunkan mesin mana pun yang
menunggu pekerjaan
        }
```

Kode 16.27 (Lanjutan) *WashLine.java: penyangga terbatas buatan tangan, lini yang sama dengan ArrayBlockingQueue, pil racun, dan penyangga yang memakai if alih-alih while.*

```
        @Override
        public synchronized String take() throws InterruptedException {
            while (items.isEmpty()) {
                wait();                              // kosong: mesin menunggu pekerjaan
            }
            String order = items.removeFirst();
            notifyAll();                             // bangunkan meja depan jika ia
menunggu tempat
            return order;
        }

        synchronized int mostEverWaiting() {
            return mostEverWaiting;
        }
    }

    /** Versi pustaka: ArrayBlockingQueue sudah merupakan penyangga terbatas. */
    static Buffer libraryBuffer(int capacity) {
        BlockingQueue<String> queue = new ArrayBlockingQueue<>(capacity);
        return new Buffer() {
            @Override
            public void put(String order) throws InterruptedException {
                queue.put(order);                    // menunggu selama antrean penuh
            }

            @Override
            public String take() throws InterruptedException {
                return queue.take();                 // menunggu selama antrean kosong
            }
        };
    }

    /** Satu meja depan (produsen) dan MACHINES mesin cuci (konsumen). */
    static List<String> runLine(Buffer buffer, int orders) throws InterruptedException {
        List<String> washed = Collections.synchronizedList(new ArrayList<>());
        Thread desk = new Thread(() -> {
            try {
                for (int i = 1; i <= orders; i++) {
                    buffer.put("A" + i);
                }
                for (int m = 0; m < MACHINES; m++) {
                    buffer.put(STOP);                // satu pil untuk setiap mesin
                }
            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
            }
        });
```

Kode 16.27 (Lanjutan) *WashLine.java: penyangga terbatas buatan tangan, lini yang sama dengan ArrayBlockingQueue, pil racun, dan penyangga yang memakai if alih-alih while.*

```
        List<Thread> threads = new ArrayList<>(List.of(desk));
        for (int m = 0; m < MACHINES; m++) {
            threads.add(new Thread(() -> {
                try {
                    while (true) {
                        String order = buffer.take();
                        if (order.equals(STOP)) {
                            return;                 // menelan pil: berhenti
                        }
                        Thread.sleep(1);            // mencuci memerlukan sedikit waktu
                        washed.add(order);
                    }
                } catch (InterruptedException e) {
                    Thread.currentThread().interrupt();
                }
            }));
        }
        threads.forEach(Thread::start);
        for (Thread t : threads) {
            t.join();
        }
        return washed;
    }

    /** Menggoda tetapi keliru: "if" alih-alih "while" di sekitar wait(). */
    static final class IfBuffer {
        private final Deque<String> items = new ArrayDeque<>();
        private int waiting = 0;

        synchronized void put(String order) {
            items.addLast(order);
            notifyAll();
        }

        synchronized String take() throws InterruptedException {
            if (items.isEmpty()) {                  // diperiksa SEKALI, sebelum tidur
                waiting++;
                wait();
                waiting--;
            }
            return items.removeFirst();             // list mungkin sudah kosong lagi
sekarang
        }
```

Kode 16.27 (Lanjutan) *WashLine.java: penyangga terbatas buatan tangan, lini yang sama dengan ArrayBlockingQueue, pil racun, dan penyangga yang memakai if alih-alih while.*

```
        synchronized int waiting() {
            return waiting;
        }
    }

    /** Dua mesin menunggu pada IfBuffer yang kosong; satu pesanan datang; apa yang didapat
masing-masing? */
    static List<String> ifBufferRace() throws InterruptedException {
        IfBuffer buffer = new IfBuffer();
        List<String> results = Collections.synchronizedList(new ArrayList<>());
        List<Thread> machines = new ArrayList<>();
        for (int m = 0; m < 2; m++) {
            Thread t = new Thread(() -> {
                try {
                    results.add(buffer.take());
                } catch (RuntimeException e) {
                    results.add(e.getClass().getSimpleName());
                } catch (InterruptedException e) {
                    results.add("interrupted");
                }
            });
            machines.add(t);
            t.start();
        }
        while (buffer.waiting() < 2) {              // sampai kedua mesin berada di dalam
wait()
            Thread.sleep(5);
        }
        buffer.put("A1");                           // satu pesanan, dan notifyAll()
membangunkan keduanya
        for (Thread t : machines) {
            t.join(2000);
        }
        List<String> sorted = new ArrayList<>(results);
        Collections.sort(sorted);
        return sorted;
    }

    private static int passed = 0;

    private static void check(String name, Object expected, Object actual) {
        if (!java.util.Objects.equals(expected, actual)) {
            throw new AssertionError(name + ": expected " + expected + " but got " +
actual);
        }
        passed++;
        System.out.println("PASS " + name);
    }
```

Kode 16.27 (Lanjutan) *WashLine.java: penyangga terbatas buatan tangan, lini yang sama dengan ArrayBlockingQueue, pil racun, dan penyangga yang memakai if alih-alih while.*

```
    public static void main(String[] args) throws InterruptedException {
        OrderBuffer handMade = new OrderBuffer(2);
        List<String> washed = runLine(handMade, 8);
        System.out.println("hand-made buffer: " + washed.size() + " orders washed, "
                           + new HashSet<>(washed).size() + " different, most ever waiting
"
                           + handMade.mostEverWaiting());
        List<String> washed2 = runLine(libraryBuffer(2), 8);
        System.out.println("library buffer  : " + washed2.size() + " orders washed, "
                           + new HashSet<>(washed2).size() + " different");
        List<String> race = ifBufferRace();
        System.out.println("if instead of while: " + race);

        check("every order washed once", 8, new HashSet<>(washed).size());
        check("no order washed twice", 8, washed.size());
        check("capacity respected", true, handMade.mostEverWaiting() <= 2);
        check("library buffer too", 8, new HashSet<>(washed2).size());
        check("the if-buffer hands out an order that is gone",
              List.of("A1", "NoSuchElementException"), race);
        System.out.println("All " + passed + " checks passed.");
    }
}
```

```
hand-made buffer: 8 orders washed, 8 different, most ever waiting 2
library buffer  : 8 orders washed, 8 different
if instead of while: [A1, NoSuchElementException]
PASS every order washed once
PASS no order washed twice
PASS capacity respected
PASS library buffer too
PASS the if-buffer hands out an order that is gone
All 5 checks passed.
```

Penyelesaian Python. **Kode 16.28** memakai `threading.Condition`, yang merupakan kunci sekaligus ruang tunggu dalam satu objek, dan membungkus `queue.Queue(maxsize=2)` sebagai penyangga pustaka.

Kode 16.28 `wash_line.py`: lini cuci yang sama dalam Python.

```
# wash_line.py - Soal 16.12: produsen-konsumen dengan penyangga terbatas, buatan tangan dan
dengan queue.Queue.
import queue
import sys
import threading
import time
import unittest
from collections import deque

STOP = "STOP"                                   # "pil racun": tidak ada pesanan lagi
MACHINES = 2


class OrderBuffer:
    """Penyangga terbatas buatan tangan: Condition dengan wait() dan notify_all()."""

    def __init__(self, capacity):
        self._items = deque()
        self._capacity = capacity
        self._changed = threading.Condition()   # kunci ditambah ruang tunggu
        self.most_ever_waiting = 0

    def put(self, order):
        with self._changed:
            while len(self._items) == self._capacity:
                self._changed.wait()            # penuh: meja depan menunggu tempat kosong
            self._items.append(order)
            self.most_ever_waiting = max(self.most_ever_waiting, len(self._items))
            self._changed.notify_all()          # bangunkan mesin mana pun yang menunggu
pekerjaan

    def take(self):
        with self._changed:
            while not self._items:
                self._changed.wait()            # kosong: mesin menunggu pekerjaan
            order = self._items.popleft()
            self._changed.notify_all()          # bangunkan meja depan jika ia menunggu
tempat
            return order


class LibraryBuffer:
    """Versi pustaka: queue.Queue sudah merupakan penyangga terbatas."""

    def __init__(self, capacity):
        self._queue = queue.Queue(maxsize=capacity)
```

Kode 16.28 (Lanjutan) *wash_line.py: lini cuci yang sama dalam Python.*

```
    def put(self, order):
        self._queue.put(order)                  # menunggu selama antrean penuh

    def take(self):
        return self._queue.get()                # menunggu selama antrean kosong


def run_line(buffer, orders):
    """Satu meja depan (produsen) dan MACHINES mesin cuci (konsumen)."""
    washed, lock = [], threading.Lock()

    def desk():
        for i in range(1, orders + 1):
            buffer.put(f"A{i}")
        for _ in range(MACHINES):
            buffer.put(STOP)                    # satu pil untuk setiap mesin

    def machine():
        while True:
            order = buffer.take()
            if order == STOP:
                return                          # menelan pil: berhenti
            time.sleep(0.001)                   # mencuci memerlukan sedikit waktu
            with lock:
                washed.append(order)

    threads = [threading.Thread(target=desk)] + [threading.Thread(target=machine)
                                                 for _ in range(MACHINES)]
    for t in threads:
        t.start()
    for t in threads:
        t.join()
    return washed


class IfBuffer:
    """Menggoda tetapi keliru: "if" alih-alih "while" di sekitar wait()."""

    def __init__(self):
        self._items = deque()
        self._changed = threading.Condition()
        self.waiting = 0
```

Kode 16.28 (Lanjutan) *wash_line.py: lini cuci yang sama dalam Python.*

```
    def put(self, order):
        with self._changed:
            self._items.append(order)
            self._changed.notify_all()

    def take(self):
        with self._changed:
            if not self._items:                 # diperiksa SEKALI, sebelum tidur
                self.waiting += 1
                self._changed.wait()
                self.waiting -= 1
            return self._items.popleft()        # deque mungkin sudah kosong lagi sekarang


def if_buffer_race():
    """Dua mesin menunggu pada IfBuffer yang kosong; satu pesanan datang; apa yang didapat
masing-masing?"""
    buffer, results = IfBuffer(), []

    def machine():
        try:
            results.append(buffer.take())
        except IndexError as e:
            results.append(type(e).__name__)

    machines = [threading.Thread(target=machine) for _ in range(2)]
    for m in machines:
        m.start()
    while buffer.waiting < 2:                   # sampai kedua mesin berada di dalam wait()
        time.sleep(0.005)
    buffer.put("A1")                            # satu pesanan, dan notify_all()
membangunkan keduanya
    for m in machines:
        m.join(2)
    return sorted(results)


class TestWashLine(unittest.TestCase):
    def test_hand_made_buffer(self):
        buffer = OrderBuffer(2)
        washed = run_line(buffer, 8)
        self.assertEqual(8, len(washed))
        self.assertEqual(8, len(set(washed)))
        self.assertLessEqual(buffer.most_ever_waiting, 2)

    def test_library_buffer(self):
        self.assertEqual(8, len(set(run_line(LibraryBuffer(2), 8))))
```

Kode 16.28 (Lanjutan) *wash_line.py: lini cuci yang sama dalam Python.*

```
    def test_if_buffer_hands_out_an_order_that_is_gone(self):
        self.assertEqual(["A1", "IndexError"], if_buffer_race())


if __name__ == "__main__":
    buffer = OrderBuffer(2)
    washed = run_line(buffer, 8)
    print(f"hand-made buffer: {len(washed)} orders washed, {len(set(washed))} different, "
          f"most ever waiting {buffer.most_ever_waiting}")
    washed2 = run_line(LibraryBuffer(2), 8)
    print(f"library buffer  : {len(washed2)} orders washed, {len(set(washed2))} different")
    print("if instead of while:", if_buffer_race())
    sys.stdout.flush()
    unittest.main(verbosity=2)
```

```
hand-made buffer: 8 orders washed, 8 different, most ever waiting 2
library buffer  : 8 orders washed, 8 different
if instead of while: ['A1', 'IndexError']
test_hand_made_buffer (__main__.TestWashLine.test_hand_made_buffer) ... ok
test_if_buffer_hands_out_an_order_that_is_gone
(__main__.TestWashLine.test_if_buffer_hands_out_an_order_that_is_gone) ... ok
test_library_buffer (__main__.TestWashLine.test_library_buffer) ... ok

----------------------------------------------------------------------
Ran 3 tests in 0.014s

OK
```

Baris demi baris. Di `put`, `while (items.size() == capacity) wait();` adalah jantung monitor itu. `wait()` melakukan tiga hal secara atomik: melepas kunci penyangga, menidurkan *thread*, dan, ketika *thread* dibangunkan, mengambil kembali kunci itu sebelum kembali. `while` memeriksa ulang kondisinya setelah setiap kali bangun, karena dua alasan: *thread* lain mungkin telah mengubah penyangga di antara pemberitahuan dan saat *thread* ini mendapatkan kembali kuncinya, dan Java memperbolehkan *spurious wake-up*, yaitu kembalinya `wait()` tanpa pemberitahuan sama sekali. `notifyAll()` membangunkan setiap *thread* yang menunggu; `notify()` hanya membangunkan satu, yang mungkin *thread* dari jenis yang salah (meja depan, padahal yang diperlukan sebuah mesin), sehingga lini itu dapat macet. `mostEverWaiting` mencatat banyaknya pesanan terbesar yang pernah ada di rak, dan pemeriksaannya memastikan kapasitas tidak pernah terlampaui. Mesin-mesin tidur 1 ms per pesanan untuk meniru proses mencuci, dan itu membuat meja depan sempat memenuhi rak. Meja depan menaruh delapan pesanan lalu dua pil; setiap mesin mengulang `take()` sampai mengambil sebuah pil. `new HashSet<>(washed).size()` bernilai 8 dan `washed.size()` bernilai 8: tidak ada pesanan yang hilang dan tidak ada yang dicuci dua kali.

`ArrayBlockingQueue` mengerjakan semua ini di dalam pustaka: `put` menunggu selama antrean penuh dan `take` menunggu selama antrean kosong. Dalam program sungguhan, pakailah itu (atau `queue.Queue` Python); tulislah monitor sendiri hanya untuk memahaminya, seperti di sini.

Penyelesaian keliru yang menggoda. `IfBuffer` menulis `if (items.isEmpty()) wait();`, memeriksa kondisinya sekali saja. Skenario di `ifBufferRace` disusun agar deterministik: kedua mesin memanggil `take()` pada penyangga kosong lalu menunggu; *thread* utama memeriksa `waiting()` berulang-ulang sampai keduanya berada di dalam `wait()`, lalu menaruh satu pesanan, dan `notifyAll()` membangunkan keduanya. Mesin pertama yang mendapat kunci mengambil A1. Mesin kedua kembali dari `wait()`, *tidak* memeriksa lagi, dan memanggil `removeFirst()` pada *deque* yang kosong: `NoSuchElementException` di Java, `IndexError` di Python. Hasil terurut `[A1,`

`NoSuchElementException]` sama pada setiap kali jalan. Selalu menunggulah di dalam gelung `while`.

> **UJI DIRI.** Meja depan menaruh satu pil `STOP` untuk setiap dari dua mesin. Apa yang terjadi jika ia hanya menaruh satu?
>
> *Jawaban.* Satu mesin mengambil pil itu dan berhenti; mesin yang lain mendapati rak kosong dan menunggu selamanya, sehingga `t.join()` di `runLine` tidak pernah kembali dan program menggantung. Banyaknya pil harus sama dengan banyaknya konsumen. (Alternatifnya adalah penanda bersama "tutup" yang diperiksa `take` dalam gelung `while`-nya, dengan `notifyAll` ketika penanda itu dipasang.)

16.9 Soal Baru yang Dikerjakan: Protokol Mungil Buatan Kita Sendiri

16.9.1 Soal 16.13 — LAUNDRY/1, protokol permintaan–balasan

> **SOAL.** Rancang dan implementasikan protokol teks kecil, `LAUNDRY/1`, yang dengannya telepon seorang kurir dapat menanyakan harga kepada Profitina Laundry. Setelah tersambung, server menyapa dengan `LAUNDRY/1 READY`. Klien boleh mengirim `SERVICES`, `PRICE <service> <kg>`, dan `QUIT`; setiap permintaan mendapat tepat satu baris balasan yang diawali `OK` atau `ERR`. Tulislah server, kelas klien, dan uji termasuk sepuluh klien sekaligus.

Dengan bahasa kita sendiri. Protokol *knock-knock* adalah percakapan yang dipimpin server. Kebanyakan protokol sungguhan justru sebaliknya: klien bertanya, server menjawab, dan setiap permintaan berdiri sendiri. Protokol *permintaan–balasan* semacam itu jauh lebih mudah dirancang, diuji, dan diperluas, karena server tidak memerlukan keadaan percakapan: seluruh protokol dapat berupa *fungsi murni* dari satu baris permintaan ke satu baris balasan.

Rancangan. **Tabel 16.9** adalah spesifikasinya. Kata status di awal setiap balasan memungkinkan klien membedakan keberhasilan dari kegagalan tanpa memahami sisanya; pesan sesudah `ERR` ditujukan bagi manusia.

Tabel 16.9 Protokol LAUNDRY/1. Kata perintah tidak membedakan huruf besar dan kecil; setiap baris permintaan mendapat tepat satu baris balasan.

Permintaan	Balasan bila berhasil	Balasan bila gagal
(saat tersambung)	`LAUNDRY/1 READY`	tidak ada
`SERVICES`	`OK EXPRESS=14000 IRON=5000 WASH=7000`	tidak ada
`PRICE <service> <kg>`	`OK <harga dalam rupiah>`	`ERR usage: ...`, `ERR unknown service ...`, `ERR weight is not a number: ...`, `ERR weight must be in (0, 50] kg`
`QUIT`	`OK BYE`, lalu server menutup koneksi	tidak ada
selain itu	tidak ada	`ERR unknown command ...`

Penyelesaian Java. **Kode 16.29** memuat fungsi murni `reply`, server, klien, dan sebuah klien dengan kekeliruan jaringan yang klasik.

Kode 16.29 `LaundryProtocol.java`: protokol sebagai fungsi murni, server banyak-klien, klien dengan batas waktu balasan, dan klien yang lupa melakukan *flush*.

```java
// LaundryProtocol.java - Soal 16.13: protokol permintaan-balasan yang mungil, server-nya, kliennya.
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.net.ServerSocket;
import java.net.Socket;
import java.net.SocketTimeoutException;
import java.nio.charset.StandardCharsets;
import java.util.ArrayList;
import java.util.List;
import java.util.Locale;
import java.util.Map;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.Future;

public class LaundryProtocol {

    static final String GREETING = "LAUNDRY/1 READY";
    static final Map<String, Long> PRICE_PER_KG = Map.of("WASH", 7000L, "IRON", 5000L, "EXPRESS", 14000L);

    /** Seluruh protokol sebagai satu fungsi murni: satu baris permintaan masuk, satu baris balasan keluar. */
    static String reply(String request) {
        String[] words = request.trim().split("\\s+");
        String command = words[0].toUpperCase(Locale.ROOT);
        switch (command) {
            case "SERVICES":
                return "OK EXPRESS=14000 IRON=5000 WASH=7000";
            case "QUIT":
                return "OK BYE";
            case "PRICE":
                if (words.length != 3) {
                    return "ERR usage: PRICE <service> <kg>";
                }
                Long rate = PRICE_PER_KG.get(words[1].toUpperCase(Locale.ROOT));
                if (rate == null) {
                    return "ERR unknown service " + words[1];
                }
                double kg;
                try {
                    kg = Double.parseDouble(words[2]);
                } catch (NumberFormatException e) {
                    return "ERR weight is not a number: " + words[2];
                }
```

Kode 16.29 (Lanjutan) *LaundryProtocol.java: protokol sebagai fungsi murni, server banyak-klien, klien dengan batas waktu balasan, dan klien yang lupa melakukan flush.*

```
                if (kg <= 0 || kg > 50) {
                    return "ERR weight must be in (0, 50] kg";
                }
                return "OK " + Math.round(rate * kg);
            default:
                return "ERR unknown command " + command;
        }
    }

    /** Server: menyapa, lalu menjawab setiap baris dengan tepat satu baris. */
    static final class Server {
        private final ExecutorService workers = Executors.newCachedThreadPool();
        private ServerSocket listener;

        int start() throws IOException {
            listener = new ServerSocket(0);
            Thread acceptor = new Thread(() -> {
                try {
                    while (true) {
                        Socket c = listener.accept();
                        workers.execute(() -> serve(c));
                    }
                } catch (IOException e) {
                    // listener ditutup
                }
            });
            acceptor.setDaemon(true);
            acceptor.start();
            return listener.getLocalPort();
        }

        private void serve(Socket c) {
            try (c;
                 BufferedReader in = new BufferedReader(new InputStreamReader(
                         c.getInputStream(), StandardCharsets.UTF_8));
                 PrintWriter out = new PrintWriter(new OutputStreamWriter(
                         c.getOutputStream(), StandardCharsets.UTF_8), true)) {
                out.println(GREETING);
                String line;
                while ((line = in.readLine()) != null) {
                    String answer = line.isBlank() ? "ERR empty request" : reply(line);
                    out.println(answer);
                    if (answer.equals("OK BYE")) {
                        return;
                    }
                }
```

Kode 16.29 (Lanjutan) *LaundryProtocol.java: protokol sebagai fungsi murni, server banyak-klien, klien dengan batas waktu balasan, dan klien yang lupa melakukan flush.*

```
            } catch (IOException e) {
                // klien menghilang; tidak ada yang perlu dibereskan selain soket
            }
        }

        void stop() throws IOException {
            listener.close();
            workers.shutdown();
        }
    }

    /** Klien: satu metode per permintaan, dengan batas waktu pada setiap balasan. */
    static class Client implements AutoCloseable {
        protected final Socket socket;
        protected final BufferedReader in;
        protected PrintWriter out;

        Client(int port) throws IOException {
            socket = new Socket("localhost", port);
            socket.setSoTimeout(500);
            in = new BufferedReader(new InputStreamReader(socket.getInputStream(),
StandardCharsets.UTF_8));
            out = new PrintWriter(new OutputStreamWriter(socket.getOutputStream(),
StandardCharsets.UTF_8), true);
            String hello = in.readLine();
            if (!GREETING.equals(hello)) {
                throw new IOException("not a LAUNDRY/1 server: " + hello);
            }
        }

        String request(String line) throws IOException {
            out.println(line);
            try {
                return in.readLine();
            } catch (SocketTimeoutException e) {
                return "(no reply within 500 ms)";
            }
        }

        @Override
        public void close() throws IOException {
            socket.close();
        }
    }
```

Kode 16.29 (Lanjutan) *LaundryProtocol.java: protokol sebagai fungsi murni, server banyak-klien, klien dengan batas waktu balasan, dan klien yang lupa melakukan flush.*

```
    /** Menggoda tetapi keliru: PrintWriter tanpa autoflush menahan permintaan di
penyangganya. */
    static final class NoFlushClient extends Client {
        NoFlushClient(int port) throws IOException {
            super(port);
            out = new PrintWriter(new OutputStreamWriter(socket.getOutputStream(),
StandardCharsets.UTF_8));
        }
    }

    private static int passed = 0;

    private static void check(String name, Object expected, Object actual) {
        if (!java.util.Objects.equals(expected, actual)) {
            throw new AssertionError(name + ": expected " + expected + " but got " +
actual);
        }
        passed++;
        System.out.println("PASS " + name);
    }

    public static void main(String[] args) throws Exception {
        Server server = new Server();
        int port = server.start();
        List<String> script = List.of("SERVICES", "PRICE wash 3.5", "price EXPRESS 2",
                                      "PRICE DRYCLEAN 1", "PRICE WASH abc", "PRICE IRON
80", "HELLO", "QUIT");
        List<String> replies = new ArrayList<>();
        try (Client c = new Client(port)) {
            for (String request : script) {
                String answer = c.request(request);
                replies.add(answer);
                System.out.printf("%-18s -> %s%n", request, answer);
            }
        }
        ExecutorService crowd = Executors.newFixedThreadPool(10);
        List<Future<String>> prices = new ArrayList<>();
        for (int i = 1; i <= 10; i++) {
            String request = "PRICE WASH " + i;
            prices.add(crowd.submit(() -> {
                try (Client c = new Client(port)) {
                    return c.request(request);
                }
            }));
        }
        long sum = 0;
        for (Future<String> f : prices) {
            sum += Long.parseLong(f.get().substring(3));
        }
        crowd.shutdown();
```

Kode 16.29 (Lanjutan) *LaundryProtocol.java: protokol sebagai fungsi murni, server banyak-klien, klien dengan batas waktu balasan, dan klien yang lupa melakukan flush.*

```
        System.out.println("ten clients at once, total of their prices: " + sum);
        try (Client wrong = new NoFlushClient(port)) {
            String answer = wrong.request("PRICE WASH 1");
            System.out.println("client without flush -> " + answer);
            check("missing flush means no reply", "(no reply within 500 ms)", answer);
        }
        server.stop();

        check("price of 3.5 kg washing", "OK 24500", replies.get(1));
        check("command words ignore case", "OK 28000", replies.get(2));
        check("unknown service", "ERR unknown service DRYCLEAN", replies.get(3));
        check("bad weight", "ERR weight is not a number: abc", replies.get(4));
        check("unknown command", "ERR unknown command HELLO", replies.get(6));
        check("sum of 1..10 kg at 7000", 385_000L, sum);
        System.out.println("All " + passed + " checks passed.");
    }
}
```

```
SERVICES           -> OK EXPRESS=14000 IRON=5000 WASH=7000
PRICE wash 3.5     -> OK 24500
price EXPRESS 2    -> OK 28000
PRICE DRYCLEAN 1   -> ERR unknown service DRYCLEAN
PRICE WASH abc     -> ERR weight is not a number: abc
PRICE IRON 80      -> ERR weight must be in (0, 50] kg
HELLO              -> ERR unknown command HELLO
QUIT               -> OK BYE
ten clients at once, total of their prices: 385000
client without flush -> (no reply within 500 ms)
PASS missing flush means no reply
PASS price of 3.5 kg washing
PASS command words ignore case
PASS unknown service
PASS bad weight
PASS unknown command
PASS sum of 1..10 kg at 7000
All 7 checks passed.
```

Penyelesaian Python. Kode 16.30:

Kode 16.30 `laundry_protocol.py`: protokol, server, dan klien yang sama dalam Python.

```python
# laundry_protocol.py - Soal 16.13: protokol permintaan-balasan yang mungil, server-nya, kliennya.
import socket
import sys
import threading
import unittest
from concurrent.futures import ThreadPoolExecutor

GREETING = "LAUNDRY/1 READY"
PRICE_PER_KG = {"WASH": 7000, "IRON": 5000, "EXPRESS": 14000}


def reply(request):
    """Seluruh protokol sebagai satu fungsi murni: satu baris permintaan masuk, satu baris balasan keluar."""
    words = request.split()
    command = words[0].upper()
    if command == "SERVICES":
        return "OK EXPRESS=14000 IRON=5000 WASH=7000"
    if command == "QUIT":
        return "OK BYE"
    if command == "PRICE":
        if len(words) != 3:
            return "ERR usage: PRICE <service> <kg>"
        rate = PRICE_PER_KG.get(words[1].upper())
        if rate is None:
            return f"ERR unknown service {words[1]}"
        try:
            kg = float(words[2])
        except ValueError:
            return f"ERR weight is not a number: {words[2]}"
        if not 0 < kg <= 50:
            return "ERR weight must be in (0, 50] kg"
        return f"OK {round(rate * kg)}"
    return f"ERR unknown command {command}"


class Server:
    """Server: menyapa, lalu menjawab setiap baris dengan tepat satu baris."""

    def start(self):
        self._listener = socket.create_server(("localhost", 0))
        threading.Thread(target=self._accept_loop, daemon=True).start()
        return self._listener.getsockname()[1]
```

Kode 16.30 (Lanjutan) *laundry_protocol.py: protokol, server, dan klien yang sama dalam Python.*

```
    def _accept_loop(self):
        try:
            while True:
                c, _ = self._listener.accept()
                threading.Thread(target=self._serve, args=(c,), daemon=True).start()
        except OSError:
            pass                                        # listener ditutup

    def _serve(self, c):
        try:
            with c, c.makefile("r", encoding="utf-8") as reader, \
                    c.makefile("w", encoding="utf-8") as writer:
                writer.write(GREETING + "\n")
                writer.flush()
                for line in reader:                     # berakhir ketika klien memutus
sambungan
                    answer = reply(line) if line.strip() else "ERR empty request"
                    writer.write(answer + "\n")
                    writer.flush()
                    if answer == "OK BYE":
                        return
        except OSError:
            pass                                        # klien menghilang

    def stop(self):
        try:
            self._listener.shutdown(socket.SHUT_RDWR)
        except OSError:
            pass
        self._listener.close()


class Client:
    """Klien: satu metode per permintaan, dengan batas waktu pada setiap balasan."""

    def __init__(self, port):
        self._socket = socket.create_connection(("localhost", port), timeout=0.5)
        self._reader = self._socket.makefile("r", encoding="utf-8")
        self._writer = self._socket.makefile("w", encoding="utf-8")
        hello = self._reader.readline().rstrip("\n")
        if hello != GREETING:
            raise OSError(f"not a LAUNDRY/1 server: {hello}")

    def _send(self, line):
        self._writer.write(line + "\n")
        self._writer.flush()                            # dorong byte ke jaringan SEKARANG
```

Kode 16.30 (Lanjutan) *laundry_protocol.py: protokol, server, dan klien yang sama dalam Python.*

```
    def request(self, line):
        self._send(line)
        try:
            return self._reader.readline().rstrip("\n")
        except TimeoutError:
            return "(no reply within 500 ms)"

    def close(self):
        self._socket.close()

    def __enter__(self):
        return self

    def __exit__(self, *exc):
        self.close()


class NoFlushClient(Client):
    """Menggoda tetapi keliru: menulis ke penyangga berkas dan tidak pernah
mengosongkannya."""

    def _send(self, line):
        self._writer.write(line + "\n")


SCRIPT = ["SERVICES", "PRICE wash 3.5", "price EXPRESS 2", "PRICE DRYCLEAN 1",
          "PRICE WASH abc", "PRICE IRON 80", "HELLO", "QUIT"]


def ten_clients_at_once(port):
    def one(kg):
        with Client(port) as c:
            return int(c.request(f"PRICE WASH {kg}")[3:])
    with ThreadPoolExecutor(max_workers=10) as pool:
        return sum(pool.map(one, range(1, 11)))


class TestLaundryProtocol(unittest.TestCase):
    @classmethod
    def setUpClass(cls):
        cls.server = Server()
        cls.port = cls.server.start()

    @classmethod
    def tearDownClass(cls):
        cls.server.stop()
```

Kode 16.30 (Lanjutan) *laundry_protocol.py: protokol, server, dan klien yang sama dalam Python.*

```
    def test_script(self):
        with Client(self.port) as c:
            replies = [c.request(r) for r in SCRIPT]
        self.assertEqual("OK 24500", replies[1])
        self.assertEqual("OK 28000", replies[2])
        self.assertEqual("ERR unknown service DRYCLEAN", replies[3])
        self.assertEqual("ERR weight is not a number: abc", replies[4])
        self.assertEqual("ERR unknown command HELLO", replies[6])

    def test_ten_clients_at_once(self):
        self.assertEqual(385_000, ten_clients_at_once(self.port))

    def test_missing_flush_means_no_reply(self):
        with NoFlushClient(self.port) as wrong:
            self.assertEqual("(no reply within 500 ms)", wrong.request("PRICE WASH 1"))


if __name__ == "__main__":
    server = Server()
    port = server.start()
    with Client(port) as c:
        for request in SCRIPT:
            print(f"{request:<18} -> {c.request(request)}")
    print("ten clients at once, total of their prices:", ten_clients_at_once(port))
    with NoFlushClient(port) as wrong:
        print("client without flush ->", wrong.request("PRICE WASH 1"))
    server.stop()
    sys.stdout.flush()
    unittest.main(verbosity=2)
```

```
SERVICES           -> OK EXPRESS=14000 IRON=5000 WASH=7000
PRICE wash 3.5     -> OK 24500
price EXPRESS 2    -> OK 28000
PRICE DRYCLEAN 1   -> ERR unknown service DRYCLEAN
PRICE WASH abc     -> ERR weight is not a number: abc
PRICE IRON 80      -> ERR weight must be in (0, 50] kg
HELLO              -> ERR unknown command HELLO
QUIT               -> OK BYE
ten clients at once, total of their prices: 385000
client without flush -> (no reply within 500 ms)
test_missing_flush_means_no_reply
(__main__.TestLaundryProtocol.test_missing_flush_means_no_reply) ... ok
test_script (__main__.TestLaundryProtocol.test_script) ... ok
test_ten_clients_at_once (__main__.TestLaundryProtocol.test_ten_clients_at_once) ... ok

----------------------------------------------------------------------
Ran 3 tests in 0.529s

OK
```

Baris demi baris. `reply` memecah permintaan pada deretan spasi (`"\\s+"` di Java; `split()` tanpa argumen di Python) lalu bercabang menurut kata pertama yang diubah ke huruf kapital, sehingga `price EXPRESS 2` tetap berjalan. Setiap cabang `ERR` bersesuaian dengan satu baris **Tabel 16.9**. Harganya adalah `Math.round(rate * kg)`: $7000 \times 3.5 = 24{,}500$ dan $14{,}000 \times 2 = 28{,}000$, seperti yang diperlihatkan keluarannya. Karena `reply` murni, sebagian besar protokol dapat diuji tanpa soket sama sekali; uji jaringan ada untuk memeriksa pembingkaian dan konkurensinya. Server menyapa, lalu menjawab setiap baris dengan tepat satu baris; baris kosong mendapat `ERR empty request` alih-alih eksepsi dari kode penguraian. `Client` membaca dan memeriksa sapaan di konstruktornya, sehingga klien yang tersambung ke jenis server yang salah langsung gagal dengan pesan yang jelas. `request` mengirim satu baris dan membaca satu baris dengan batas 500

ms. Sepuluh klien sekaligus menanyakan harga 1 sampai 10 kg cucian; jumlah balasannya adalah $7000 \times (1 + 2 + \cdots + 10) = 7000 \times 55 = 385{,}000$, yang dikonfirmasi pemeriksaannya.

Penyelesaian keliru yang menggoda. `NoFlushClient` berbeda dari `Client` hanya pada satu argumen: `PrintWriter`-nya dibuat tanpa *autoflush*. `println` lalu menulis permintaan ke penyangga internal *writer*, bukan ke jaringan. Server tidak pernah menerimanya, sehingga tidak pernah membalas; klien menunggu di `readLine()` dan, berkat batas waktu, melaporkan `(no reply within 500 ms)`. Tanpa batas waktu, klien itu akan menggantung selamanya, dan tidak ada apa pun dalam kodenya yang tampak salah. `NoFlushClient` pada Python melakukan kekeliruan yang sama dengan menghilangkan `self._writer.flush()`. Penyangga membuat proses menulis efisien, tetapi protokol permintaan–balasan harus melakukan *flush* setelah setiap permintaan.

> **UJI DIRI.** Tambahkan permintaan `QUOTE <service> <kg>` yang mengembalikan harga yang sama dengan `PRICE` tetapi ditambah 10% untuk pengantaran, dibulatkan ke rupiah utuh. Apa balasan `QUOTE WASH 3.5`, dan di mana perubahannya diletakkan?
>
> *Jawaban.* `OK 26950` ($24{,}500 \times 1.1 = 26{,}950$). Perubahannya hanya masuk ke `reply`: `case "QUOTE"` baru yang menghitung balasan `PRICE` dan, bila diawali `OK`, mengalikan bilangannya dengan 1,1 lalu membulatkannya. Server maupun klien tidak berubah, dan itulah manfaat menjaga protokol tetap berupa fungsi murni.

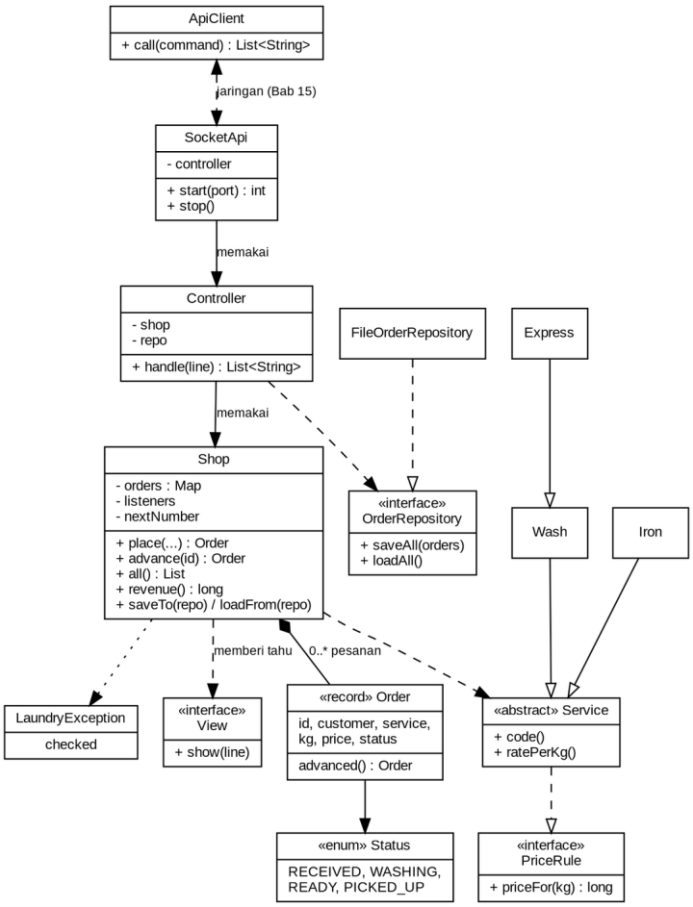
16.10 Proyek Puncak: Profitina Laundry dari Ujung ke Ujung

16.10.1 Soal 16.14 — Sistem binatu yang kecil tetapi lengkap

> **SOAL.** Bangunlah sistem Profitina Laundry kecil yang menggabungkan gagasan-gagasan utama seluruh buku. Pelanggan memesan sebuah layanan (cuci, setrika, atau ekspres) dengan suatu berat; setiap pesanan mendapat pengenal dan harga, lalu bergerak melalui status diterima, dicuci, siap, dan diambil. Toko melaporkan pesanan-pesanannya dan pendapatannya, menyimpan pesanan ke sebuah berkas, dan memuatnya kembali setelah dijalankan ulang. Beberapa loket bekerja bersamaan melalui sebuah API soket. Pakailah kelas, antarmuka, pewarisan, koleksi, eksepsi *checked*, berkas, dan pola Model–View–Controller, dalam Java dan dalam Python, beserta ujinya.

Dengan bahasa kita sendiri. Soal ini adalah buku ini dalam versi mini. Cara terbaik membacanya adalah sebagai daftar keputusan, yang masing-masing pernah diajarkan salah satu bab sebelumnya, dan mengenali setiap keputusan itu ketika muncul.

Rancangan. **Gambar 16.9** adalah diagram kelasnya. Sistem ini memiliki empat lapis, dari jaringan ke dalam: `SocketApi` menerima koneksi dan meneruskan setiap baris perintah ke `Controller`; pengendali memanggil `Shop`, yaitu modelnya, dan mengubah hasil serta penolakan menjadi baris balasan; toko menyimpan pesanan dan memberi tahu para pengamatnya, yaitu *view*-*view*, tentang setiap perubahan; pesanan, layanan, dan repositori adalah blok bangunan terkecil. **Tabel 16.10** mendaftarkan tempat setiap gagasan buku ini muncul.



Gambar 16.9 Diagram kelas UML proyek puncak. Segitiga berongga berarti "mewarisi" (garis utuh) atau "mengimplementasikan" (garis putus-putus); belah ketupat berarti toko menyimpan pesanan-pesanan.

Tabel 16.10 Tempat gagasan-gagasan seluruh buku muncul di proyek puncak.

Gagasan buku	Bab	Tempatnya muncul di proyek puncak
kelas, objek, *string*	3	setiap kelas; `describe` memformat sebuah pesanan
eksepsi, berkas	5	`LaundryException` yang *checked*; `FileOrderRepository`
antarmuka, dokumentasi, uji	6	`PriceRule`, `OrderRepository`, `View`; uji diri
pewarisan, polimorfisme, `super`	7	`Service`, `Wash`, `Express`; `ratePerKg()` di-*override*
koleksi, generik	9	`LinkedHashMap<String, Order>`, `List<Consumer<Order>>`
peristiwa, pengamat	10	*listener* yang dipanggil pada setiap perubahan
Model–View–Controller	11	`Shop`, `View`, `Controller`
imutabilitas, *enum*	13	*record* `Order`, `advanced()`, `Status.next()`, *list* potret
keamanan *thread*	14	metode toko yang `synchronized`, `CopyOnWriteArrayList` berisi *listener*
soket, pembingkaian protokol	15	`SocketApi`, `ApiClient`, penutup berupa titik tunggal

API soket memerlukan satu bagian baru dalam rancangan protokol. Perintah `LIST` mengembalikan satu baris per pesanan, sehingga balasan kini memiliki banyak baris yang tidak tetap, dan klien memerlukan aturan untuk mengetahui di mana sebuah balasan berakhir. Kita memakai aturan protokol surel SMTP [5]: setiap balasan diakhiri baris yang hanya berisi sebuah titik. Tidak ada baris balasan yang dapat berupa titik tunggal, karena setiap baris pesanan diawali sebuah pengenal dan setiap baris status diawali `OK` atau `ERR`.

Penyelesaian Java. Modelnya adalah **Kode 16.31**, `LaundryCore.java`, sebuah kelas wadah bagi tipe-tipe model. Kelas ini memiliki konstruktor privat, seperti `ListOps` di Bab 8, karena tak seorang pun perlu membuat objek `LaundryCore`.

Kode 16.31 `LaundryCore.java`: model proyek puncak: eksepsi *checked*, antarmuka `PriceRule`, hierarki layanan, *enum* status, *record* `Order` yang imutabel, repositori berkas, dan `Shop` yang aman-*thread*.

```
// LaundryCore.java - Soal 16.14 (bagian 1): model Profitina Laundry.
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.util.ArrayList;
import java.util.Collection;
import java.util.LinkedHashMap;
import java.util.List;
import java.util.Locale;
import java.util.Map;
import java.util.concurrent.CopyOnWriteArrayList;
import java.util.function.Consumer;

public final class LaundryCore {

    private LaundryCore() {
    }

    /** Eksepsi checked: setiap pemanggil harus memutuskan arti permintaan yang ditolak
baginya. */
    public static final class LaundryException extends Exception {
        private static final long serialVersionUID = 1L;

        public LaundryException(String message) {
            super(message);
        }
    }

    /** Apa saja yang dapat menghargai suatu berat (Bab 6: antarmuka). */
    public interface PriceRule {
        long priceFor(double kg);
    }

    /** Layanan-layanan (Bab 7: kelas abstrak dan hierarki kecil). */
    public abstract static class Service implements PriceRule {
        public abstract String code();

        public abstract long ratePerKg();

        @Override
        public long priceFor(double kg) {
            return Math.round(ratePerKg() * kg);
        }
    }
```

Kode 16.31 (Lanjutan) *LaundryCore.java: model proyek puncak: eksepsi checked, antarmuka PriceRule, hierarki layanan, enum status, record Order yang imutabel, repositori berkas, dan Shop yang aman-thread.*

```
    public static class Wash extends Service {
        public String code() {
            return "WASH";
        }

        public long ratePerKg() {
            return 7000;
        }
    }

    public static final class Iron extends Service {
        public String code() {
            return "IRON";
        }

        public long ratePerKg() {
            return 5000;
        }
    }

    public static final class Express extends Wash {        // cuci ekspres ADALAH cuci
        @Override
        public String code() {
            return "EXPRESS";
        }

        @Override
        public long ratePerKg() {
            return 2 * super.ratePerKg();                   // dua kali tarif cuci
        }
    }

    static final Map<String, Service> SERVICES = Map.of("WASH", new Wash(), "IRON", new
Iron(),
                                                        "EXPRESS", new Express());

    public static Service service(String code) throws LaundryException {
        Service s = SERVICES.get(code.toUpperCase(Locale.ROOT));
        if (s == null) {
            throw new LaundryException("unknown service " + code);
        }
        return s;
    }
```

Kode 16.31 (Lanjutan) *LaundryCore.java: model proyek puncak: eksepsi checked, antarmuka PriceRule, hierarki layanan, enum status, record Order yang imutabel, repositori berkas, dan Shop yang aman-thread.*

```
    /** Riwayat hidup pesanan, secara berurutan. */
    public enum Status {
        RECEIVED, WASHING, READY, PICKED_UP;

        Status next() throws LaundryException {
            if (this == PICKED_UP) {
                throw new LaundryException("already picked up");
            }
            return values()[ordinal() + 1];
        }
    }

    /** Satu pesanan (Bab 13: imutabel; perubahan membuat Order baru). */
    public record Order(String id, String customer, String service, double kg, long price,
                        Status status) {
        Order advanced() throws LaundryException {
            return new Order(id, customer, service, kg, price, status.next());
        }

        String toLine() {
            return String.join(";", id, customer, service, Double.toString(kg),
                               Long.toString(price), status.name());
        }

        static Order fromLine(String line) {
            String[] p = line.split(";");
            return new Order(p[0], p[1], p[2], Double.parseDouble(p[3]),
Long.parseLong(p[4]),
                             Status.valueOf(p[5]));
        }
    }

    /** Tempat pesanan disimpan di antara dua kali jalan (Bab 5: berkas, di balik
antarmuka). */
    public interface OrderRepository {
        void saveAll(Collection<Order> orders) throws IOException;

        List<Order> loadAll() throws IOException;
    }

    public static final class FileOrderRepository implements OrderRepository {
        private final Path file;

        public FileOrderRepository(Path file) {
            this.file = file;
        }
```

Kode 16.31 (Lanjutan) *LaundryCore.java: model proyek puncak: eksepsi checked, antarmuka PriceRule, hierarki layanan, enum status, record Order yang imutabel, repositori berkas, dan Shop yang aman-thread.*

```
        @Override
        public void saveAll(Collection<Order> orders) throws IOException {
            List<String> lines = new ArrayList<>();
            for (Order o : orders) {
                lines.add(o.toLine());
            }
            Files.write(file, lines);
        }

        @Override
        public List<Order> loadAll() throws IOException {
            List<Order> result = new ArrayList<>();
            if (Files.exists(file)) {
                for (String line : Files.readAllLines(file)) {
                    result.add(Order.fromLine(line));
                }
            }
            return result;
        }
    }

    /** Model (Bab 11). Setiap metode publik synchronized: banyak klien memakai satu toko
bersama. */
    public static final class Shop {
        private final Map<String, Order> orders = new LinkedHashMap<>();   // menjaga
urutan kedatangan
        private final List<Consumer<Order>> listeners = new CopyOnWriteArrayList<>();
        private int nextNumber = 1;

        /** Pengamat (huruf V dari MVC) diberi tahu setiap perubahan. */
        public void addListener(Consumer<Order> listener) {
            listeners.add(listener);
        }

        public synchronized Order place(String customer, String serviceCode, String kgText)
                throws LaundryException {
            if (customer.isBlank() || customer.contains(";")) {
                throw new LaundryException("customer name is missing or contains ';'");
            }
            Service s = service(serviceCode);
            double kg;
            try {
                kg = Double.parseDouble(kgText);
            } catch (NumberFormatException e) {
                throw new LaundryException("weight is not a number: " + kgText);
            }
            if (kg <= 0 || kg > 50) {
                throw new LaundryException("weight must be in (0, 50] kg");
```

Kode 16.31 (Lanjutan) *LaundryCore.java: model proyek puncak: eksepsi checked, antarmuka PriceRule, hierarki layanan, enum status, record Order yang imutabel, repositori berkas, dan Shop yang aman-thread.*

```
            }
            Order o = new Order("L" + nextNumber++, customer, s.code(), kg, s.priceFor(kg),
                                Status.RECEIVED);
            orders.put(o.id(), o);
            listeners.forEach(l -> l.accept(o));
            return o;
        }

        public synchronized Order advance(String id) throws LaundryException {
            Order old = orders.get(id);
            if (old == null) {
                throw new LaundryException("no order " + id);
            }
            Order changed = old.advanced();
            orders.put(id, changed);                        // ganti, jangan pernah ubah
            listeners.forEach(l -> l.accept(changed));
            return changed;
        }

        public synchronized List<Order> all() {
            return List.copyOf(orders.values());           // potret yang tidak dapat
diubah pemanggil
        }

        public synchronized long revenue() {
            long sum = 0;
            for (Order o : orders.values()) {
                sum += o.price();
            }
            return sum;
        }

        public synchronized void saveTo(OrderRepository repo) throws IOException {
            repo.saveAll(orders.values());
        }

        public synchronized void loadFrom(OrderRepository repo) throws IOException {
            orders.clear();
            for (Order o : repo.loadAll()) {
                orders.put(o.id(), o);
                nextNumber = Math.max(nextNumber, Integer.parseInt(o.id().substring(1)) +
1);
            }
        }
    }
}
```

Kode 16.32, `ProfitinaLaundry.java`, menambahkan pengendali, *view*, dan API soket. Metode `main`-nya memiliki tiga mode: `java ProfitinaLaundry 5000 orders.csv` menjalankan server pada port 5000 dengan *view* konsol; `java ProfitinaLaundry client 5000` adalah klien papan ketik; tanpa argumen, ia menjalankan uji diri.

Kode 16.32 `ProfitinaLaundry.java`: pengendali, *view*, API soket dengan balasan yang diakhiri titik, klien, dan uji diri.

```
// ProfitinaLaundry.java - Soal 16.14 (bagian 2): pengendali, view, dan API soket di
sekeliling LaundryCore.
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.net.ServerSocket;
import java.net.Socket;
import java.nio.charset.StandardCharsets;
import java.nio.file.Files;
import java.nio.file.Path;
import java.util.ArrayList;
import java.util.Collections;
import java.util.HashSet;
import java.util.List;
import java.util.Locale;
import java.util.Set;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.Future;

public class ProfitinaLaundry {

    static final String GREETING = "PROFITINA-LAUNDRY/1 READY";
    static final String END = ".";                  // baris berisi satu titik saja
mengakhiri setiap balasan

    /** Huruf V dari MVC: apa saja yang dapat menampilkan sebaris teks. */
    interface View {
        void show(String line);
    }

    /** Huruf C dari MVC: mengubah satu baris perintah menjadi panggilan ke model dan
baris-baris balasan. */
    static final class Controller {
        private final LaundryCore.Shop shop;
        private final LaundryCore.OrderRepository repo;

        Controller(LaundryCore.Shop shop, LaundryCore.OrderRepository repo) {
            this.shop = shop;
            this.repo = repo;
        }
```

Kode 16.32 (Lanjutan) *ProfitinaLaundry.java: pengendali, view, API soket dengan balasan yang diakhiri titik, klien, dan uji diri.*

```
        List<String> handle(String commandLine) {
            String[] w = commandLine.trim().split("\\s+");
            try {
                switch (w[0].toUpperCase(Locale.ROOT)) {
                    case "PLACE":
                        need(w, 4, "PLACE <customer> <service> <kg>");
                        LaundryCore.Order o = shop.place(w[1], w[2], w[3]);
                        return List.of("OK " + o.id() + " " + o.price());
                    case "ADVANCE":
                        need(w, 2, "ADVANCE <id>");
                        return List.of("OK " + w[1] + " " + shop.advance(w[1]).status());
                    case "LIST":
                        List<String> lines = new ArrayList<>();
                        for (LaundryCore.Order x : shop.all()) {
                            lines.add(describe(x));
                        }
                        lines.add("OK " + lines.size() + " orders");
                        return lines;
                    case "REVENUE":
                        return List.of("OK " + shop.revenue());
                    case "SAVE":
                        shop.saveTo(repo);
                        return List.of("OK saved " + shop.all().size() + " orders");
                    default:
                        return List.of("ERR unknown command " + w[0]);
                }
            } catch (LaundryCore.LaundryException e) {
                return List.of("ERR " + e.getMessage());   // permintaan yang ditolak,
bukan crash
            } catch (IOException e) {
                return List.of("ERR storage failed: " + e.getClass().getSimpleName());
            }
        }

        private static void need(String[] w, int n, String usage) throws
LaundryCore.LaundryException {
            if (w.length != n) {
                throw new LaundryCore.LaundryException("usage: " + usage);
            }
        }
    }

    static String describe(LaundryCore.Order o) {
        return String.format(Locale.ROOT, "%s %s %s %.1f kg IDR %,d %s", o.id(),
o.customer(),
                             o.service(), o.kg(), o.price(), o.status());
    }
```

Kode 16.32 (Lanjutan) *ProfitinaLaundry.java: pengendali, view, API soket dengan balasan yang diakhiri titik, klien, dan uji diri.*

```
    /** API soket (Bab 15): satu thread per klien, semuanya memakai satu pengendali
bersama. */
    static final class SocketApi {
        private final Controller controller;
        private final ExecutorService workers = Executors.newCachedThreadPool();
        private ServerSocket listener;

        SocketApi(Controller controller) {
            this.controller = controller;
        }

        int start(int port) throws IOException {
            listener = new ServerSocket(port);
            Thread acceptor = new Thread(() -> {
                try {
                    while (true) {
                        Socket c = listener.accept();
                        workers.execute(() -> serve(c));
                    }
                } catch (IOException e) {
                    // listener ditutup
                }
            });
            acceptor.setDaemon(true);
            acceptor.start();
            return listener.getLocalPort();
        }

        private void serve(Socket c) {
            try (c;
                 BufferedReader in = new BufferedReader(new InputStreamReader(
                         c.getInputStream(), StandardCharsets.UTF_8));
                 PrintWriter out = new PrintWriter(new OutputStreamWriter(
                         c.getOutputStream(), StandardCharsets.UTF_8), true)) {
                out.println(GREETING);
                String line;
                while ((line = in.readLine()) != null && !line.equalsIgnoreCase("QUIT")) {
                    for (String reply : controller.handle(line)) {
                        out.println(reply);
                    }
                    out.println(END);
                }
            } catch (IOException e) {
                // klien menghilang: thread-nya cukup berakhir
            }
        }
```

Kode 16.32 (Lanjutan) *ProfitinaLaundry.java: pengendali, view, API soket dengan balasan yang diakhiri titik, klien, dan uji diri.*

```
        void stop() throws IOException {
            listener.close();
            workers.shutdown();
        }
    }

    /** Klien API soket: mengirim satu perintah, membaca baris sampai titik tunggal. */
    static final class ApiClient implements AutoCloseable {
        private final Socket socket;
        private final BufferedReader in;
        private final PrintWriter out;

        ApiClient(int port) throws IOException {
            socket = new Socket("localhost", port);
            socket.setSoTimeout(2000);
            in = new BufferedReader(new InputStreamReader(socket.getInputStream(),
StandardCharsets.UTF_8));
            out = new PrintWriter(new OutputStreamWriter(socket.getOutputStream(),
StandardCharsets.UTF_8), true);
            if (!GREETING.equals(in.readLine())) {
                throw new IOException("not a Profitina Laundry server");
            }
        }

        List<String> call(String command) throws IOException {
            out.println(command);
            List<String> reply = new ArrayList<>();
            String line;
            while (!(line = in.readLine()).equals(END)) {
                reply.add(line);
            }
            return reply;
        }

        /** Menggoda tetapi keliru: menganggap setiap balasan tepat satu baris. */
        String callOneLine(String command) throws IOException {
            out.println(command);
            return in.readLine();
        }

        @Override
        public void close() throws IOException {
            socket.close();
        }
    }

    private static int passed = 0;
```

Kode 16.32 (Lanjutan) *ProfitinaLaundry.java: pengendali, view, API soket dengan balasan yang diakhiri titik, klien, dan uji diri.*

```
    private static void check(String name, Object expected, Object actual) {
        if (!java.util.Objects.equals(expected, actual)) {
            throw new AssertionError(name + ": expected " + expected + " but got " +
actual);
        }
        passed++;
        System.out.println("PASS " + name);
    }

    public static void main(String[] args) throws Exception {
        if (args.length == 2 && args[0].equals("client")) {   // java ProfitinaLaundry
client 5000
            BufferedReader keyboard = new BufferedReader(new InputStreamReader(System.in));
            try (ApiClient client = new ApiClient(Integer.parseInt(args[1]))) {
                String command;
                while ((command = keyboard.readLine()) != null &&
!command.equalsIgnoreCase("QUIT")) {
                    client.call(command).forEach(System.out::println);
                }
            }
            return;
        }
        Path file = (args.length > 1) ? Path.of(args[1]) : Files.createTempFile("laundry",
".csv");
        LaundryCore.Shop shop = new LaundryCore.Shop();
        Controller controller = new Controller(shop, new
LaundryCore.FileOrderRepository(file));
        if (args.length > 0 && args[0].matches("\\d+")) {  // java ProfitinaLaundry 5000
orders.csv
            View console = System.out::println;
            shop.loadFrom(new LaundryCore.FileOrderRepository(file));
            shop.addListener(o -> console.show("[shop] " + describe(o)));
            new SocketApi(controller).start(Integer.parseInt(args[0]));
            console.show("Profitina Laundry listening on port " + args[0] + "; orders in "
+ file);
            Thread.currentThread().join();
        }

        List<String> viewLog = Collections.synchronizedList(new ArrayList<>());
        View recording = viewLog::add;
        shop.addListener(o -> recording.show(describe(o)));

        String[] session = {"PLACE Sari WASH 3.5", "PLACE Budi EXPRESS 2", "PLACE Dewi
DRYCLEAN 1",
                            "PLACE Rudi IRON -2", "ADVANCE L1", "ADVANCE L1", "ADVANCE L9",
                            "LIST", "REVENUE", "FOLD L1"};
        for (String command : session) {
            System.out.println("> " + command);
            controller.handle(command).forEach(r -> System.out.println("  " + r));
        }
        check("view saw every change", 4, viewLog.size());
```

Kode 16.32 (Lanjutan) *ProfitinaLaundry.java: pengendali, view, API soket dengan balasan yang diakhiri titik, klien, dan uji diri.*

```
        SocketApi api = new SocketApi(controller);
        int port = api.start(0);
        ExecutorService crowd = Executors.newFixedThreadPool(5);
        List<Future<List<String>>> futures = new ArrayList<>();
        for (int c = 0; c < 5; c++) {
            String name = "Client" + c;
            futures.add(crowd.submit(() -> {
                List<String> ids = new ArrayList<>();
                try (ApiClient client = new ApiClient(port)) {
                    for (int k = 1; k <= 4; k++) {
                        ids.add(client.call("PLACE " + name + " WASH " + k).get(0).split("
")[1]);
                    }
                }
                return ids;
            }));
        }
        Set<String> ids = new HashSet<>();
        for (Future<List<String>> f : futures) {
            ids.addAll(f.get());
        }
        crowd.shutdown();
        System.out.println("5 clients x 4 orders over the network: " + ids.size() + "
different ids");

        try (ApiClient client = new ApiClient(port)) {
            System.out.println("SAVE    -> " + client.call("SAVE"));
            System.out.println("REVENUE -> " + client.call("REVENUE"));
        }
        String outOfStep;
        try (ApiClient sloppy = new ApiClient(port)) {
            sloppy.callOneLine("LIST");                    // hanya membaca baris pertama
dari 24 baris
            outOfStep = sloppy.callOneLine("REVENUE");     // ... jadi ini baris kedua
dari LIST
            System.out.println("one-line client asks REVENUE, gets: " + outOfStep);
        }
        api.stop();

        LaundryCore.Shop reopened = new LaundryCore.Shop();
        reopened.loadFrom(new LaundryCore.FileOrderRepository(file));
        LaundryCore.Order next = reopened.place("Nia", "IRON", "1");
        System.out.println("after a restart: " + reopened.all().size() + " orders, next id
" + next.id());
```

Kode 16.32 (Lanjutan) *ProfitinaLaundry.java: pengendali, view, API soket dengan balasan yang diakhiri titik, klien, dan uji diri.*

```
        check("20 different ids", 20, ids.size());
        check("revenue", shop.revenue(), reopened.revenue() - next.price());
        check("numbering continues", "L23", next.id());
        check("the one-line client is out of step", describe(shop.all().get(1)),
outOfStep);
        Files.deleteIfExists(file);
        System.out.println("All " + passed + " checks passed.");
    }
}
```

```
> PLACE Sari WASH 3.5
  OK L1 24500
> PLACE Budi EXPRESS 2
  OK L2 28000
> PLACE Dewi DRYCLEAN 1
  ERR unknown service DRYCLEAN
> PLACE Rudi IRON -2
  ERR weight must be in (0, 50] kg
> ADVANCE L1
  OK L1 WASHING
> ADVANCE L1
  OK L1 READY
> ADVANCE L9
  ERR no order L9
> LIST
  L1 Sari WASH 3.5 kg IDR 24,500 READY
  L2 Budi EXPRESS 2.0 kg IDR 28,000 RECEIVED
  OK 2 orders
> REVENUE
  OK 52500
> FOLD L1
  ERR unknown command FOLD
PASS view saw every change
5 clients x 4 orders over the network: 20 different ids
SAVE    -> [OK saved 22 orders]
REVENUE -> [OK 402500]
one-line client asks REVENUE, gets: L2 Budi EXPRESS 2.0 kg IDR 28,000 RECEIVED
after a restart: 23 orders, next id L23
PASS 20 different ids
PASS revenue
PASS numbering continues
PASS the one-line client is out of step
All 5 checks passed.
```

Penyelesaian Python. **Kode 16.33** dan **Kode 16.34** adalah kedua lapis yang sama dalam Python, dengan tiga mode yang sama di baris perintah.

Kode 16.33 `laundry_core.py`: model dalam Python, dengan `RLock` yang melindungi toko.

```python
# laundry_core.py - Soal 16.14 (bagian 1): model Profitina Laundry.
import os
import threading
from abc import ABC, abstractmethod
from dataclasses import dataclass, replace
from enum import Enum


class LaundryError(Exception):
    """Permintaan yang ditolak: setiap pemanggil harus memutuskan artinya baginya."""


class PriceRule(ABC):
    """Apa saja yang dapat menghargai suatu berat (Bab 6: antarmuka)."""

    @abstractmethod
    def price_for(self, kg):
        ...


class Service(PriceRule):
    """Layanan-layanan (Bab 7: kelas abstrak dan hierarki kecil)."""
    code = "?"

    @abstractmethod
    def rate_per_kg(self):
        ...

    def price_for(self, kg):
        return round(self.rate_per_kg() * kg)


class Wash(Service):
    code = "WASH"

    def rate_per_kg(self):
        return 7000


class Iron(Service):
    code = "IRON"

    def rate_per_kg(self):
        return 5000
```

Kode 16.33 (Lanjutan) *laundry_core.py: model dalam Python, dengan RLock yang melindungi toko.*

```
class Express(Wash):                            # cuci ekspres ADALAH cuci
    code = "EXPRESS"

    def rate_per_kg(self):
        return 2 * super().rate_per_kg()        # dua kali tarif cuci


SERVICES = {s.code: s for s in (Wash(), Iron(), Express())}


def service(code):
    s = SERVICES.get(code.upper())
    if s is None:
        raise LaundryError(f"unknown service {code}")
    return s


class Status(Enum):
    """Riwayat hidup pesanan, secara berurutan."""
    RECEIVED = 1
    WASHING = 2
    READY = 3
    PICKED_UP = 4

    def next(self):
        if self is Status.PICKED_UP:
            raise LaundryError("already picked up")
        return Status(self.value + 1)


@dataclass(frozen=True)
class Order:
    """Satu pesanan (Bab 13: imutabel; perubahan membuat Order baru)."""
    id: str
    customer: str
    service: str
    kg: float
    price: int
    status: Status

    def advanced(self):
        return replace(self, status=self.status.next())
```

Kode 16.33 (Lanjutan) *laundry_core.py: model dalam Python, dengan RLock yang melindungi toko.*

```
    def to_line(self):
        return ";".join((self.id, self.customer, self.service, str(self.kg),
                         str(self.price), self.status.name))

    @staticmethod
    def from_line(line):
        p = line.rstrip("\n").split(";")
        return Order(p[0], p[1], p[2], float(p[3]), int(p[4]), Status[p[5]])


class OrderRepository(ABC):
    """Tempat pesanan disimpan di antara dua kali jalan (Bab 5: berkas, di balik
antarmuka)."""

    @abstractmethod
    def save_all(self, orders):
        ...

    @abstractmethod
    def load_all(self):
        ...


class FileOrderRepository(OrderRepository):
    def __init__(self, file_name):
        self._file_name = file_name

    def save_all(self, orders):
        with open(self._file_name, "w", encoding="utf-8") as f:
            for o in orders:
                f.write(o.to_line() + "\n")

    def load_all(self):
        if not os.path.exists(self._file_name):
            return []
        with open(self._file_name, encoding="utf-8") as f:
            return [Order.from_line(line) for line in f if line.strip()]


class Shop:
    """Model (Bab 11). Setiap metode publik memegang kunci: banyak klien memakai satu toko
bersama."""
```

Kode 16.33 (Lanjutan) *laundry_core.py: model dalam Python, dengan RLock yang melindungi toko.*

```
    def __init__(self):
        self._orders = {}                       # dict menjaga urutan kedatangan
        self._listeners = []
        self._next_number = 1
        self._lock = threading.RLock()

    def add_listener(self, listener):
        """Pengamat (huruf V dari MVC) diberi tahu setiap perubahan."""
        self._listeners.append(listener)

    def _tell(self, order):
        for listener in list(self._listeners):
            listener(order)

    def place(self, customer, service_code, kg_text):
        with self._lock:
            if not customer.strip() or ";" in customer:
                raise LaundryError("customer name is missing or contains ';'")
            s = service(service_code)
            try:
                kg = float(kg_text)
            except ValueError:
                raise LaundryError(f"weight is not a number: {kg_text}") from None
            if not 0 < kg <= 50:
                raise LaundryError("weight must be in (0, 50] kg")
            o = Order(f"L{self._next_number}", customer, s.code, kg, s.price_for(kg),
Status.RECEIVED)
            self._next_number += 1
            self._orders[o.id] = o
            self._tell(o)
            return o

    def advance(self, order_id):
        with self._lock:
            old = self._orders.get(order_id)
            if old is None:
                raise LaundryError(f"no order {order_id}")
            changed = old.advanced()
            self._orders[order_id] = changed    # ganti, jangan pernah ubah
            self._tell(changed)
            return changed

    def all(self):
        with self._lock:
            return tuple(self._orders.values())     # potret yang tidak dapat diubah
pemanggil
```

Kode 16.33 (Lanjutan) *laundry_core.py: model dalam Python, dengan RLock yang melindungi toko.*

```
    def revenue(self):
        with self._lock:
            return sum(o.price for o in self._orders.values())

    def save_to(self, repo):
        with self._lock:
            repo.save_all(self._orders.values())

    def load_from(self, repo):
        with self._lock:
            self._orders.clear()
            for o in repo.load_all():
                self._orders[o.id] = o
                self._next_number = max(self._next_number, int(o.id[1:]) + 1)
```

Kode 16.34 `profitina_laundry.py`: pengendali, API soket, klien, dan uji dalam Python.

```
# profitina_laundry.py - Soal 16.14 (bagian 2): pengendali, view, dan API soket di
sekeliling laundry_core.
import os
import socket
import sys
import tempfile
import threading
import unittest
from concurrent.futures import ThreadPoolExecutor
from laundry_core import FileOrderRepository, LaundryError, Shop

GREETING = "PROFITINA-LAUNDRY/1 READY"
END = "."                                       # baris berisi satu titik saja mengakhiri
setiap balasan


def describe(o):
    return f"{o.id} {o.customer} {o.service} {o.kg:.1f} kg IDR {o.price:,} {o.status.name}"


class Controller:
    """Huruf C dari MVC: mengubah satu baris perintah menjadi panggilan ke model dan baris-
baris balasan."""

    def __init__(self, shop, repo):
        self._shop = shop
        self._repo = repo

    def handle(self, command_line):
        w = command_line.split()
        try:
            command = w[0].upper() if w else ""
            if command == "PLACE":
                self._need(w, 4, "PLACE <customer> <service> <kg>")
                o = self._shop.place(w[1], w[2], w[3])
                return [f"OK {o.id} {o.price}"]
            if command == "ADVANCE":
                self._need(w, 2, "ADVANCE <id>")
                return [f"OK {w[1]} {self._shop.advance(w[1]).status.name}"]
            if command == "LIST":
                lines = [describe(o) for o in self._shop.all()]
                return lines + [f"OK {len(lines)} orders"]
            if command == "REVENUE":
                return [f"OK {self._shop.revenue()}"]
            if command == "SAVE":
                self._shop.save_to(self._repo)
                return [f"OK saved {len(self._shop.all())} orders"]
            return [f"ERR unknown command {w[0] if w else ''}"]
        except LaundryError as e:
```

Kode 16.34 (Lanjutan) *profitina_laundry.py: pengendali, API soket, klien, dan uji dalam Python.*

```
            return [f"ERR {e}"]                 # permintaan yang ditolak, bukan crash
        except OSError as e:
            return [f"ERR storage failed: {type(e).__name__}"]

    @staticmethod
    def _need(w, n, usage):
        if len(w) != n:
            raise LaundryError(f"usage: {usage}")


class SocketApi:
    """API soket (Bab 15): satu thread per klien, semuanya memakai satu pengendali
bersama."""

    def __init__(self, controller):
        self._controller = controller

    def start(self, port):
        self._listener = socket.create_server(("localhost", port))
        threading.Thread(target=self._accept_loop, daemon=True).start()
        return self._listener.getsockname()[1]

    def _accept_loop(self):
        try:
            while True:
                c, _ = self._listener.accept()
                threading.Thread(target=self._serve, args=(c,), daemon=True).start()
        except OSError:
            pass                                # listener ditutup

    def _serve(self, c):
        try:
            with c, c.makefile("r", encoding="utf-8") as reader, \
                    c.makefile("w", encoding="utf-8") as writer:
                writer.write(GREETING + "\n")
                writer.flush()
                for line in reader:
                    if line.strip().upper() == "QUIT":
                        return
                    for reply in self._controller.handle(line):
                        writer.write(reply + "\n")
                    writer.write(END + "\n")
                    writer.flush()
        except OSError:
            pass                                # klien menghilang: thread-nya cukup
berakhir
```

Kode 16.34 (Lanjutan) *profitina_laundry.py: pengendali, API soket, klien, dan uji dalam Python.*

```
    def stop(self):
        try:
            self._listener.shutdown(socket.SHUT_RDWR)
        except OSError:
            pass
        self._listener.close()


class ApiClient:
    """Klien API soket: mengirim satu perintah, membaca baris sampai titik tunggal."""

    def __init__(self, port):
        self._socket = socket.create_connection(("localhost", port), timeout=2)
        self._reader = self._socket.makefile("r", encoding="utf-8")
        self._writer = self._socket.makefile("w", encoding="utf-8")
        if self._reader.readline().rstrip("\n") != GREETING:
            raise OSError("not a Profitina Laundry server")

    def _send(self, command):
        self._writer.write(command + "\n")
        self._writer.flush()

    def call(self, command):
        self._send(command)
        reply = []
        while (line := self._reader.readline().rstrip("\n")) != END:
            reply.append(line)
        return reply

    def call_one_line(self, command):
        """Menggoda tetapi keliru: menganggap setiap balasan tepat satu baris."""
        self._send(command)
        return self._reader.readline().rstrip("\n")

    def close(self):
        self._socket.close()

    def __enter__(self):
        return self

    def __exit__(self, *exc):
        self.close()
```

Kode 16.34 (Lanjutan) *profitina_laundry.py: pengendali, API soket, klien, dan uji dalam Python.*

```
SESSION = ["PLACE Sari WASH 3.5", "PLACE Budi EXPRESS 2", "PLACE Dewi DRYCLEAN 1",
           "PLACE Rudi IRON -2", "ADVANCE L1", "ADVANCE L1", "ADVANCE L9",
           "LIST", "REVENUE", "FOLD L1"]


def crowd(port):
    """5 klien x 4 pesanan melalui jaringan, pada saat yang sama; mengembalikan semua
id."""
    def one(name):
        with ApiClient(port) as client:
            return [client.call(f"PLACE {name} WASH {k}")[0].split()[1] for k in range(1,
5)]
    with ThreadPoolExecutor(max_workers=5) as pool:
        return {i for ids in pool.map(one, [f"Client{c}" for c in range(5)]) for i in ids}


class TestProfitinaLaundry(unittest.TestCase):
    def setUp(self):
        fd, self.file = tempfile.mkstemp(suffix=".csv")
        os.close(fd)
        self.shop = Shop()
        self.controller = Controller(self.shop, FileOrderRepository(self.file))

    def tearDown(self):
        os.remove(self.file)

    def test_session(self):
        replies = [self.controller.handle(c) for c in SESSION]
        self.assertEqual(["OK L1 24500"], replies[0])
        self.assertEqual(["ERR unknown service DRYCLEAN"], replies[2])
        self.assertEqual(["OK L1 READY"], replies[5])
        self.assertEqual(["OK 52500"], replies[8])

    def test_network_save_and_restart(self):
        api = SocketApi(self.controller)
        port = api.start(0)
        self.assertEqual(20, len(crowd(port)))
        with ApiClient(port) as client:
            self.assertEqual(["OK saved 20 orders"], client.call("SAVE"))
        api.stop()
        reopened = Shop()
        reopened.load_from(FileOrderRepository(self.file))
        self.assertEqual(self.shop.revenue(), reopened.revenue())
        self.assertEqual("L21", reopened.place("Nia", "IRON", "1").id)
```

Kode 16.34 (Lanjutan) *profitina_laundry.py: pengendali, API soket, klien, dan uji dalam Python.*

```
    def test_one_line_client_is_out_of_step(self):
        self.controller.handle("PLACE Sari WASH 3.5")
        self.controller.handle("PLACE Budi EXPRESS 2")
        api = SocketApi(self.controller)
        with ApiClient(api.start(0)) as sloppy:
            sloppy.call_one_line("LIST")
            self.assertEqual(describe(self.shop.all()[1]), sloppy.call_one_line("REVENUE"))
        api.stop()


def serve(port, file_name):                    # python profitina_laundry.py 5000
orders.csv
    shop = Shop()
    shop.load_from(FileOrderRepository(file_name))
    shop.add_listener(lambda o: print("[shop]", describe(o), flush=True))
    SocketApi(Controller(shop, FileOrderRepository(file_name))).start(port)
    print(f"Profitina Laundry listening on port {port}; orders in {file_name}", flush=True)
    threading.Event().wait()


def client(port):                              # python profitina_laundry.py client 5000
    with ApiClient(port) as c:
        for command in sys.stdin:
            if command.strip().upper() == "QUIT":
                break
            print("\n".join(c.call(command.strip())))


def demo(folder):
    """Jalannya sama dengan metode main Java: satu sesi, jaringan, dan mulai ulang."""
    file_name = os.path.join(folder, "orders.csv")
    shop, view_log = Shop(), []
    shop.add_listener(lambda o: view_log.append(describe(o)))
    controller = Controller(shop, FileOrderRepository(file_name))
    for command in SESSION:
        print(">", command)
        for reply in controller.handle(command):
            print(" ", reply)
    print("view saw", len(view_log), "changes")
    api = SocketApi(controller)
    port = api.start(0)
    print("5 clients x 4 orders over the network:", len(crowd(port)), "different ids")
    with ApiClient(port) as c:
        print("SAVE    ->", c.call("SAVE"))
        print("REVENUE ->", c.call("REVENUE"))
    with ApiClient(port) as sloppy:
        sloppy.call_one_line("LIST")                    # hanya membaca baris pertama dari
24 baris
```

Kode 16.34 (Lanjutan) *profitina_laundry.py: pengendali, API soket, klien, dan uji dalam Python.*

```
        print("one-line client asks REVENUE, gets:", sloppy.call_one_line("REVENUE"))
    api.stop()
    reopened = Shop()
    reopened.load_from(FileOrderRepository(file_name))
    nxt = reopened.place("Nia", "IRON", "1")
    print(f"after a restart: {len(reopened.all())} orders, next id {nxt.id}")
    os.remove(file_name)


if __name__ == "__main__":
    if len(sys.argv) == 3 and sys.argv[1] == "client":
        client(int(sys.argv[2]))
    elif len(sys.argv) >= 2 and sys.argv[1].isdigit():
        serve(int(sys.argv[1]), sys.argv[2] if len(sys.argv) > 2 else "orders.csv")
    else:
        demo(tempfile.mkdtemp())
        sys.stdout.flush()
        unittest.main(verbosity=2)
```

```
> PLACE Sari WASH 3.5
  OK L1 24500
> PLACE Budi EXPRESS 2
  OK L2 28000
> PLACE Dewi DRYCLEAN 1
  ERR unknown service DRYCLEAN
> PLACE Rudi IRON -2
  ERR weight must be in (0, 50] kg
> ADVANCE L1
  OK L1 WASHING
> ADVANCE L1
  OK L1 READY
> ADVANCE L9
  ERR no order L9
> LIST
  L1 Sari WASH 3.5 kg IDR 24,500 READY
  L2 Budi EXPRESS 2.0 kg IDR 28,000 RECEIVED
  OK 2 orders
> REVENUE
  OK 52500
> FOLD L1
  ERR unknown command FOLD
view saw 4 changes
5 clients x 4 orders over the network: 20 different ids
SAVE    -> ['OK saved 22 orders']
REVENUE -> ['OK 402500']
one-line client asks REVENUE, gets: L2 Budi EXPRESS 2.0 kg IDR 28,000 RECEIVED
after a restart: 23 orders, next id L23
test_network_save_and_restart (__main__.TestProfitinaLaundry.test_network_save_and_restart)
... ok
test_one_line_client_is_out_of_step
(__main__.TestProfitinaLaundry.test_one_line_client_is_out_of_step) ... ok
test_session (__main__.TestProfitinaLaundry.test_session) ... ok

----------------------------------------------------------------------
Ran 3 tests in 0.018s

OK
```

Bila dijalankan sebagai server dan dipakai dari terminal kedua, program Java berperilaku sebagai berikut. Terminal pertama adalah *view* konsol server, tempat *listener* toko mencetak setiap perubahan:

```
Profitina Laundry listening on port 5000; orders in orders.csv
[shop] L1 Sari WASH 3.5 kg IDR 24,500 RECEIVED
[shop] L2 Budi IRON 2.0 kg IDR 10,000 RECEIVED
[shop] L1 Sari WASH 3.5 kg IDR 24,500 WASHING
```

dan terminal kedua adalah sebuah loket yang mengetik perintah (baris yang diketik pengguna adalah baris yang diawali kata perintah):

```
PLACE Sari WASH 3.5
OK L1 24500
PLACE Budi IRON 2
OK L2 10000
ADVANCE L1
OK L1 WASHING
LIST
L1 Sari WASH 3.5 kg IDR 24,500 WASHING
L2 Budi IRON 2.0 kg IDR 10,000 RECEIVED
OK 2 orders
SAVE
OK saved 2 orders
QUIT
```

Setelah `SAVE`, berkas `orders.csv` berisi satu baris per pesanan, `L1;Sari;WASH;3.5;24500;WASHING` dan `L2;Budi;IRON;2.0;10000;RECEIVED`, yang dibaca kembali oleh server ketika dijalankan lagi.

Baris demi baris. Kita menelusuri keputusan-keputusannya lapis demi lapis, dari dalam ke luar.

Layanan. `PriceRule` adalah antarmuka dengan satu metode, `priceFor(kg)`. Kelas abstrak `Service` mengimplementasikannya satu kali, berdasarkan dua metode abstrak, `code()` dan `ratePerKg()`; inilah pola *template method* dari Bab 8. `Wash` dan `Iron` menyediakan angkanya. `Express extends Wash`, karena cuci ekspres *adalah* cuci, dan `ratePerKg()`-nya mengembalikan `2 * super.ratePerKg()`: bila tarif cuci berubah, tarif ekspres otomatis mengikuti. `SERVICES` memetakan kode ke ketiga objek, dan `service(code)` mengubah kode yang tidak dikenal menjadi `LaundryException` yang *checked*.

Pesanan. `Status` adalah *enum* yang `next()`-nya mengembalikan konstanta berikutnya (`values()[ordinal() + 1]`) dan menolak melewati `PICKED_UP`. `Order` adalah *record*, sehingga imutabel; `advanced()` mengembalikan pesanan *baru* dengan status berikutnya. `toLine` dan `fromLine` mengubah pesanan menjadi sebaris berkas dan sebaliknya; nama pelanggan tidak boleh mengandung `;`, yang diperiksa `place`, karena berkasnya memakai `;` sebagai pemisah.

Repositori. `OrderRepository` adalah kontrak penyimpanan, seperti `JokeStore` pada Soal 16.1: toko bergantung pada antarmukanya, dan `FileOrderRepository` mengimplementasikannya dengan `Files.write` dan `Files.readAllLines`. Implementasi basis data dapat menggantikannya tanpa perubahan apa pun pada toko.

Toko. Pesanan-pesanan tinggal di sebuah `LinkedHashMap`, yaitu *hash map* yang juga mengingat urutan penyisipan, sehingga `LIST` menampilkan pesanan menurut urutan kedatangannya. Setiap metode publik yang menyentuh pesanan bersifat `synchronized`, karena API soket memanggil satu toko itu dari banyak *thread*; pemeriksaan pada keluaran, “20 different ids” dari 5 klien yang masing-masing menaruh 4 pesanan pada saat yang sama, adalah uji bagi keputusan itu. Tanpa `synchronized`, dua *thread* dapat membaca `nextNumber` yang sama dan membuat dua pesanan yang sama-sama bernama, misalnya, L7: pembaruan yang hilang dari Soal 16.11 di tempat yang baru. `advance` *mengganti* pesanan di *map* dengan objek baru, sehingga potret yang sebelumnya dikembalikan `all()` (`List.copyOf`, tak dapat diubah) tidak pernah berubah di bawah kaki pemanggilnya. *Listener*-nya disimpan dalam `CopyOnWriteArrayList`, yaitu *list* yang aman

ditelusuri selagi *thread* lain menambah isinya. `loadFrom` memulihkan pesanan-pesanan dan menetapkan `nextNumber` menjadi satu lebih besar daripada nomor terbesar yang ditemukan, dan itulah sebabnya pesanan yang ditaruh setelah dijalankan ulang bernomor L23.

Pengendali. `handle` memecah perintah, memeriksa banyaknya kata dengan `need`, memanggil model, dan mengubah setiap `LaundryException` menjadi baris yang diawali `ERR`. Pengubahan itu adalah baris terpenting pengendali: permintaan yang ditolak menghasilkan balasan yang sopan, tidak pernah *thread* koneksi yang mati. Sesi pada keluaran mencoba setiap perintah dan setiap jenis penolakan: layanan yang tidak dikenal, berat negatif, pesanan yang tidak ada, dan perintah yang tidak dikenal. Pendapatan setelah sesi itu adalah $24{,}500 + 28{,}000 = 52{,}500$.

View. `View` memiliki satu metode, `show(String)`, sehingga sebuah *lambda* atau rujukan metode sudah merupakan *view*: `System.out::println` untuk konsol server, `viewLog::add` untuk uji. Toko tidak tahu *view* apa saja yang ada; ia hanya memanggil *listener*-nya. Itulah hubungan pengamat yang menjadi jantung MVC (Bab 11).

API soket. `SocketApi.serve` adalah gelung server Soal 16.3 dengan tambahan aturan pembingkaian: setelah baris-baris balasan setiap perintah, ia menulis sebuah `.` tunggal. `ApiClient.call` membaca baris sampai titik itu. Pada keluaran, lima klien menaruh dua puluh pesanan secara serentak; `SAVE` dan `REVENUE` lalu melaporkan 22 pesanan dan pendapatan $52{,}500 + 5 \times 7000 \times (1 + 2 + 3 + 4) = 402{,}500$. Sebuah `Shop` baru yang dimuat dari berkas menaruh satu pesanan lagi dan menerima pengenal L23, dan pendapatannya dikurangi pesanan baru itu sama dengan pendapatan toko lama: perjalanan pulang-pergi lewat berkas tidak kehilangan apa pun.

Versi Python mengambil keputusan yang sama dengan perkakas Python: kelas dasar abstrak untuk antarmuka, atribut kelas `code` di setiap layanan, *dataclass* beku dengan `dataclasses.replace` untuk `advanced()`, *dictionary* (yang menjaga urutan penyisipan) untuk pesanan-pesanan, dan sebuah `RLock`, kunci yang boleh diambil ulang oleh *thread* yang sama, sehingga sebuah *listener* dapat dengan aman memanggil kembali toko itu.

Penyelesaian keliru yang menggoda. `callOneLine` adalah klien yang berjalan sempurna untuk protokol satu baris pada Soal 16.13 lalu dipakai ulang tanpa perubahan. Setelah `LIST`, ia hanya membaca baris pertama dari dua puluh empat baris balasan. Permintaan berikutnya, `REVENUE`, kemudian membaca baris *kedua* dari balasan `LIST` yang lama, yaitu pesanan L2, alih-alih pendapatan, dan setiap balasan sesudahnya juga bergeser. Tidak ada yang *crash*; klien itu hanya memercayai jawaban yang salah. Keluarannya memperlihatkan hal itu: `one-line client asks REVENUE, gets: L2 Budi EXPRESS 2.0 kg IDR 28,000 RECEIVED`. Bila balasan dapat terdiri atas beberapa baris, protokolnya memerlukan penanda akhir yang eksplisit, dan klien harus membaca sampai penanda itu.

> **UJI DIRI.** Sebuah layanan baru, cuci kering, bertarif 20.000 per kilogram. Kelas mana yang berubah, dan apa balasan `PLACE Ayu DRYCLEAN 1.5` pada toko yang baru?
>
> *Jawaban.* Satu kelas baru, `DryClean extends Service` dengan `code()` yang mengembalikan `"DRYCLEAN"` dan `ratePerKg()` yang mengembalikan 20000, serta satu entri baru di `SERVICES` (Python: tambahkan `DryClean()` ke *tuple*-nya). Tidak ada yang lain berubah: pengendali, toko, format berkas, dan API soket bekerja dengan `Service` apa pun. Balasannya `OK L1 30000`, karena $20{,}000 \times 1.5 = 30{,}000$.

16.11 Perbandingan Soal-Soal dalam Bab Ini

Tabel 16.11 menjajarkan keempat belas soal. Seperti di Bab 8, bacalah kolom terakhir dari atas ke bawah: hampir setiap penyelesaian keliru lolos dari uji yang tampak jelas dengan satu klien, satu *thread*, atau satu baris, dan baru gagal dalam kondisi yang menjadi pokok soalnya.

Tabel 16.11 Keempat belas soal dalam bab ini: apa yang dilatih masing-masing dan satu-satunya kasus yang disalahkan penyelesaian keliru yang menggoda.

Soal	Topik	Gagasan utama	Yang disalahkan penyelesaian keliru yang menggoda
16.1	kontrak	antarmuka ditambah uji kontrak	penomoran mulai 1: gagal pada `get(0)` dan `get(size())`
16.2	protokol	mesin keadaan, satu baris per giliran	tanpa keadaan: menjawab setiap "… who?"
16.3	server	satu *thread* per klien	satu per satu: klien kedua ditelantarkan
16.4	pengujian	setiap pihak menguji pihak lain dengan *stub*	uji lemah meloloskan klien mutan
16.5	konkurensi	gerbang mulai, orakel, tenggat	satu objek protokol bersama
16.6	kegagalan	batas waktu; akhir *stream* itu wajar	`null` atau `""` tidak diperiksa; *crash* atau gelung sibuk
16.7	GUI	jaringan di luar *thread* UI; MVC	jaringan di penangan tombol: jendela membeku
16.8	dokumentasi	kriteria terukur yang dapat diulang	satu pengukuran mujur ditulis sebagai fakta
16.9	iterator	iterator baru per gelung; *fail fast*	koleksi menjadi iteratornya sendiri
16.10	objek nilai	imutabel; `equals` dengan `hashCode`	tanpa `hashCode`; kunci yang dapat berubah menjadi hantu
16.11	*race*	kunci, `synchronized`, atau atomik	`count++`, bahkan `volatile`; "ujinya lolos"
16.12	produsen–konsumen	`while` di sekitar `wait`; pil racun	`if` di sekitar `wait`
16.13	protokol	fungsi murni; *flush* setiap permintaan	tanpa *flush*: permintaan tidak pernah berangkat
16.14	seluruh buku	lapisan, MVC, pembingkaian	membaca satu baris dari balasan banyak baris

16.12 Seluruh Buku dalam Satu Tabel

Enam belas bab yang lalu, buku ini dimulai dengan program yang mencetak satu baris. **Tabel 16.12** adalah jalan dari sana menuju proyek puncak, satu baris per bab, dengan pertanyaan yang dijawab setiap bab dan tempat jawabannya dipakai dalam bab ini.

Tabel 16.12 Seluruh buku dalam satu tabel: pertanyaan setiap bab dan tempat jawabannya dipakai dalam bab ini.

Bab	Pertanyaan yang dijawabnya	Dipakai dalam bab ini oleh
1	Bagaimana menulis dan menjalankan program dalam Java dan Python?	setiap kode; README kode bab ini
2	Bagaimana menyunting, menjalankan, dan men-*debug* dalam suatu lingkungan?	menjalankan dua terminal; men-*debug thread*
3	Apa itu kelas, objek, *string*, tipe, dan perulangan?	setiap kelas; `describe`, `String.format`
4	Bagaimana beranjak dari spesifikasi ke kelas yang teruji?	jalur pengerjaan soal dalam bab ini
5	Bagaimana memakai larik, *list*, masukan, eksepsi, dan berkas?	`PickupSchedule`, `FileJokeStore`, `FileOrderRepository`
6	Bagaimana mendokumentasikan, menguji, dan menulis antarmuka?	`JokeStore`, `View`, uji kontrak
7	Bagaimana pewarisan, polimorfisme, dan kontrol akses bekerja?	`Service`, `Wash`, `Express`; *field* privat
8	Bagaimana merancang beberapa kelas yang bekerja sama?	metode sepuluh langkah pada Subbab 16.2
9	Bagaimana koleksi dan generik bekerja?	*map*, *list*, *set*, antrean, `Future<T>`
10	Bagaimana membangun jendela dan menangani peristiwa?	`KnockKnockGui`, *event dispatch thread*
11	Bagaimana memisahkan model, *view*, dan *controller*?	Soal 16.7 dan 16.14
12	Bagaimana menggabungkan koleksi, jendela, dan MVC dalam satu program?	lapisan-lapisan proyek puncak
13	Bagaimana iterator dan objek imutabel bekerja?	Soal 16.9 dan 16.10; `Order`, `Joke`
14	Bagaimana *thread* berbagi data dengan aman?	Soal 16.5, 16.11, 16.12; toko
15	Bagaimana program berkomunikasi melalui jaringan?	Soal 16.3 sampai 16.8, 16.13, 16.14

Bacalah kolom tengah dari atas ke bawah, dan Anda akan melihat bahwa sifat pertanyaannya berubah di tengah jalan. Bab-bab awal bertanya "bagaimana saya menulis ini?"; bab-bab kemudian bertanya "bagaimana bagian-bagian ini bekerja sama, dan apa yang terjadi bila beberapa hal terjadi sekaligus?" Pada akhirnya, pemrograman berorientasi objek adalah keterampilan menjawab pertanyaan jenis kedua: membagi sistem menjadi objek-objek dengan tanggung jawab yang jelas dan kontrak yang jelas, sehingga setiap bagian dapat dipahami, diuji, dan diubah secara tersendiri.

16.13 Ke Mana Setelah Ini

Anda kini sanggup membangun aplikasi lengkap dalam dua bahasa. Tiga jenis pengetahuan akan membuat aplikasi berikutnya lebih baik, lebih cepat dibangun, dan lebih mudah diikuti orang lain: pola rancangan, perkakas *build*, dan *framework*.

16.13.1 Pola rancangan

Pola rancangan (*design pattern*) adalah penyelesaian bernama yang telah teruji untuk masalah yang berulang dalam rancangan berorientasi objek [6]. Anda sudah memakai beberapa di antaranya tanpa namanya, dan **Tabel 16.13** memberikan nama-namanya. Mengenal nama-nama itu memungkinkan sebuah tim mengatakan dalam satu kata apa yang sebaliknya memerlukan satu paragraf ("toko adalah subjeknya, *view*-*view* adalah pengamatnya"). Katalog klasiknya ditulis Gamma dkk. [6]; pengantar yang lebih ramah ditulis Freeman dan Robson [7].

Tabel 16.13 Pola-pola rancangan yang dipakai dalam buku ini, menurut namanya.

Pola	Tujuannya	Tempat Anda bertemu dengannya
Iterator	menelusuri koleksi tanpa mengetahui strukturnya	Soal 16.9
Observer	memberi tahu banyak objek tentang perubahan pada satu objek	*listener* toko, Bab 10–11
Model–View–Controller	memisahkan data, tampilan, dan penanganan masukan	Bab 11, Soal 16.7, 16.14
Strategy	menjadikan sebuah algoritma objek yang dapat diganti	`PriceRule`, fungsi pengguna pada klien
Template method	menetapkan langkah-langkah, membiarkan subkelas mengisi sebagian	`Service.priceFor`, Bab 8
Factory method	membuat objek tanpa menyebut nama kelasnya	`JokeStore.of`, `Money.of`
Repository	menyembunyikan penyimpanan di balik antarmuka mirip koleksi	`JokeStore`, `OrderRepository`
Producer–consumer	memisahkan produsen dan konsumen dengan penyangga	Soal 16.12
State	membuat perilaku objek bergantung pada keadaannya	`KnockKnockProtocol`

Satu pesan kehati-hatian menyertai pola: pola adalah penyelesaian bagi masalah, bukan tujuan. Kelas yang memakai tiga pola untuk menjumlahkan dua bilangan lebih buruk daripada kelas yang sekadar menjumlahkannya. Raihlah sebuah pola ketika Anda mengenali masalahnya.

16.13.2 Perkakas *build* dan lingkungan

Program-program dalam buku ini dikompilasi dengan `javac` dan dijalankan dengan `java` atau `python`, satu folder demi satu folder. Proyek sungguhan bergantung pada pustaka (JUnit, *driver* basis data, *framework* web), dan mengelolanya dengan tangan segera menjadi mustahil. *Perkakas build* menggambarkan sebuah proyek dalam satu berkas dan mengerjakan sisanya.

Di Java, dua perkakas bakunya adalah **Maven** [8] dan **Gradle** [9]. Proyek Maven digambarkan oleh berkas `pom.xml` yang menyebut nama proyek dan dependensinya; perintah `mvn test` lalu mengunduh dependensi, mengompilasi kode di `src/main/java`, mengompilasi dan menjalankan uji di `src/test/java`, dan mencetak laporan. Gradle melakukan hal yang sama dengan berkas `build.gradle` (atau `build.gradle.kts`) dan perintah `gradle test`. Keduanya mengikuti konvensi folder yang sama, yang dipahami setiap editor Java, dan keduanya dapat mengemas program menjadi satu berkas JAR yang dijalankan dengan `java -jar`.

Di Python, padanannya adalah *lingkungan virtual* dan **pip** [10]. Lingkungan virtual adalah folder pribadi berisi paket-paket terpasang untuk satu proyek, sehingga dua proyek dapat memakai versi berbeda dari pustaka yang sama:

```
python -m venv .venv                    create the environment (once)
.venv\Scripts\activate                  activate it on Windows
source .venv/bin/activate               activate it on macOS and Linux
pip install pytest                      install a package into it
pip freeze > requirements.txt           record the exact versions
pip install -r requirements.txt         recreate the same environment elsewhere
```

Perintah-perintah di atas berturut-turut membuat lingkungan (sekali saja), mengaktifkannya di Windows, mengaktifkannya di macOS dan Linux, memasang sebuah paket ke dalamnya, mencatat versi-versi yang persis, dan membuat ulang lingkungan yang sama di tempat lain. Proyek Python

modern menggambarkan dirinya dalam `pyproject.toml`, dan `pytest`, *test runner* yang paling luas dipakai, menjalankan uji `unittest` tanpa perubahan di samping gayanya sendiri yang lebih ringkas. Profitina sendiri mengunci setiap dependensinya dalam sebuah berkas kunci (*lock file*), dengan alasan yang sama dengan keberadaan `requirements.txt`: program yang berjalan hari ini harus tetap berjalan setelah instalasi berikutnya.

Dua perkakas lagi layak menyertai setiap proyek sejak hari pertama: **Git**, untuk kendali versi, yang mencatat setiap perubahan dan memungkinkan sebuah tim bekerja pada berkas yang sama; dan *continuous integration* (integrasi berkelanjutan), yaitu server yang menjalankan seluruh rangkaian uji pada setiap perubahan, bentuk otomatis dari Langkah 6 pada Subbab 16.2.

16.13.3 *Framework*

Framework adalah pustaka yang memanggil kode Anda, bukan sebaliknya: Anda menulis kelas-kelasnya, dan *framework* menentukan kapan membuatnya dan memanggil metode-metodenya. Server-server soket dalam bab ini adalah persis apa yang dikerjakan *framework* untuk Anda, dengan HTTP alih-alih protokol baris kita, dan dengan kepedulian yang jauh lebih besar terhadap keamanan dan kinerja. Di Java, **Spring Boot** [11] adalah *framework* yang paling luas dipakai untuk layanan web: dalam Stack Overflow Developer Survey 2025, ia satu-satunya *framework* Java di antara *framework* web yang didaftar, dipakai oleh 14,7% responden pertanyaan itu [12]; dan **JavaFX** adalah penerus modern Swing untuk jendela desktop. Di Python, **FastAPI** [13] membangun API web dari fungsi-fungsi biasa dengan petunjuk tipe, **Django** adalah *framework* web lengkap dengan lapisan basis data dan situs administrasi, dan **Flask** adalah *framework* yang kecil; survei yang sama mencatat 14,8%, 12,6%, dan 14,4% responden memakainya [12]. Profitina, yang bentuknya diuraikan di Subbab 16.14, dibangun dengan Python dan FastAPI.

Semua isi buku ini tetap berlaku. *Controller* Spring adalah `Controller` pada Soal 16.14 dengan anotasi; *endpoint* FastAPI adalah fungsi murni `reply` pada Soal 16.13 yang dibungkus HTTP; entitas JPA atau model Django adalah kelas seperti `Order` beserta repositorinya. Mempelajari sebuah *framework* setelah buku ini berarti mempelajari di mana *framework* itu meletakkan potongan-potongan yang sudah Anda kenal.

> **JALUR BACAAN.** Untuk Java: Bloch, *Effective Java* [14], lalu Goetz dkk., *Java Concurrency in Practice* [15], lalu panduan-panduan Spring Boot. Untuk Python: Ramalho, *Fluent Python* [16], lalu tutorial FastAPI. Untuk keduanya: Gamma dkk. [6] untuk pola, *Refactoring* karya Fowler [17] untuk memperbaiki kode yang sudah berjalan, dan Brooks, *The Mythical Man-Month* [18], untuk memahami mengapa menambah orang ke proyek yang terlambat justru membuatnya makin terlambat.

16.14 Profitina dalam Praktik

Profitina, platform visibilitas AI (GEO) yang berjalan di lingkungan produksi pada app.profitina.com, mengukur seberapa siap situs web sebuah merek untuk dikutip mesin penjawab AI lalu membantu memperbaiki apa yang ditemukannya. Profitina adalah aplikasi Python dan FastAPI yang dilayani Uvicorn, dan hampir setiap soal dalam bab ini memiliki padanan dalam bentuk publiknya.

Kontrak antarbagian. Pemindai Profitina memeriksa sebuah situs melalui sekumpulan *sinyal*, masing-masing diberi skor dan dijelaskan secara mandiri di balik satu antarmuka bersama, yaitu gagasan `JokeStore` pada Soal 16.1 dalam skala produk: sinyal baru berarti implementasi baru, yang diperiksa oleh uji kontrak yang sama. Seluruh rangkaian uji otomatisnya, jauh lebih dari

seribu uji dalam Python ditambah penjaga JavaScript untuk antarmukanya, harus lolos sebelum rilis apa pun: rencana uji Soal 16.4 sampai 16.6, dijadikan sebuah gerbang.

Server waktu nyata dengan banyak klien. Fitur komunitas Profitina mendorong kiriman baru ke setiap peramban yang terbuka melalui *Server-Sent Events*: koneksi berumur panjang tempat server menulis sebaris setiap kali sesuatu terjadi. Itulah server Soal 16.3 yang dibungkus HTTP, dan ia menghadapi pertanyaan yang sama: setiap koneksi memiliki keadaannya sendiri, data bersama harus imutabel atau dilindungi, dan klien yang menghilang harus disadari lalu dibereskan. Dasbornya mengumpulkan semua yang diperlukan pengguna dalam satu permintaan, `GET /api/me/dashboard`, yang merupakan rancangan permintaan–balasan Soal 16.13: satu permintaan, satu balasan yang lengkap.

Kegagalan yang ditangani dengan jujur. Ketika pengguna mengirim kiriman, kotak isiannya langsung kosong dan kirimannya langsung tampak, dan bila pengiriman gagal, teksnya dipulihkan: deteksi kegagalan di sisi klien pada Soal 16.6, yang diwujudkan dalam antarmuka. Penerapannya bekerja dengan cara yang sama di sisi server: sebuah rilis baru dilaporkan berhasil setelah versi yang diterapkan menjawab pemeriksaan kesehatannya di `/api/health`, yaitu permintaan yang satu-satunya tujuan adalah memastikan server benar-benar hidup. Dan ketika tidak ada model AI lokal yang tersedia, aplikasi itu *mengatakannya* alih-alih diam-diam menyajikan templat, sesuai prinsip Soal 16.6 bahwa pengguna harus selalu diberi tahu apa yang sebenarnya terjadi.

Data pribadi, ditangani di satu tempat. Fitur akun Profitina dibangun untuk undang-undang pelindungan data pribadi Indonesia, UU PDP 27/2022, dengan persetujuan eksplisit, ekspor data, dan penghapusan akun secara menyeluruh. Aturan semacam ini hanya dapat ditaati bila data pribadi diubah di satu tempat yang terkendali, sebagaimana `Shop` yang tersinkronisasi pada Soal 16.14 adalah satu-satunya tempat pesanan berubah.

Seperti biasa, produk itu memberi saran dan penggunanya yang memutuskan: skor visibilitas adalah pengukuran beserta penjelasannya, bukan jaminan. Versi yang dapat dijalankan dari setiap gagasan di atas adalah kode Profitina Laundry dalam bab ini.

16.15 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya

Menyepakati kontrak hanya secara lisan. Tulislah setiap kontrak sebagai antarmuka atau tabel protokol, beserta uji kontrak yang dijalankan setiap implementasi (Soal 16.1, 16.2).

Protokol tanpa jawaban untuk setiap masukan. Diagram alir memperlihatkan jalur mulus; tabel keadaan dengan baris “masukan lain” di setiap keadaan memperlihatkan sisanya (Soal 16.2).

Tanpa aturan pembingkaian. Tentukan bagaimana penerima mengetahui di mana sebuah pesan berakhir: satu baris per pesan, atau baris penutup untuk balasan banyak baris (Soal 16.2, 16.14).

Melayani klien satu per satu. Server yang menangani klien di dalam gelung `accept`-nya menelantarkan semua yang lain; berikan setiap koneksi *thread* atau tugasnya sendiri (Soal 16.3).

Memakai bersama keadaan per koneksi. Keadaan sebuah percakapan adalah milik koneksinya. Pakailah bersama hanya objek yang imutabel atau aman-*thread* (Soal 16.3, 16.5).

Uji yang tidak mungkin gagal. Periksa hasil yang persis, bukan “sesuatu terjadi”; rusakkan kode dengan sengaja dan pastikan ada uji yang menyadarinya (Soal 16.4).

Panggilan menahan tanpa batas waktu. `connect`, pembacaan, atau `join` tanpa batas dapat menggantung program beserta ujinya selamanya. Pakailah batas waktu dan tenggat (Soal 16.5, 16.6).

Melupakan akhir *stream*. `readLine()` mengembalikan `null` di Java dan `readline()` mengembalikan `""` di Python ketika pihak lain menutup koneksi; periksalah pada setiap pembacaan (Soal 16.6).

Kode jaringan pada *thread* antarmuka pengguna. Biarkan *event dispatch thread* dan gelung utama Tk tetap bebas; kerjakan semua penantian di *thread* lain dan kirimkan pembaruan *view* kembali (Soal 16.7).

Klaim kinerja dari satu kali jalan. Lakukan pemanasan, ulangi, laporkan median dan maksimum, dan nyatakan kondisinya (Soal 16.8).

Mengembalikan `this` dari `iterator()`. Setiap gelung memerlukan iteratornya sendiri; dan objek generator Python hanya dapat ditelusuri satu kali (Soal 16.9).

`equals` tanpa `hashCode`, dan kunci yang dapat berubah. Override keduanya dari *field* yang sama, dan buatlah kunci imutabel (Soal 16.10).

Percaya bahwa uji yang lolos membuktikan tidak ada *race*. Lindungi setiap variabel yang dibaca dan ditulis beberapa *thread*, dan lebarkan celahnya dengan sengaja untuk menguji perbaikannya (Soal 16.11).

`if` di sekitar `wait`. Selalu `while`; pakailah `notifyAll`; satu pil racun per konsumen (Soal 16.12).

Lupa melakukan *flush*. *Writer* berpenyangga menahan permintaan untuk dirinya sendiri; lakukan *flush* setelah setiap permintaan, atau pakailah *autoflush* (Soal 16.13).

16.16 Ringkasan Bab dan Kata Penutup

- Proyek tim dihadapi dalam sepuluh langkah (Subbab 16.2): tujuan yang kecil, cerita pengguna, bagian beserta pemiliknya, kontrak lebih dulu, protokol sebagai mesin keadaan, uji silang dengan *stub*, kerangka yang berjalan, konkurensi dan penanganan kegagalan yang disengaja, pengukuran dan dokumentasi, serta laporan yang ditulis sambil pekerjaan berjalan.
- Kontrak adalah antarmuka atau protokol yang ditulis sebelum kode, masing-masing dengan uji yang harus dilalui setiap implementasi (16.1). Protokol dispesifikasikan dengan mesin keadaan, Persamaan (16.1), dan tabel transisi dengan baris untuk setiap masukan (16.2).
- Server banyak-klien memberi setiap koneksi *thread* dan keadaannya sendiri; objek bersama bersifat imutabel atau aman-*thread* (16.3). Setiap pihak menguji pihak lain melalui kontrak, dengan *stub* (16.4); sistem yang berjalan diuji dengan banyak klien yang dilepas oleh gerbang mulai dan diperiksa terhadap orakel (16.5).
- Setiap panggilan yang menahan memiliki batas waktu; akhir *stream* adalah hasil yang wajar; setiap kegagalan menjadi pesan yang jelas bagi pengguna atau entri catatan di server, dan server tetap melayani (16.6). Jendela tidak pernah menunggu jaringan (16.7). Kriteria kinerja diukur, diulang, dan dinyatakan beserta kondisinya, Persamaan (16.2) dan Persamaan (16.3) (16.8).
- *Iterable* menyerahkan iterator baru untuk setiap gelung, dan iteratornya dapat *fail fast*, Persamaan (16.4) (16.9). Objek nilai bersifat imutabel dan `hashCode`-nya sejalan dengan `equals` (16.10).

- *Race* menghilangkan pembaruan, Persamaan (16.5) dan Persamaan (16.6); `synchronized`, kunci, dan variabel atomik menawarnya, dan uji yang lolos tidak membuktikan ketiadaannya (16.11). Penyangga terbatas menunggu dalam gelung `while` dan menghentikan konsumennya dengan satu pil per konsumen (16.12).
- Protokol permintaan–balasan paling baik ditulis sebagai fungsi murni dan harus melakukan *flush* pada setiap permintaan (16.13). Proyek puncak menggabungkan setiap bab buku ini menjadi satu aplikasi berjaringan yang berlapis dan teruji (16.14).

Kata penutup. Ketika Anda membuka buku ini, kelas barangkali hanyalah kata yang pernah Anda dengar dan objek adalah sesuatu di atas meja Anda. Kini Anda telah menulis kelas-kelas yang saling berjanji dan menepati janjinya, objek-objek yang bercakap melintasi jaringan, *thread-thread* yang berbagi pekerjaan tanpa merusaknya, dan jendela-jendela yang tetap hidup walaupun dunia sedang lamban. Yang lebih penting daripada semua program itu adalah kebiasaan berpikir di baliknya: menanyakan, tentang setiap bagian sebuah sistem, apa yang diketahuinya, apa yang dikerjakannya, siapa yang dipercayainya, dan apa yang dijanjikannya; menuliskan janji itu; dan mengujinya sebelum memercayainya. Kebiasaan itu bukan milik Java atau Python. Ia akan melayani Anda dalam bahasa dan *framework* yang belum ada, dalam tim yang belum Anda masuki, dan pada masalah yang belum pernah diajukan siapa pun. Bangunlah sesuatu yang Anda pedulikan dengan kebiasaan itu. Bangunlah bersama orang lain, tuliskan apa yang kalian janjikan satu sama lain, dan ujilah. Lalu bangunlah hal berikutnya sedikit lebih baik. Itulah yang dilakukan para pemrogram, dan Anda kini salah seorang di antara mereka.

16.17 Soal Latihan

Latihan 16.1, 16.2, 16.3, 16.6, 16.8, dan 16.10 meminta kode; penyelesaiannya berupa program di Subbab 16.18. Latihan lainnya meminta penalaran dan dijawab dengan kata-kata.

1. **(Hitung mundur.)** Tulislah kelas `Countdown` yang merupakan `Iterable<Integer>` dan menghasilkan $n, n-1, \ldots, 1$, seperti layar lift yang sedang turun. Di Python, tulislah sebagai fungsi generator. Apa yang dilakukan gelung atas `Countdown(0)`?
2. **(Menghapus di dalam gelung.)** Daftar jam penjemputan adalah `[9, 14, 11, 17, 9]`. Hapuslah setiap jam sebelum pukul 12 dengan dua cara yang benar di Java, dan satu cara di Python. Mengapa gelung for-each yang memanggil `list.remove` gagal?
3. **(Membagi uang.)** Tulislah `split(Money m, int n)` yang membagi sejumlah uang menjadi n bagian yang berselisih paling banyak satu satuan dan jumlahnya tepat sama dengan semula. Berapakah `split(IDR 10,000, 3)`?
4. **(*Race* dengan tangan.)** Dua *thread* masing-masing menjalankan `count++` tiga kali pada pencacah tak tersinkronisasi yang bermula dari 0. Berapakah nilai akhir terbesar dan terkecil yang mungkin? Uraikan selang-seling yang menghasilkan nilai terkecil.
5. **(*Deadlock*.)** Dua mesin cuci masing-masing memegang antrean pesanan yang dilindungi kuncinya sendiri. Untuk memindahkan pesanan dari mesin A ke mesin B, sebuah *thread* mengunci antrean A, lalu antrean B; untuk memindahkan dari B ke A, *thread* lain mengunci B, lalu A. Tunjukkan bagaimana kedua *thread* dapat mengalami *deadlock*, dan berikan aturan yang mencegahnya.
6. **(Permintaan TOTAL.)** Perluas LAUNDRY/1 dengan `TOTAL <service> <kg> [<service> <kg> ...]`, yang mengembalikan `OK` dan jumlah harga-harganya, atau balasan `ERR` yang pertama. Pakailah ulang `reply`. Apa balasan `TOTAL WASH 3 IRON 2`?

7. **(Detak.)** Rancanglah *heartbeat* untuk protokol *knock-knock* sehingga server menyadari dalam 15 detik bahwa kabel sebuah klien tercabut. Pesan apa yang ditambahkan, siapa yang mengirimnya, dan apa yang berubah pada **Tabel 16.5**?
8. **(Penyimpan berbasis data.)** Tulislah `SqliteJokeStore` dalam Python dengan modul `sqlite3` dan tunjukkan bahwa ia menepati kontrak Soal 16.1. Apa yang diperlukan `JdbcJokeStore` dalam Java yang tidak disertakan JDK?
9. **(Pemecahan masalah.)** Tulislah entri pemecahan masalah, dengan gaya **Tabel 16.8**, untuk "jendela menampilkan *Connected* tetapi tidak ada yang muncul" dan untuk "konsol server menampilkan `Too many open files`".
10. **(Pengamat melalui jaringan.)** Tambahkan ke proyek puncak sebuah *listener* yang mencacah berapa pesanan telah mencapai `READY`, dan ujilah melalui API soket: taruh dua pesanan, majukan satu sampai `READY` dan yang lain sampai `PICKED_UP`, lalu periksa cacahannya.
11. **(Rancangan uji.)** Anggota GUI pada Soal 16.1 memakai `converse` milik anggota klien. Daftarkan lima uji yang sebaiknya ditulis anggota GUI untuknya, masing-masing dengan masukan dan hasil yang diharapkan.
12. **(Ke mana setelah ini.)** Uraikan, dengan tata letak folder Maven, di mana `LaundryCore.java`, `ProfitinaLaundry.java`, dan sebuah kelas uji JUnit untuk `Shop` diletakkan, serta perintah apa yang menjalankan ujinya.

16.18 Penyelesaian Soal Latihan

Latihan 1, 2, 3, 6, dan 10. **Kode 16.35** memuat penyelesaian Java, masing-masing diperiksa terhadap nilai-nilai yang dihitung di bawah; kompilasilah bersama berkas-berkas lain bab ini (`javac *.java`), karena program ini memakai `Money`, `LaundryProtocol`, `LaundryCore`, dan `ProfitinaLaundry`.

Kode 16.35 `ExerciseSolutions.java`: `Countdown`, penghapusan yang benar di dalam gelung, `split`, `TOTAL`, dan pencacah `READY` yang diuji melalui jaringan.

```java
// ExerciseSolutions.java - penyelesaian soal latihan pemrograman Bab 16.
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import java.util.NoSuchElementException;
import java.util.concurrent.atomic.AtomicInteger;

public class ExerciseSolutions {

    /** Latihan 16.1: n, n-1, ..., 1, seperti layar lift yang sedang turun. */
    static final class Countdown implements Iterable<Integer> {
        private final int from;

        Countdown(int from) {
            this.from = from;
        }

        @Override
        public Iterator<Integer> iterator() {
            return new Iterator<>() {
                private int next = from;

                @Override
                public boolean hasNext() {
                    return next >= 1;
                }

                @Override
                public Integer next() {
                    if (!hasNext()) {
                        throw new NoSuchElementException();
                    }
                    return next--;
                }
            };
        }
    }
```

Kode 16.35 (Lanjutan) *ExerciseSolutions.java: Countdown, penghapusan yang benar di dalam gelung, split, TOTAL, dan pencacah READY yang diuji melalui jaringan.*

```
    /** Latihan 16.2: menghapus penjemputan sebelum pukul 12 dengan benar, dengan dua cara. */
    static List<Integer> removeMorningWithIterator(List<Integer> hours) {
        List<Integer> copy = new ArrayList<>(hours);
        Iterator<Integer> it = copy.iterator();
        while (it.hasNext()) {
            if (it.next() < 12) {
                it.remove();                        // remove milik iterator sendiri diizinkan
            }
        }
        return copy;
    }

    static List<Integer> removeMorningWithRemoveIf(List<Integer> hours) {
        List<Integer> copy = new ArrayList<>(hours);
        copy.removeIf(h -> h < 12);
        return copy;
    }

    /** Latihan 16.3: n bagian yang berselisih paling banyak satu satuan dan jumlahnya tepat. */
    static List<Money> split(Money m, int n) {
        if (n <= 0) {
            throw new IllegalArgumentException("n must be positive: " + n);
        }
        long base = m.amount() / n;
        long extra = m.amount() % n;                // sebanyak "extra" bagian pertama mendapat satu satuan lebih
        List<Money> parts = new ArrayList<>();
        for (int i = 0; i < n; i++) {
            parts.add(Money.of(base + (i < extra ? 1 : 0), m.currency()));
        }
        return parts;
    }

    /** Latihan 16.6: TOTAL WASH 3 IRON 2 menjumlahkan beberapa balasan PRICE. */
    static String total(String request) {
        String[] w = request.trim().split("\\s+");
        if (w.length < 3 || w.length % 2 == 0) {
            return "ERR usage: TOTAL <service> <kg> [<service> <kg> ...]";
        }
        long sum = 0;
        for (int i = 1; i < w.length; i += 2) {
            String one = LaundryProtocol.reply("PRICE " + w[i] + " " + w[i + 1]);
            if (one.startsWith("ERR")) {
                return one;                         // galat pertama yang menentukan
            }
            sum += Long.parseLong(one.substring(3));
        }
```

Kode 16.35 (Lanjutan) *ExerciseSolutions.java: Countdown, penghapusan yang benar di dalam gelung, split, TOTAL, dan pencacah READY yang diuji melalui jaringan.*

```
        return "OK " + sum;
    }

    private static int passed = 0;

    private static void check(String name, Object expected, Object actual) {
        if (!java.util.Objects.equals(expected, actual)) {
            throw new AssertionError(name + ": expected " + expected + " but got " +
actual);
        }
        passed++;
        System.out.println("PASS " + name);
    }

    public static void main(String[] args) throws Exception {
        List<Integer> seen = new ArrayList<>();
        for (int floor : new Countdown(4)) {
            seen.add(floor);
        }
        check("16.1 countdown", List.of(4, 3, 2, 1), seen);
        check("16.1 empty countdown", false, new Countdown(0).iterator().hasNext());

        List<Integer> hours = List.of(9, 14, 11, 17, 9);
        check("16.2 iterator remove", List.of(14, 17), removeMorningWithIterator(hours));
        check("16.2 removeIf", List.of(14, 17), removeMorningWithRemoveIf(hours));

        List<Money> parts = split(Money.rupiah(10_000), 3);
        check("16.3 parts", "[IDR 3,334, IDR 3,333, IDR 3,333]", parts.toString());
        check("16.3 nothing lost", Money.rupiah(10_000),
parts.get(0).plus(parts.get(1)).plus(parts.get(2)));

        check("16.6 total", "OK 31000", total("TOTAL WASH 3 IRON 2"));
        check("16.6 error", "ERR unknown service SILK", total("TOTAL WASH 3 SILK 1"));

        // Latihan 16.10: pengamat yang mencacah pesanan yang mencapai READY, diuji melalui
jaringan.
        LaundryCore.Shop shop = new LaundryCore.Shop();
        AtomicInteger ready = new AtomicInteger();
        shop.addListener(o -> {
            if (o.status() == LaundryCore.Status.READY) {
                ready.incrementAndGet();
            }
        });
        ProfitinaLaundry.SocketApi api = new ProfitinaLaundry.SocketApi(
                new ProfitinaLaundry.Controller(shop, new LaundryCore.FileOrderRepository(
                        java.nio.file.Path.of("unused.csv"))));
        int port = api.start(0);
        try (ProfitinaLaundry.ApiClient c = new ProfitinaLaundry.ApiClient(port)) {
            c.call("PLACE Sari WASH 3");
```

Kode 16.35 (Lanjutan) *ExerciseSolutions.java: Countdown, penghapusan yang benar di dalam gelung, split, TOTAL, dan pencacah READY yang diuji melalui jaringan.*

```
            c.call("PLACE Budi IRON 1");
            for (String id : new String[] {"L1", "L1", "L2", "L2", "L2"}) {
                c.call("ADVANCE " + id);            // L1 -> READY; L2 -> READY ->
PICKED_UP
            }
        }
        api.stop();
        check("16.10 two orders reached READY", 2, ready.get());
        System.out.println("All " + passed + " checks passed.");
    }
}
```

```
PASS 16.1 countdown
PASS 16.1 empty countdown
PASS 16.2 iterator remove
PASS 16.2 removeIf
PASS 16.3 parts
PASS 16.3 nothing lost
PASS 16.6 total
PASS 16.6 error
PASS 16.10 two orders reached READY
All 9 checks passed.
```

Kode 16.36 memuat penyelesaian Python, termasuk Latihan 8.

Kode 16.36 `exercise_solutions.py`: penyelesaian yang sama dalam Python, dan penyimpan lelucon SQLite untuk Latihan 8.

```
# exercise_solutions.py - penyelesaian soal latihan pemrograman Bab 16.
import os
import sqlite3
import unittest
from joke_store import Joke, JokeStore
from joke_stores import contract
from laundry_core import FileOrderRepository, Shop, Status
from laundry_protocol import reply
from money import Money
from profitina_laundry import ApiClient, Controller, SocketApi


def countdown(start):
    """Latihan 16.1: start, start-1, ..., 1, seperti layar lift yang sedang turun."""
    while start >= 1:
        yield start
        start -= 1


def remove_morning(hours):
    """Latihan 16.2: list baru tanpa penjemputan sebelum pukul 12."""
    return [h for h in hours if h >= 12]


def split(money, n):
    """Latihan 16.3: n bagian yang berselisih paling banyak satu satuan dan jumlahnya
tepat."""
    if n <= 0:
        raise ValueError(f"n must be positive: {n}")
    base, extra = divmod(money.amount, n)       # sebanyak "extra" bagian pertama mendapat
satu satuan lebih
    return [Money(base + (1 if i < extra else 0), money.currency) for i in range(n)]


def total(request):
    """Latihan 16.6: TOTAL WASH 3 IRON 2 menjumlahkan beberapa balasan PRICE."""
    w = request.split()
    if len(w) < 3 or len(w) % 2 == 0:
        return "ERR usage: TOTAL <service> <kg> [<service> <kg> ...]"
    s = 0
    for service, kg in zip(w[1::2], w[2::2]):
        one = reply(f"PRICE {service} {kg}")
        if one.startswith("ERR"):
            return one                          # galat pertama yang menentukan
        s += int(one[3:])
    return f"OK {s}"
```

Kode 16.36 (Lanjutan) *exercise_solutions.py: penyelesaian yang sama dalam Python, dan penyimpan lelucon SQLite untuk Latihan 8.*

```
class SqliteJokeStore(JokeStore):
    """Latihan 16.8: penyimpan milik anggota basis data, di atas SQLite (bagian pustaka
standar Python)."""

    def __init__(self, jokes):
        self._db = sqlite3.connect(":memory:", check_same_thread=False)
        self._db.execute("CREATE TABLE joke (n INTEGER PRIMARY KEY, clue TEXT, answer
TEXT)")
        self._db.executemany("INSERT INTO joke VALUES (?, ?, ?)",       # ? menjaga data
tetap di luar teks SQL
                             [(i, j.clue, j.answer) for i, j in enumerate(jokes)])
        self._db.commit()

    def size(self):
        return self._db.execute("SELECT COUNT(*) FROM joke").fetchone()[0]

    def get(self, index):
        row = self._db.execute("SELECT clue, answer FROM joke WHERE n = ?",
(index,)).fetchone()
        if row is None:
            raise IndexError(f"joke {index} does not exist")
        return Joke(*row)


class TestExercises(unittest.TestCase):
    def test_countdown(self):
        self.assertEqual([4, 3, 2, 1], list(countdown(4)))
        self.assertEqual([], list(countdown(0)))

    def test_remove_morning(self):
        self.assertEqual([14, 17], remove_morning([9, 14, 11, 17, 9]))

    def test_split(self):
        parts = split(Money(10_000), 3)
        self.assertEqual(["IDR 3,334", "IDR 3,333", "IDR 3,333"], [str(p) for p in parts])
        self.assertEqual(Money(10_000), parts[0] + parts[1] + parts[2])

    def test_total(self):
        self.assertEqual("OK 31000", total("TOTAL WASH 3 IRON 2"))
        self.assertEqual("ERR unknown service SILK", total("TOTAL WASH 3 SILK 1"))

    def test_sqlite_store_keeps_the_contract(self):
        store = SqliteJokeStore([Joke("Lettuce", "Lettuce in, it's cold out here!"),
                                 Joke("Harry", "Harry up, the laundry is ready!")])
        self.assertEqual([], contract(store))
        self.assertEqual("Harry", store.get(1).clue)
```

Kode 16.36 (Lanjutan) *exercise_solutions.py: penyelesaian yang sama dalam Python, dan penyimpan lelucon SQLite untuk Latihan 8.*

```
    def test_ready_counter_over_the_network(self):
        shop, ready = Shop(), []
        shop.add_listener(lambda o: ready.append(o.id) if o.status is Status.READY else
None)
        api = SocketApi(Controller(shop, FileOrderRepository(os.devnull)))
        with ApiClient(api.start(0)) as c:
            c.call("PLACE Sari WASH 3")
            c.call("PLACE Budi IRON 1")
            for order_id in ("L1", "L1", "L2", "L2", "L2"):
                c.call(f"ADVANCE {order_id}")   # L1 -> READY; L2 -> READY -> PICKED_UP
        api.stop()
        self.assertEqual(["L1", "L2"], ready)


if __name__ == "__main__":
    unittest.main(verbosity=2)
```

```
test_countdown (__main__.TestExercises.test_countdown) ... ok
test_ready_counter_over_the_network
(__main__.TestExercises.test_ready_counter_over_the_network) ... ok
test_remove_morning (__main__.TestExercises.test_remove_morning) ... ok
test_split (__main__.TestExercises.test_split) ... ok
test_sqlite_store_keeps_the_contract
(__main__.TestExercises.test_sqlite_store_keeps_the_contract) ... ok
test_total (__main__.TestExercises.test_total) ... ok

----------------------------------------------------------------------
Ran 6 tests in 0.003s

OK
```

1. Lihat `Countdown` dan `countdown`. Iterator Java menyimpan `next`-nya sendiri, bermula dari n; `hasNext()` adalah `next >= 1`. Gelung atas `Countdown(0)` berjalan nol kali, karena `hasNext()` langsung bernilai `false`; `list(countdown(0))` adalah `[]`.
2. Lihat `removeMorningWithIterator` dan `removeMorningWithRemoveIf`; keduanya menghasilkan `[14, 17]`. `Iterator.remove()` menghapus elemen yang terakhir dikembalikan `next()` dan memperbarui harapan iterator itu sendiri, sehingga tidak ada eksepsi yang menyusul; `removeIf` mengerjakan seluruh gelung di dalam *list*. Gelung for-each yang memanggil `list.remove` mengubah *list* di belakang punggung iteratornya, dan panggilan `next()` berikutnya melempar `ConcurrentModificationException`, persis seperti pada Soal 16.9. Di Python, bangunlah *list* baru dengan *comprehension*, `[h for h in hours if h >= 12]`, alih-alih menghapus sambil beriterasi.
3. Dengan a jumlah uangnya dan n banyaknya bagian,

$$b = \lfloor \frac{a}{n} \rfloor, \qquad r = a \bmod n \tag{16.7}$$

artinya setiap bagian mendapat b dan r bagian pertama mendapat satu satuan lebih, sehingga bagian-bagian itu berjumlah $nb + r = a$. Menurut Persamaan (16.7), untuk $a = 10{,}000$ dan $n = 3$: $b = 3333$ dan $r = 1$, sehingga bagian-bagiannya 3.334, 3.333, dan 3.333. Mengalikan dengan $1/3$ lalu membulatkan akan menghasilkan tiga kali 3.333 dan kehilangan satu rupiah. 4. Nilai terbesar adalah 6, ketika *thread-thread* sama sekali tidak berselang-seling. Nilai terkecil adalah 2. Selang-selingnya: *thread* A membaca 0; *thread* B menuntaskan dua penambahan (cacahan 2); A menulis 1; B membaca 1; A menuntaskan dua penambahan sisanya (cacahan 3); B menulis 2. Tulisan terakhir setiap *thread* dapat membatalkan kerja *thread* lain, tetapi penambahan pertama dan terakhir milik satu *thread* tidak dapat sama-sama hilang, dan itulah sebabnya minimumnya 2 dan

bukan 1, berapa pun banyaknya penambahan per *thread*. 5. *Thread* 1 mengunci A dan menunggu B; *thread* 2 mengunci B dan menunggu A; masing-masing menunggu kunci yang dipegang pihak lain, selamanya. Inilah situasi para filsuf yang makan malam dari Bab 14 dengan dua filsuf. Aturannya adalah *urutan penguncian global* [19]: berikan nomor kepada setiap antrean dan selalu kunci nomor yang lebih kecil lebih dulu. Kedua pemindahan lalu mengunci A sebelum B, dan penantian yang melingkar tidak mungkin timbul. (Batas waktu pada kunci kedua, `tryLock` di Java, adalah alternatif yang mengubah *deadlock* menjadi percobaan ulang.) 6. Lihat `total`. `TOTAL WASH 3 IRON 2` menghasilkan `OK 31000`, karena $7000 \times 3 + 5000 \times 2 = 21{,}000 + 10{,}000$. `TOTAL WASH 3 SILK 1` menghasilkan galat pertama, `ERR unknown service SILK`. 7. Server mengirim `PING` bila selama 5 detik ia tidak mengirim apa pun; klien harus segera menjawab `PONG`, dalam keadaan apa pun. Server memasang `setSoTimeout(15_000)` pada setiap soket klien; `SocketTimeoutException` berarti tiga detak terlewat, dan koneksi ditutup sebagai "klien hilang". Pada **Tabel 16.5**, setiap keadaan mendapat baris tambahan (`PONG`, tanpa balasan, keadaan tetap), sehingga jawaban detak tidak pernah dihitung sebagai baris yang salah. Klien, secara simetris, boleh memperlakukan 15 detik tanpa baris apa pun, baik detak maupun lelucon, sebagai "server hilang". 8. Lihat `SqliteJokeStore`. Ia membuat tabel `joke(n, clue, answer)`, menyisipkan lelucon-lelucon dengan penampung `?`, yang menjaga data tetap di luar teks SQL dan mencegah *SQL injection*, lalu menjawab `get(i)` dengan `SELECT ... WHERE n = ?`, memunculkan `IndexError` bila tidak ada baris yang ditemukan. Uji kontrak Soal 16.1 tidak melaporkan kegagalan. `JdbcJokeStore` dalam Java akan memakai antarmuka `java.sql` milik JDK (`Connection`, `PreparedStatement`, `ResultSet`) tanpa perubahan, tetapi memerlukan *driver JDBC* untuk basis data yang dipilih, yaitu berkas JAR seperti *driver* SQLite atau PostgreSQL, yang tidak disertakan JDK; itulah saatnya mulai memakai Maven atau Gradle (Subbab 16.13.2). 9. "Jendela menampilkan *Connected* tetapi tidak ada yang muncul": kemungkinan penyebabnya, server menerima koneksi tetapi tidak menanggapi (kelebihan beban, atau tersangkut pada percakapan sebelumnya, seperti server satu-per-satu pada Soal 16.3); obatnya, tunggu pesan batas waktu, lalu jalankan ulang server dan periksa konsolnya. "`Too many open files`": kemungkinan penyebabnya, server tidak menutup soket klien yang sudah selesai atau yang telah pergi, atau batas sistem operasinya rendah; obatnya, kesalahan pemrograman yang diperbaiki dengan `try`-with-resources atau `with` di sekeliling setiap soket (Soal 16.6), dan, untuk pengguna sungguhan yang sangat banyak, menaikkan batas sistem operasinya. 10. Lihat bagian akhir `ExerciseSolutions.main` dan `test_ready_counter_over_the_network`. *Listener*-nya memeriksa `o.status() == Status.READY`; setelah kelima perintah `ADVANCE`, L1 telah melewati `READY` satu kali dan L2 telah melewati `READY` dalam perjalanannya menuju `PICKED_UP`, sehingga cacahannya 2. Karena *listener* dipanggil dari *thread-thread* jaringan, cacahan Java berupa `AtomicInteger`. 11. Lima uji, dengan *stub* server dari Soal 16.4: (1) satu lelucon utuh, baris `Knock! Knock!`, `Lettuce`, `Lettuce in, ... (y/n)`, `Bye.` dengan pengguna yang menjawab `n`: klien menulis `Who's there?`, `Lettuce who?`, `n`; (2) server memutus setelah `Knock! Knock!`: `converse` kembali tanpa eksepsi dan transkripnya dua baris; (3) fungsi pengguna mengembalikan `null`: klien berhenti tanpa menulis apa pun lagi; (4) baris sesudah `Bye.` diabaikan; (5) baris dengan karakter non-ASCII (`Kopi ☕`) tiba utuh melalui *reader* dan *writer* UTF-8. 12. `src/main/java/LaundryCore.java` dan `src/main/java/ProfitinaLaundry.java` (pada proyek sungguhan di dalam folder paket seperti `src/main/java/com/profitina/laundry/`); kelas uji `src/test/java/ShopTest.java`, memakai API JUnit Jupiter (JUnit 5, atau JUnit 6, versi mayor terkininya [20]) sebagaimana dideklarasikan di `pom.xml`. Perintah `mvn test` mengompilasi kedua folder dan menjalankan ujinya; `mvn package` juga membangun sebuah JAR.

Daftar Pustaka

[1] Python Software Foundation, "unittest — Unit testing framework," *The Python Standard Library*, Python 3.14. [Online]. Available: https://docs.python.org/3/library/unittest.html

[2] G. J. Myers, C. Sandler, and T. Badgett, *The Art of Software Testing*, 3rd ed. Hoboken, NJ, USA: Wiley, 2011.

[3] J. D. C. Little, "A proof for the queuing formula: L = λW," *Operations Research*, vol. 9, no. 3, pp. 383–387, 1961.

[4] P. Deitel and H. Deitel, *Java How to Program, Early Objects*, 11th ed. Hoboken, NJ, USA: Pearson, 2017.

[5] J. Klensin, "Simple Mail Transfer Protocol," RFC 5321, Internet Engineering Task Force, Oct. 2008. [Online]. Available: https://www.rfc-editor.org/rfc/rfc5321

[6] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Reading, MA, USA: Addison-Wesley, 1994.

[7] E. Freeman and E. Robson, *Head First Design Patterns*, 2nd ed. Sebastopol, CA, USA: O'Reilly, 2020.

[8] The Apache Software Foundation, "Maven Getting Started Guide." [Online]. Available: https://maven.apache.org/guides/getting-started/

[9] Gradle Inc., "Gradle User Manual." [Online]. Available: https://docs.gradle.org/current/userguide/userguide.html

[10] Python Packaging Authority, "Installing packages using pip and virtual environments," *Python Packaging User Guide*. [Online]. Available: https://packaging.python.org/

[11] VMware Tanzu, "Spring Boot Reference Documentation." [Online]. Available: https://docs.spring.io/spring-boot/

[12] Stack Overflow, "2025 Stack Overflow Developer Survey: Technology — Web frameworks and technologies," Jul. 2025. [Online]. Available: https://survey.stackoverflow.co/2025/technology

[13] S. Ramírez, "FastAPI Documentation." [Online]. Available: https://fastapi.tiangolo.com/

[14] J. Bloch, *Effective Java*, 3rd ed. Boston, MA, USA: Addison-Wesley, 2018.

[15] B. Goetz, T. Peierls, J. Bloch, J. Bowbeer, D. Holmes, and D. Lea, *Java Concurrency in Practice*. Upper Saddle River, NJ, USA: Addison-Wesley, 2006.

[16] L. Ramalho, *Fluent Python*, 2nd ed. Sebastopol, CA, USA: O'Reilly, 2022.

[17] M. Fowler, *Refactoring: Improving the Design of Existing Code*, 2nd ed. Boston, MA, USA: Addison-Wesley, 2018.

[18] F. P. Brooks, Jr., *The Mythical Man-Month: Essays on Software Engineering*, anniversary ed. Reading, MA, USA: Addison-Wesley, 1995.

[19] E. W. Dijkstra, "Hierarchical ordering of sequential processes," *Acta Informatica*, vol. 1, no. 2, pp. 115–138, 1971.

[20] The JUnit Team, *JUnit User Guide*, version 6.1.3 (Aug. 2026), and "Release notes: 6.0.0" (Java 17 baseline). [Online]. Available: https://docs.junit.org/current/

[21] Oracle, "The Java Tutorials: All About Sockets" and "Lesson: Concurrency." [Online]. Available: https://docs.oracle.com/javase/tutorial/

[22] Python Software Foundation, "socket," "threading," "queue" and "dataclasses," *The Python Standard Library*, Python 3.14. [Online]. Available: https://docs.python.org/3/library/

[23] Oracle, "Lesson: Concurrency in Swing," *The Java Tutorials*. [Online]. Available: https://docs.oracle.com/javase/tutorial/uiswing/concurrency/

Kelas-kelas soket, protokol *knock-knock*, dan server banyak-kliennya mengikuti tutorial Java [21], dan versi Python-nya memakai modul socket, threading, queue, dan dataclasses dari pustaka standar [22]; kaidah bahwa komponen Swing hanya dipakai pada *event dispatch thread* berasal

dari pelajaran konkurensi Swing [23]. *Race*, monitor, `wait`/`notifyAll`, variabel atomik, dan rancangan kelas yang aman-*thread* mengikuti Goetz dkk. [15] dan Bloch [14], yang juga memberikan kontrak *hash* dan argumen bagi objek nilai yang imutabel; contoh produsen–konsumen mengikuti Deitel dan Deitel [4], dan urutan penguncian global yang mencegah *deadlock* bermula dari Dijkstra [19]. Hukum Little berasal dari Little [3]; balasan yang diakhiri titik adalah konvensi SMTP [5]; pengujian mutasi dan rancangan uji dibahas oleh Myers dkk. [2]; iterator dan generator di Python dijelaskan secara mendalam oleh Ramalho [16]. Pola-pola rancangan adalah pola dari Gamma dkk. [6] serta Freeman dan Robson [7]; *refactoring* berasal dari Fowler [17], dan pelajaran tentang tim dari Brooks [18]. Maven [8], Gradle [9], pip dan lingkungan virtual [10], Spring Boot [11], serta FastAPI [13] didokumentasikan oleh proyek masing-masing; uji Python memakai `unittest` [1].