Pemrograman Berorientasi Objek (EF234302)

Kuliah 1 — Objek, Program, dan Meja Kerja Pertama Anda

Buku: Irfan Subakti, *Pemrograman Berorientasi Objek dengan Java dan Python*, Bab 1

Dosen: Irfan Subakti — `yifana@gmail.com`

Departemen Teknik Informatika, Institut Teknologi Sepuluh Nopember, Surabaya

Hari ini sebuah kios laundry di sudut jalan Surabaya menjadi masyarakat objek, dan program pertama berjalan dalam kedua bahasa. Buku: Bab 1.

Agenda

1. Apa itu program: sebuah resep, dua rumus, dan bagaimana kode sumber menjadi program yang berjalan.

2. Berpikir dalam objek: keadaan, perilaku, identitas; kelas sebagai formulir kosong; pesan; abstraksi.

3. Mengapa prosedur saja tidak cukup: program buku catatan dan dua baris cerobohnya.

4. Mengapa Java dan mengapa Python; JDK, interpreter, dan VS Code di Windows, macOS, dan Linux.

5. Program pertama, sesi *debugging* pertama, dan kelas pertama: `Order` *laundry* yang menolak permintaan yang tidak masuk akal.

Buku: Bab 1, Subbab 1.1–1.14.

Tempat Kita Memulai

Kios Bu Sari: meja layanan dengan timbangan, tiga mesin cuci, dua pengering, rak kantong, dan sebuah buku catatan tebal.

Sudah penuh dengan gagasan kita: sebuah **prosedur** (timbang, tanya, catat, hitung harga, cuci, keringkan, lipat, serahkan), **informasi** (nama, berat, layanan, harga, status), dan **benda-benda** yang masing-masing mengetahui sesuatu dan dapat melakukan sesuatu.

Ketika mempekerjakan pegawai kedua, Bu Sari menjelaskan kiosnya *satu jenis benda demi satu jenis benda*: cara kerja nota, cara kerja mesin. Itulah pemrograman berorientasi objek.

> **WAWASAN KUNCI:** Program dapat berupa satu daftar panjang instruksi yang memindah-mindahkan angka, atau masyarakat kecil **objek** yang masing-masing menjaga informasinya sendiri. Cara kedua itulah cara sebagian besar perangkat lunak sehari-hari disusun.

Setiap gagasan ditunjukkan dengan kata-kata, lalu dalam Java, lalu dalam Python, lalu dijalankan, lalu dijelaskan baris demi baris. Buku: Subbab 1.1.

Resep untuk Meja Layanan

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
```

Setiap baris menyatakan satu hal; “round” dan “floor” tidak menyisakan ruang tafsir; baris 5 dikerjakan hanya *jika* baris 4 benar. Nilai yang berubah diberi nama pendek, w dan p.

> **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.

Pseudokode 1.1 adalah kartu yang ditempel Bu Sari di meja layanan untuk keponakannya, Dimas. Buku: Subbab 1.2.1 dan Pseudokode 1.1.

Harga sebagai Dua Rumus

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

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

$\lfloor x \rfloor$, "*floor* dari x", adalah bilangan bulat terbesar yang tidak lebih besar daripada x: $\lfloor 12250.0 \rfloor = 12250$, $\lfloor 7.9 \rfloor = 7$.

> **DENGAN TANGAN.** Ani, 3.5 kg biasa, menurut Persamaan (1.1): $p(3.5) = \text{round}(24500) = 24500$. Budi, 5 kg ekspres, menurut Persamaan (1.2): $p(5) = 35000$, lalu $35000 + \lfloor 35000/2 \rfloor = 52500$.

Ingat Rp 24.500 dan Rp 52.500: komputer akan mencetaknya beberapa kali hari ini. Buku: Subbab 1.2.1, Persamaan (1.1) dan (1.2).

Bahasa untuk Manusia dan untuk Mesin

Prosesor hanya memahami **bahasa mesin**: barisan panjang angka, masing-masing bermakna sesuatu yang sangat kecil.

Bahasa pemrograman cukup tepat untuk diterjemahkan tanpa menebak dan cukup mudah dibaca untuk dijelaskan kepada rekan. Baris 3 kartu itu, dalam kedua bahasa:

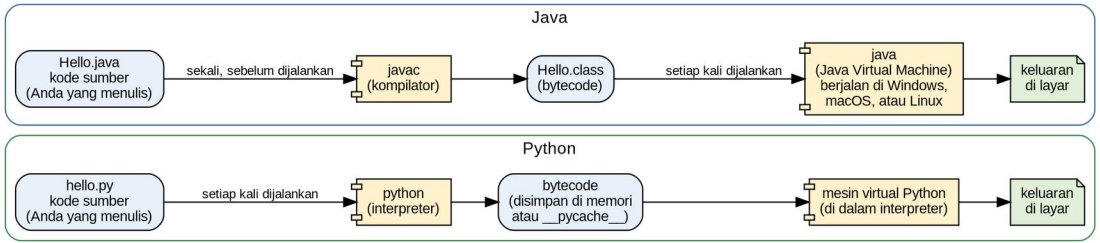
```
long p = Math.round(7000 * w);      Java
p = round(7000 * w)                 Python
```

Java menyebutkan *jenis* nilai yang disimpan `p` (`long`) dan mengakhiri pernyataan dengan `;`. Python membiarkan nilai menentukan jenisnya sendiri dan berakhir di ujung baris.

Teks yang Anda tulis adalah **kode sumber**; berkas sumber `.java` dan `.py` adalah teks biasa.

Bab 3 membahas perbedaan-perbedaan ini dengan cermat. Buku: Subbab 1.2.2.

Dari Kode Sumber menjadi Program yang Berjalan



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.

Gambar 1.1: kompilator menerjemahkan seluruh buku sekali; interpreter menerjemahkan sambil berjalan. Java dan Python masing-masing memakai sedikit dari keduanya. Buku: Subbab 1.2.3 dan Gambar 1.1.

Kompilasi, Lalu Jalankan — atau Langsung Jalankan

Java: `javac Hello.java` menulis `Hello.class`, yang berisi ***bytecode*** untuk **Java Virtual Machine**; `java Hello` menyalakan JVM yang menjalankannya. *Bytecode* yang sama berjalan di setiap mesin: *write once, run anywhere*.

Python: `python hello.py` mengompilasi ke *bytecode* Python di dalam interpreter (modul yang diimpor disimpan di `__pycache__`) lalu langsung menjalankannya.

Data dan operasi. Setiap program adalah data (3.5 kg, “Ani”, ekspres atau tidak) ditambah operasi terhadapnya. Kode prosedural memisahkan keduanya; kode berorientasi objek menyatukan setiap keping data *dengan* operasi miliknya.

> **PERINGATAN:** Di Java, setelah mengubah kode sumber Anda harus mengompilasi ulang, atau Anda akan menjalankan versi lama.

Buku: Subbab 1.2.3–1.2.4.

Melihat-lihat di Dalam Kios

Ajukan dua pertanyaan kepada setiap *benda*: **apa yang diketahuinya?** dan **apa yang dapat dilakukannya?**

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

Tabel 1.1, dan bukan kode apa pun, adalah titik awal sebuah program berorientasi objek. Buku: Subbab 1.3.1 dan Tabel 1.1.

Keadaan, Perilaku, dan Identitas

> **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).

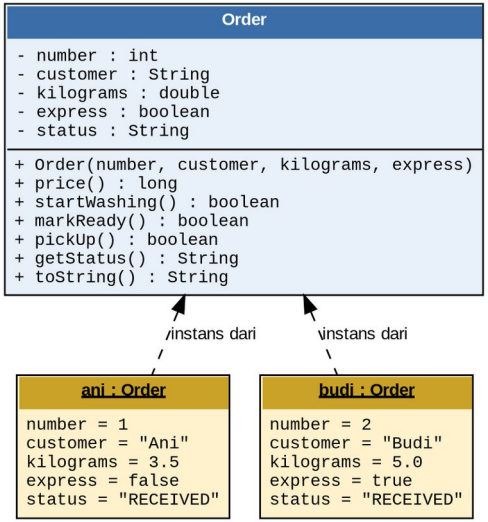
Dua pelanggan dengan kantong yang sama berat, layanan sama, harga sama: tetap **dua nota**. Menyerahkan pakaian Ani kepada Budi adalah kesalahan serius.

Keadaan berubah seiring waktu: RECEIVED pukul tujuh, WASHING pukul sepuluh, READY pukul dua. Notanya tetap nota yang sama.

Perilaku adalah *cara* keadaan berubah, melalui langkah-langkah yang terdefinisi — dan sebagian langkah **ditolak** pada saat tertentu: cucian yang masih di mesin tidak dapat diambil.

Buku: Subbab 1.3.2.

Kelas: Formulir Kosong



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.

Gambar 1.2 dalam UML: kelas adalah formulir kosong yang dicetak dalam satu blok; `ani` dan `budi` adalah dua nota yang sudah diisi. Buku: Subbab 1.3.3 dan Gambar 1.2.

Membaca Kotak Kelas

Atas: nama kelas. **Tengah:** ***field*** beserta tipenya — `int`, `String`, `double`, `boolean`; tanda minus berarti *private*.

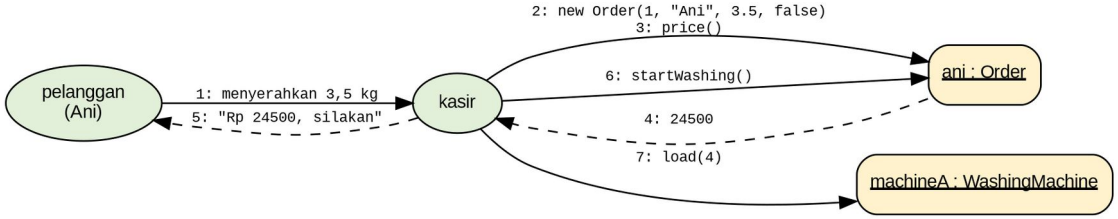
Bawah: **metode**, dengan tanda plus untuk *public*. `Order(number, customer, kilograms, express)` adalah **konstruktor**, yang mengisi formulir baru.

Kotak objek: `nama : Kelas` yang digarisbawahi, menampilkan *nilai*. Setiap objek punya salinan sendiri dari setiap *field*; mengubah status Ani tidak menyentuh status Budi.

> **WAWASAN KUNCI:** Kelas ditulis sekali, sebagai kode sumber. Objek diciptakan ketika program berjalan, sebanyak yang diperlukan. “Tulis sebuah kelas” berarti rancang formulir kosongnya; “buat sebuah objek” berarti isi satu formulir.

*Objek juga disebut **instance** dari kelasnya. Buku: Subbab 1.3.3.*

Pesan: Objek-Objek Bekerja Sama



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.

Pada Gambar 1.3 panah utuh adalah pesan, panah putus-putus adalah jawaban, dan angka menunjukkan urutannya. Buku: Subbab 1.3.4 dan Gambar 1.3.

Mengikuti Nomor-Nomornya

1 pelanggan menyerahkan 3.5 kg; **2** kasir membuat `new Order(1, "Ani", 3.5, false)`; **3** kasir menanyakan harganya; **4** pesanan menjawab `24500`, dihitung menurut Persamaan (1.1) dari beratnya *sendiri*.

5 kasir memberi tahu pelanggan; **6** memberi tahu pesanan bahwa pencucian dimulai; **7** memasukkan empat kilogram ke mesin A.

Kasir **tidak menghitung harga**: ia bertanya kepada pesanan. Jika daftar harga berubah tahun depan, hanya `price()` milik pesanan yang berubah.

Nama pesan adalah nama metode, ditulis dengan titik: `ani.price()` mengirim pesan *price* kepada `ani`. Istilah itu berasal dari Smalltalk.

Pemrogram Java dan Python biasanya berkata “memanggil metode pada objek”; maknanya sama. Buku: Subbab 1.3.4.

Abstraksi, dan Empat Gagasan Besar

Abstraksi: simpan hanya yang penting untuk tugasnya — tanpa noda kopi, tanpa lipatan. Terlalu sedikit, model tidak dapat bekerja; terlalu banyak, model mahal dan membingungkan. *Apa* bersifat publik; *bagaimana* urusan objek itu sendiri.

Enkapsulasi: keadaan dibundel bersama, dan disembunyikan di balik, perilaku yang memakainya (Bab 3 dan 7).

Antarmuka (*interface*) mempertajam abstraksi (Bab 6). **Pewarisan:** sebuah `ExpressOrder` adalah sebuah `Order` (Bab 7).

Polimorfisme: pesan yang sama, `price()`, dijawab dengan benar oleh setiap jenis pesanan dengan caranya sendiri (Bab 7).

Seluruh isi buku selanjutnya tumbuh dari beberapa gagasan ini. Buku: Subbab 1.3.5–1.3.6.

Program Buku Catatan, dalam Java

Kode 1.1 `LaundryLedger.java`, bagian 1: empat larik paralel di dalam `main`, dan dua baris ceroboh.

```
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);
    }
```

*Kode 1.1 adalah **pemrograman prosedural**: **larik** (array) adalah deretan kotak bernomor, dan posisi **i** setiap larik menggambarkan pesanan **i**. Buku: Subbab 1.4.1 dan Kode 1.1.*

Dua Fungsi yang Berdiri Sendiri

Kode 1.2 `LaundryLedger.java`, bagian 2: fungsi harga dan fungsi pencetak, yang menerima larik-larik itu dari luar.

```
    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]);
        }
    }
}
```

Data tergeletak terbuka di `main`; operasinya tinggal di tempat lain dan setiap kali menerima empat larik.

Kode 1.2 melengkapi kelas itu; `base / 2` antara dua bilangan bulat sudah merupakan floor dari Persamaan (1.2). Buku: Subbab 1.4.1 dan Kode 1.1.

Buku Catatan yang Sama dalam Python

Kode 1.3 `laundry_ledger.py`, bagian 1: dua fungsi yang sama atas empat *list* Python yang paralel.

```
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])
```

```
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
```

Fungsi `main` pada Kode 1.3 membangun keempat list persis seperti versi Java; kedua program mencetak tujuh baris yang sama. Buku: Subbab 1.4.1 dan Kode 1.2.

Apa yang Salah

Tiga baris pertama benar: $p(3.5) = 24500$, $p_{\text{express}}(5) = 52500$, $p(2) = 14000$.

Lalu Budi berbobot **minus lima kilogram** dan harganya **minus Rp 52.500**; Citra berstatus **PICKED UP** padahal pakaiannya belum pernah dicuci.

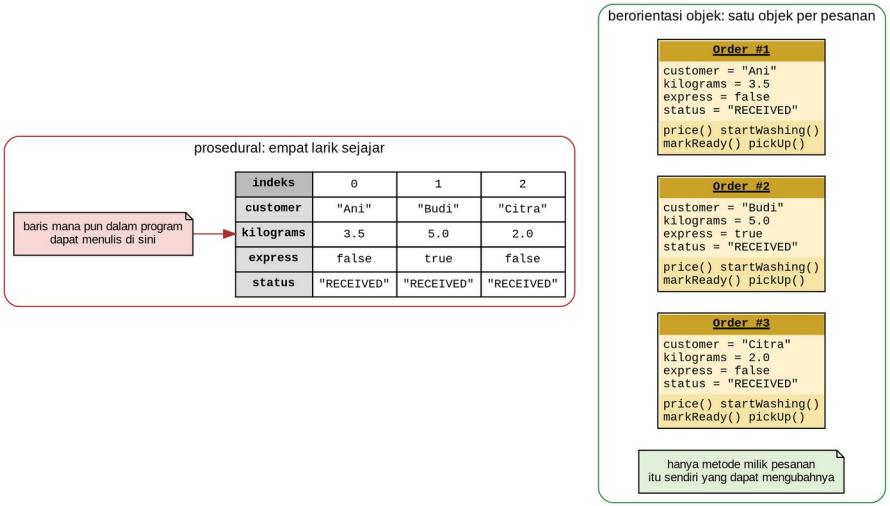
Bila program tumbuh menjadi dua puluh ribu baris yang ditulis lima orang, baris *mana pun* dapat menulis ke `status` atau `kilograms`. Aturan "status hanya bergerak maju" hanya hidup di kepala para pemrogram.

Keempat larik harus selalu *seiring*; menambah nomor telepon berarti larik kelima dan mengubah setiap fungsi.

> **PERINGATAN:** "Hati-hati saja" bukanlah strategi. Cara itu pasti gagal pada setiap program yang dipakai bersama, diubah, atau disimpan lebih dari beberapa minggu.

Buku: Subbab 1.4.2.

Jawaban Objek



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 sisi kanan Gambar 1.4 setiap pesanan adalah kotak tertutup dengan empat tombol, dan `pickUp()` memeriksa status lebih dahulu lalu menolak bila pakaian belum siap. Orientasi objek bukan aritmetika baru; ia adalah cara menata aritmetika yang sama. Buku: Subbab 1.4.3 dan Gambar 1.4.

Mengapa Java

Diciptakan di Sun Microsystems oleh tim James Gosling; dirilis tahun 1995; dikelola Oracle sejak 2010, bersama **OpenJDK** yang terbuka.

Tata bahasanya dipinjam dari C dan C++, tanpa aritmetika *pointer* dan pembebasan memori secara manual; ***garbage collector*** mengambil kembali objek yang tidak terpakai.

Portabel (WORA), **didukung di mana-mana**, **dipakai luas** untuk sistem klien–server, dan **eksplisit** tentang tipe dan akses — tata bahasanya mengeja gagasan-gagasan objek.

Rilis LTS: 8, 11, 17, 21, dan 25. Buku ini menyasar **Java 25**; setiap program juga berjalan di 17 dan 21.

Buku: Subbab 1.5.1–1.5.2.

Dan Mengapa Python Juga

Guido van Rossum, 1991: kode lebih sering dibaca daripada ditulis. Tanpa kurung kurawal — **indentasi** menunjukkan bloknya.

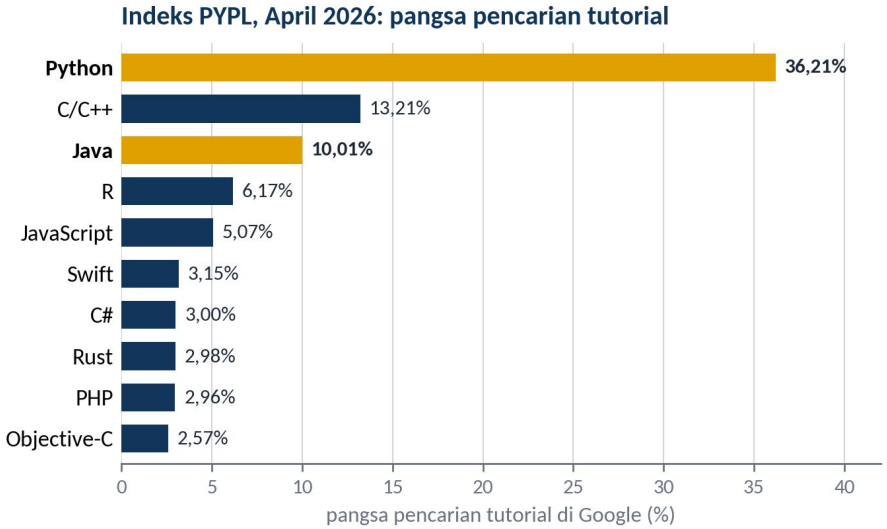
Pengetikan dinamis (diperiksa saat berjalan) lawan **pengetikan statis** Java (diperiksa saat kompilasi). Di Python setiap nilai, bahkan 7, adalah objek.

Bahasa utama analisis data, komputasi ilmiah, otomasi, AI, dan pembelajaran mesin. Buku ini menyasar **Python 3.14**; setiap program juga berjalan di 3.10+.

Java memperlihatkan objek dengan setiap sendinya terlihat; Python memperlihatkan gagasan yang sama tanpa perancah. Melihat keduanya, Anda mempelajari gagasannya, bukan satu ejaannya.

Buku: Subbab 1.5.3.

Kata Peringkat Popularitas: PYPL

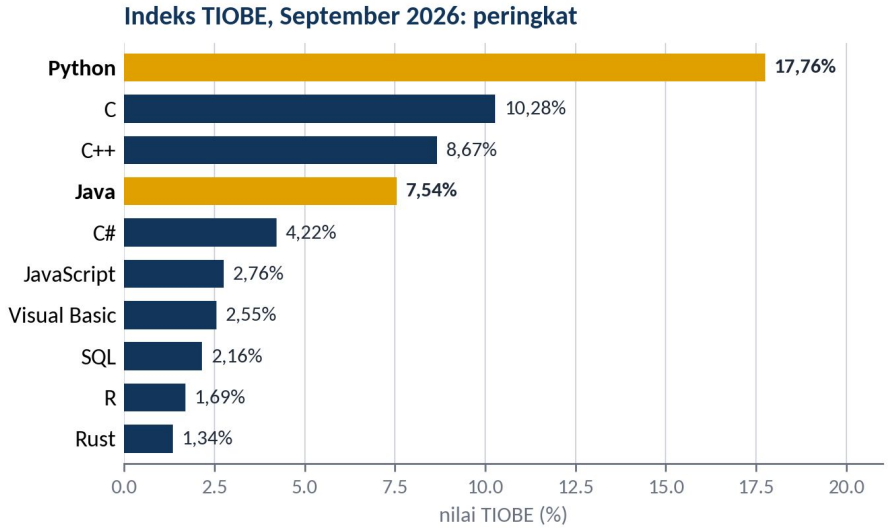


Sumber: PYPL PopularitY of Programming Language index, seluruh dunia, April 2026.

Gambar 1.5 Peringkat PYPL untuk April 2026, berdasarkan seberapa sering tutorial setiap bahasa dicari: Python pertama, Java ketiga.

Gambar 1.5 menghitung pencarian tutorial: Python 36.21 %*, C/C++* 13.21 %*, Java* 10.01 % *(April 2026). Survei Pengembang Stack Overflow 2025 menempatkan Python di urutan keempat (*57.9 % *responden) dan Java di urutan ketujuh (*29.4 %*). Buku: Subbab 1.5.4 dan Gambar 1.5–1.6.*

Kata Peringkat Popularitas: TIOBE



Sumber: TIOBE Index edisi September 2026.

Gambar 1.6 Peringkat TIOBE untuk September 2026, berdasarkan insinyur, kursus, dan vendor per bahasa: Python pertama, Java keempat.

Gambar 1.6 menghitung insinyur, kursus, dan vendor: Python 17.76 %, C 10.28 %, C++ 8.67 %, Java 7.54 % (September 2026). GitHub Octoverse 2025 menghitung kontributor bulanan: TypeScript, Python, JavaScript, lalu Java. Pengukuran-pengukuran itu tidak sepakat soal urutan pertama, tetapi Python selalu di puncak atau dekat puncak dan Java selalu di tujuh besar. Buku: Subbab 1.5.4 dan Gambar 1.7–1.8.

Kedua Bahasa Dibandingkan

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

Baris terpenting Tabel 1.2 justru tidak ada: gagasannya sama. Pelajari gagasannya lebih dulu, lalu ejaan Java-nya, lalu ejaan Python-nya. Buku: Subbab 1.5.5–1.5.6 dan Tabel 1.2.

Perkakas Kerja

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

Sejak Java 11, **JDK** memuat *runtime*-nya sendiri: pasang JDK saja. Buku ini menganjurkan **Eclipse Temurin**. Distribusi Python membawa interpreter, pustaka standar, IDLE, dan **pip**.

*Tabel 1.3 mendaftar perkakas JDK yang dipakai buku ini; **lingkungan virtual** Python (`.venv`) bersifat opsional di sini. Buku: Subbab 1.6.1 dan 1.6.3 serta Tabel 1.3.*

Folder `bin` dan PATH



Gambar 1.7 Folder `bin` sebuah JDK, ditampilkan di terminal Linux; kelima perkakas pada Tabel 1.3 disorot, dan di Windows setiap berkasnya berakhiran `.exe`.

*Pada Gambar 1.7, `javac` mengompilasi dan `java` menjalankan (`javac.exe` dan `java.exe` di Windows); terminal menemukannya hanya bila `bin` ada di **PATH**. "Tidak dikenali" berarti: tidak ada di PATH jendela ini. Buku: Subbab 1.6.2 dan Gambar 1.9–1.11.*

Meja Kerja di Tiga Sistem

1. **JDK:** Temurin 25 dari adoptium.net — `.msi` (atau `winget install EclipseAdoptium.Temurin.25.JDK`), `.pkg` (atau `brew install --cask temurin@25`), atau `sudo apt install temurin-25-jdk`.
2. **Python 3.14:** penginstal python.org (centang **Add python.exe to PATH**), `brew install python@3.14`, atau `sudo apt install python3.14 python3.14-venv`.

3. Periksa di terminal baru, **4. pasang VS Code**, **5. tambahkan kedua ekstensinya**.

6. Jalankan: `javac Hello.java` lalu `java Hello`; `python hello.py` (`python3` di macOS dan Linux).

Pasang perkakas bahasanya dulu, periksa di terminal, baru pasang penyunting dan hubungkan keduanya; ambil pembaruan terbaru dari versi yang disebut. Perintah untuk Fedora dan Arch ada di buku. Buku: Subbab 1.7 dan Tabel 1.5.

Periksa Kedua Perkakas di Terminal Baru

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

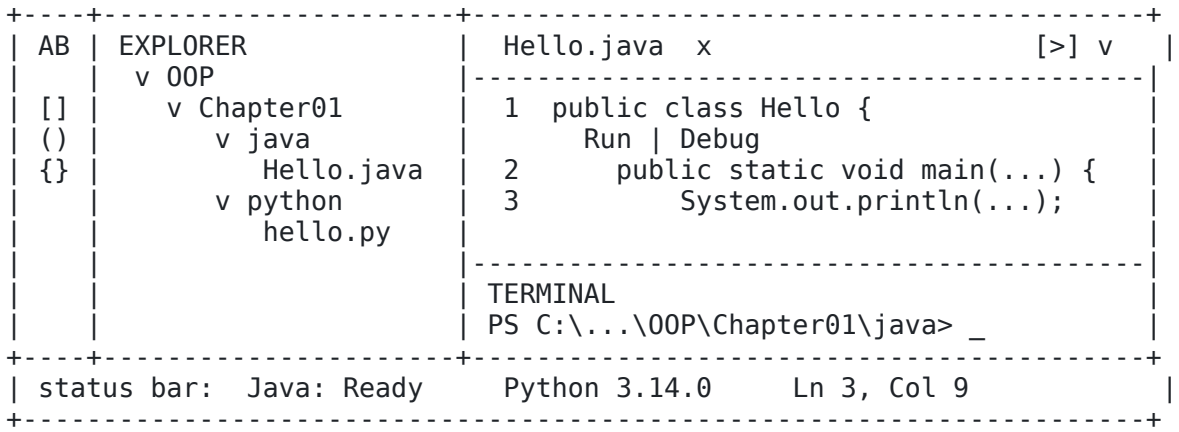
Tutup semua terminal lama lebih dulu: jendela lama menyimpan PATH lama. Di macOS dan Linux ketik `python3` (atau `python3.14`).

Windows: centang **Add python.exe to PATH**; bila `python` membuka Microsoft Store, pakai `py` atau matikan *app execution aliases*.

macOS: Temurin `.pkg` (aarch64 atau x64), penginstal Python *universal2*, lalu *Install Certificates.command*. **Linux:** repositori Adoptium dan `temurin-25-jdk`; Python 3.14 *di samping* Python bawaan sistem.

Buku: Subbab 1.7.1–1.7.3.

VS Code dan Kedua Ekstensinya



Pasang **Extension Pack for Java** dan **Python** (keduanya dari Microsoft); periksa **Java: Configure Java Runtime** dan **Python: Select Interpreter**.

Activity bar, side bar, editor dengan tombol Run, panel dengan Terminal, status bar. Satu folder per bab: `OOP/Chapter01/java`, `OOP/Chapter01/python`. *Buku: Subbab 1.7.4.*

Hello, dalam Kedua Bahasa

Kode 1.4 `Hello.java`: program Java pertama, yang menyapa pembaca dan melaporkan versi Java yang menjalankannya.

```java
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") + ".");
    }
}
```

Kode 1.5 `hello.py`: program Python pertama, yang melakukan hal yang sama dengan Kode 1.4.

```python
import platform

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

Jalankan Kode 1.4 dengan tombol Run, dengan `javac Hello.java` lalu `java Hello`, atau dengan `java Hello.java`; jalankan Kode 1.5 dengan `python hello.py`. Nama berkas harus sama dengan nama kelas, termasuk huruf H kapitalnya. Buku: Subbab 1.7.5–1.7.6 dan Kode 1.3–1.4.

Membaca Hello Baris demi Baris

`public class Hello {` — semua kode Java tinggal di dalam kelas. `public static void main(String[] args)` — titik masuk peluncur: *public* agar dapat dipanggil, *static* agar tidak perlu objek, *void* karena tidak mengembalikan apa pun.

`System.out.println(...)` — `out` adalah objek di dalam kelas `System`; `println` adalah pesan yang dikirim kepadanya dengan titik. `+` menyambung *string*.

Python: `import platform` mendatangkan sebuah **modul**; `print` tidak memerlukan kelas, `main`, atau titik koma; berkasnya dijalankan dari atas.

> **WAWASAN KUNCI:** Program pertama pun sudah penuh objek dan pesan: `System.out` adalah objek, dan `println(...)` adalah pesan yang dikirim kepadanya dengan titik.

Baris keluaran kedua jujur memberi tahu Java atau Python mana yang sebenarnya berjalan. Buku: Subbab 1.7.5–1.7.6.

Berapa Kali Mesin Diisi?

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

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

Satu mesin menampung $c = 8$ kg. $\lceil x \rceil$ adalah bilangan bulat terkecil yang tidak kurang dari x: 9 kg perlu $\lceil 1.125 \rceil = 2$ kali cuci. Pembagian bilangan bulat menghasilkan *floor*, maka Persamaan (1.4) mengubahnya menjadi *ceiling*.

Persamaan (1.3) adalah yang kita inginkan; Persamaan (1.4) adalah cara komputer memperolehnya dari pembagian bilangan bulat. Buku: Subbab 1.7.7 dan Persamaan (1.3)–(1.4).

Mengapa Trik *Ceiling* Itu Benar

Tulis $w = q\,c + r$ dengan $0 \le r \le c - 1$.

Jika $r = 0$: $(w + c - 1)/c = q + (c - 1)/c$, dan $(c - 1)/c < 1$, sehingga *floor*-nya adalah $q = w/c$.

Jika $r \ge 1$: *ceiling*-nya $q + 1$, dan $(w + c - 1)/c = q + (r + c - 1)/c$ dengan $c \le r + c - 1 \le 2c - 2 < 2c$, sehingga *floor*-nya $q + 1$.

> **PERIKSA.** $c = 8$: $w = 9$ memberi $\lfloor 16/8 \rfloor = 2$; $w = 8$ memberi $\lfloor 15/8 \rfloor = 1$; $w = 3$ memberi $\lfloor 10/8 \rfloor = 1$. Kedua kasus sesuai dengan Persamaan (1.3).

Buku: Subbab 1.7.7.

Menghitung Isian Mesin dalam Java

Kode 1.6 `Loads.java`: menghitung muatan mesin untuk pesanan-pesanan hari ini, program kecil dengan perulangan untuk diamati di *debugger*.

```java
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);
    }
}
```

Dengan tangan: 1+2+1+2+1=7, *persis yang dicetak Kode 1.6 di halaman berikutnya. Buku: Subbab 1.7.7 dan Kode 1.5.*

Menghitung Isian Mesin dalam Python

Kode 1.7 `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))
```

```
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
```

Kedua kode mencetak enam baris ini. Pada Kode 1.7, `//` adalah pembagian bilangan bulat dan `str(...)` mengubah bilangan menjadi teks; tidak ada kelas dan tidak ada `main`. Buku: Subbab 1.7.7 dan Kode 1.6.

Mengamati Program Berpikir

1. *Breakpoint*: klik di kiri `totalLoads = totalLoads + loads;` — muncul titik merah. **2. Mulai:** *Debug* di atas `main`, atau `F5`; baris kuning *belum* dijalankan.

3. Panel Variables: `capacity = 8`, `totalLoads = 0`, `i = 0`, `loads = 1`.

4. Melangkah: `F10` menjalankan satu baris; `F5` melanjutkan ke putaran berikutnya. Amati totalnya bertambah: 1, 3, 4, 6, 7.

5. *Watch* `orders[i] / capacity`: untuk `i = 1`, Java menampilkan `1`, Python `1.125` — itulah sebabnya kedua kode membagi dengan cara berbeda. **6. Berhenti:** `Shift+F5`.

`F11` masuk ke dalam pemanggilan metode dan `Shift+F11` keluar darinya; Bab 2 memakai semuanya. Buku: Subbab 1.7.7.

Ketika Ada yang Tidak Beres

`'java' is not recognized` / `command not found` — `bin` tidak ada di PATH terminal ini: buka terminal *baru*, atau pasang ulang dengan **Add to PATH**.

`class Hello is public, should be declared in a file named Hello.java` — samakan nama berkas persis dengan nama kelas.

`Could not find or load main class Hello.class` — ketik `java Hello`, menyebut kelas, bukan berkas.

`IndentationError: unexpected indent` — ratakan blok dengan spasi saja. VS Code menjalankan Python yang salah — **Python: Select Interpreter**.

Baca baris pertama pesannya, cari di Tabel 1.6, terapkan perbaikannya sebelum mencoba hal lain. Buku: Subbab 1.7.8 dan Tabel 1.6.

Sebuah Struk: Variabel dan Aritmetika

Kode 1.8 `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);
    }
}
```

*Pada Kode 1.8 setiap variabel **dideklarasikan** dengan tipenya; `long` menampung hingga sekitar 9.2×10^{18}, sehingga uang tidak pernah meluap.* $50000 - 24500 = 25500$*. Buku: Subbab 1.8.1 dan Kode 1.7.*

Struk yang Sama dalam Python

Kode 1.9 `receipt.py`: struk yang sama dalam Python, tempat nilai-nilai menentukan tipenya sendiri.

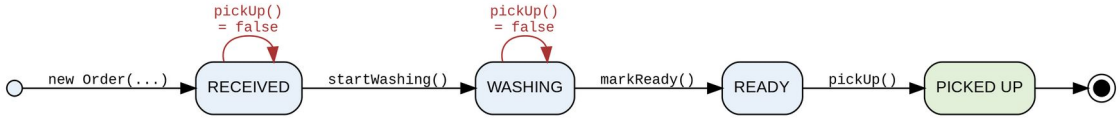
```
customer = "Ani"
kilograms = 3.5
price_per_kg = 7000
price = round(kilograms * price_per_kg)
paid = 50000
change = paid - price

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))
```

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

*Kode 1.9 tanpa kata tipe, bilangan bulat tanpa batas, dan `str(...)` yang eksplisit. Nama Java berbentuk **camelCase** (`pricePerKg`), nama Python **snake_case** (`price_per_kg`). Buku: Subbab 1.8.1 dan Kode 1.8.*

Merancang Kelas `Order`



Gambar 1.8 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.

Mengetahui: nomor, pelanggan, berat, ekspres, status. Melakukan: harga, mulai dicuci, tandai siap, diambil. Tidak boleh: melompati langkah atau mundur. Lingkaran merah pada Gambar 1.8 menyatakan bahwa pesan pada saat yang salah mengembalikan `false` dan tidak mengubah apa pun. Buku: Subbab 1.8.2 dan Gambar 1.12.

`Order.java`: *Field* dan Konstruktor

Kode 1.10 `Order.java`, bagian 1: lima *field* privat dan konstruktornya.

```
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";
    }
```

`private`: kelas lain tidak boleh menyentuh *field* — `ani.status = "PICKED UP";` di luar kelas tidak lolos kompilasi.
`final`: diisi sekali. `this.number` adalah *field*, `number` adalah parameter.

Kode 1.10: konstruktor tidak memiliki tipe kembalian, dijalankan sekali untuk setiap objek baru, dan selalu memulai status dari RECEIVED. Buku: Subbab 1.8.3 dan Kode 1.9.

`Order.java`: Harga dan Langkah Pertama

Kode 1.11 `Order.java`, bagian 2: harga menurut kedua rumus, dan sebuah metode status yang memeriksa sebelum bertindak.

```
    // 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;
    }
```

Pada Kode 1.11, `price()` tidak memerlukan parameter — semuanya ada di dalam objek — dan `base / 2` adalah floor dari Persamaan (1.2). Polanya: periksa, lalu bertindak. Buku: Subbab 1.8.3 dan Kode 1.9.

`Order.java`: Siap dan Diambil

Kode 1.12 `Order.java`, bagian 3: dua metode status terakhir, masing-masing memeriksa status lebih dahulu.

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

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

`!status.equals("READY")` bertanya "apakah statusnya *bukan* READY?" Jika ya, `return false;` langsung mengakhiri metode dan status tidak tersentuh.

Pada Kode 1.12 string dibandingkan dengan `equals`, bukan `==`; halaman Kesalahan yang Sering Terjadi menjelaskan sebabnya. Buku: Subbab 1.8.3 dan Kode 1.9.

`Order.java`: *Getter* dan `toString`

Kode 1.13 `Order.java`, bagian 4: *getter* untuk status, dan deskripsi satu baris.

```
    public String getStatus() {
        return status;
    }

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

`getStatus()` adalah ***getter***: pihak luar boleh *melihat* status, tetapi tidak ada metode yang membiarkan mereka *mengisinya* sesuka hati.

`toString()` dipanggil otomatis ketika pesanan dicetak atau disambung dengan *string*; `@Override` menyatakan bahwa kita sengaja mengganti bawaan yang hambar, `Order@1b6d3586`. `express ? " express" : ""` adalah **operator kondisional**.

Kode 1.13 menutup kelas itu. Kelas ini tidak memiliki `main`: ia formulir kosong, bukan program. Buku: Subbab 1.8.3 dan Kode 1.9.

`order.py`: Konstruktor dan Harga

Kode 1.14 `order.py`, bagian 1: `__init__` menciptakan *field*-nya, dan `price` menghitung dengan rumus yang sama.

```
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
```

`class Order:` dengan titik dua dan indentasi; konstruktornya `__init__`; `self` adalah `this`-nya Java, ditulis sebagai parameter pertama setiap metode.

Pada Kode 1.14 field baru ada ketika pertama kali diisi; `base // 2` langsung merupakan floor dari Persamaan (1.2). Buku: Subbab 1.8.4 dan Kode 1.10.

`order.py`: Metode-Metode Status

Kode 1.15 `order.py`, bagian 2: ketiga metode status.

```
    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
```

*Pada Kode 1.15 privasi hanyalah **konvensi**: garis bawah di depan `_status` berarti "jangan disentuh dari luar", dan perkakas seperti Pylance menandai pelanggarannya. `!=` membandingkan isi string dengan aman. Buku: Subbab 1.8.4 dan Kode 1.10.*

`order.py`: *Getter* dan `__str__`

Kode 1.16 `order.py`, bagian 3: *getter* dan `__str__`, padanan `toString` di Python.

```
    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)
```

Ekspresi kondisional ditulis dengan kata-kata: `" express" if self._express else ""`. Nama metode berbentuk snake_case: `start_washing`, `mark_ready`, `pick_up`, `get_status`.

Dengan Kode 1.16 kelas Python itu lengkap: gagasannya identik dengan Java, ejaannya berbeda di lima tempat. Buku: Subbab 1.8.4 dan Kode 1.10.

Menghidupkan Pesanan-Pesanan

Kode 1.17 `OrderDemo.java`: menciptakan dua pesanan dan mengirimi mereka pesan, termasuk pengambilan yang terlalu dini dan ditolak.

```
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);
    }
}
```

Kompilasi kedua berkas, `javac Order.java OrderDemo.java`, lalu `java OrderDemo`. Pengambilan yang terlalu dini pada Kode 1.17 mengembalikan `false`. Buku: Subbab 1.8.5 dan Kode 1.11.

Program yang Sama dalam Python

Kode 1.18 `order_demo.py`: program yang sama dalam Python, yang mengimpor kelasnya dari `order.py`.

```
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)
```

Tujuh baris keluaran yang sama, kecuali `False` dan `True` berhuruf kapital. `order.py` dan `order_demo.py` harus berada dalam satu folder.

Kode 1.18 mengimpor kelasnya dengan `from order import Order`; tidak ada `new`. Buku: Subbab 1.8.5 dan Kode 1.12.

Menelusuri Demo dengan Diagram Keadaan

```
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
```

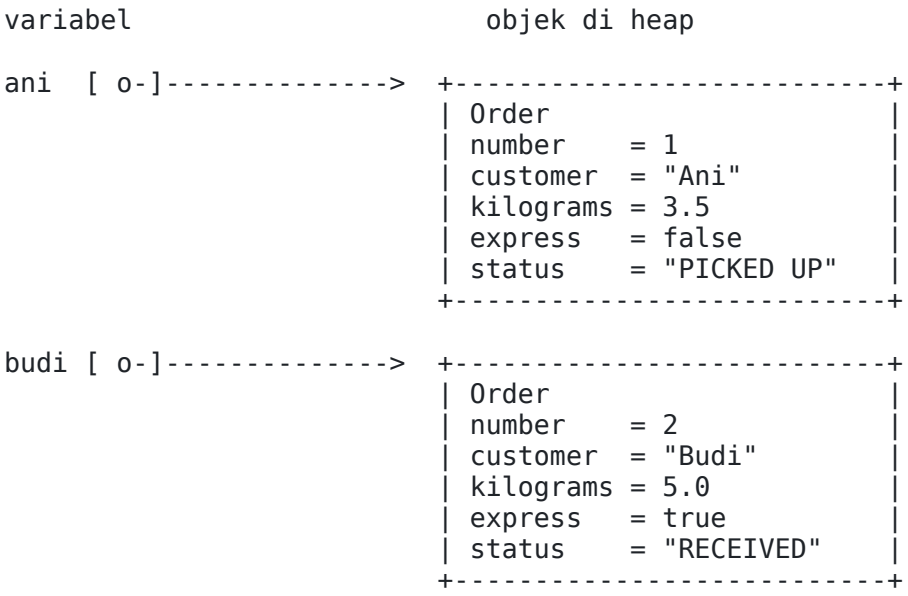
`new Order(...)` menciptakan objek dan menjalankan konstruktor; mencetak berarti memanggil `toString()`. `ani.pickUp()` ketika RECEIVED adalah lingkaran merah pada Gambar 1.8: `false`, status tidak berubah.

Dua panah hitam, lalu `pickUp()` mengembalikan `true`. Budi tetap RECEIVED: semua pesan dikirim ke `ani` — **identitas** sedang bekerja.

> **CONTOH:** Tambahkan `ani.status = "PICKED UP";` ke `OrderDemo.java`. Kompilator menolak: `error: status has private access in Order`. Kesalahan diam-diam pada Kode 1.1 kini mustahil terjadi.

Di atas adalah keluaran Java; Python mencetak `False` dan `True`. Buku: Subbab 1.8.5.

Apa yang Disimpan Variabel Bertipe Kelas



`ani` menyimpan sebuah **referensi** ke objek di ***heap***; `Order x = ani;` menyalin referensinya, bukan pesanannya. Python berperilaku sama.

Referensi adalah saudara pointer C yang lebih aman: tidak ada aritmetika atasnya. Buku: Subbab 1.8.6.

Apa yang Kita Peroleh dari Kelas

Buku catatan memuat data dan rumus yang sama, tetapi aturannya tidak tinggal di mana pun. Kelas menyimpan satu aturannya di tepat satu tempat, dan setiap program yang memakai pesanan mendapatkannya dengan cuma-cuma.

Langkah baru seperti IRONING ditambahkan di satu kelas; semua program ikut.

Belum selesai: konstruktor masih menerima -5 kg (Bab 3 dan 5), dan salah ketik seperti `"REDAY"` di dalam kelas tidak akan ketahuan (bab-bab berikutnya mengganti *string* itu).

Kelas yang baik tumbuh selangkah demi selangkah dengan cermat. Buku: Subbab 1.8.7.

Profitina dalam Praktik

Profitina (`app.profitina.com`) adalah platform **visibilitas AI (GEO)**, ditulis dalam Python dengan FastAPI.

Sinyal adalah objek: setiap hal yang diukur pemindai adalah kelasnya sendiri; pemindai berperan seperti kasir pada Gambar 1.3, mengirim pesan yang sama kepada setiap sinyal. Sinyal baru berarti satu kelas baru.

Model AI berada di balik sebuah objek (Qwen2.5, lokal, melalui Ollama): bila model tidak tersedia, objek itu *mengatakannya*, seperti `pickUp()` yang menolak.

Hukum menjadi bagian rancangan: menurut UU PDP 27/2022, satu kelas yang menjaga data pribadi berarti satu tempat untuk persetujuan, ekspor, dan penghapusan. Aplikasi menyarankan; pengguna memutuskan.

Buku: Subbab 1.10.

Sehari di Profitina Laundry

Kode 1.19 `ShopDay.java`, jalannya hari: empat pesanan, digerakkan sepanjang hari oleh pesan, dengan satu pengambilan terlalu dini yang ditolak.

```
        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();
```

Kode 1.19 memakai kelas `Order` tanpa perubahan; `for (Order o : orders)` mengunjungi setiap pesanan bergantian. Buku: Subbab 1.10.1 dan Kode 1.13.

Uang Kas Hari Itu

Kode 1.20 `shop_day.py`, hitungan sore hari: uang dari pesanan yang sudah diambil, dan pesanan yang masih menunggu.

```
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))
```

```
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
```

Kode 1.20 adalah ejaan Python dari hitungan yang sama. Dewi: $52500+26250=78750$ *menurut Persamaan (1.2); kas* $24500+14000=38500$*; dua pesanan masih di kios.* `ShopDay` *tidak memerlukan satu baris pun untuk menolak Dewi — kelasnya yang melakukannya. Buku: Subbab 1.10.1 dan Kode 1.13–1.14.*

Kesalahan yang Sering Terjadi

Menjalankan versi lama — kompilasi ulang setelah setiap penyuntingan, atau pakai Run / `java Hello.java`. **Nama berkas dan nama kelas berbeda**, bahkan hanya pada huruf kapital.

Huruf besar-kecil dan tanda baca: `system.out`, `Print("hi")`, `;` yang hilang, indentasi nyasar. Baca baris yang disebut *dan baris sebelumnya*.

`==` **pada *string* di Java** membandingkan referensi; pakai `equals`. ***Field* publik "demi menghemat waktu"** mengundang bencana buku catatan.

Kelas lawan objek: kelas tidak memiliki status; setiap objek memiliki statusnya sendiri. **Format angka regional:** `String.format("%.1f", 3.5)` dapat mencetak `3,5`.

Memercayai program karena ia berjalan: buku catatan berjalan mulus dan mencetak harga negatif. Periksa keluaran dengan tangan terhadap rumusnya.

Buku: Subbab 1.11.

Ringkasan — Lima Hal untuk Diingat

1. Program adalah algoritma dalam bahasa pemrograman. Java mengompilasi ke *bytecode* untuk JVM (`javac`, lalu `java`); Python mengompilasi dan menjalankan di dalam interpreter.
2. Objek memiliki keadaan, perilaku, dan identitas; kelas adalah formulir kosong; objek bekerja sama melalui pesan yang ditulis dengan titik.
3. Larik paralel tidak menugaskan siapa pun menjaga data; objek membundel datanya dengan satu-satunya metode yang boleh mengubahnya, dan metode itu dapat **menolak**.
4. Meja kerja: JDK 25 (Temurin), Python 3.14, VS Code dengan dua ekstensi; *breakpoint*, `F5`, `F10`, `F11`, Variables.
5. `Order`: *field* privat, konstruktor, `price()` menurut Persamaan (1.1) dan Persamaan (1.2), dan metode status yang mengikuti Gambar 1.8. Variabel menyimpan referensi.

Buku: Subbab 1.12.

Pertemuan Berikutnya

Bab 2 — Lingkungan Pengembangan: Menyunting, Menjalankan, dan Men-debug.

Apa yang sebenarnya terjadi antara *Run* dan baris keluaran pertama: `javac`, *bytecode*, `javap`, JVM; kompilator tersembunyi Python dan `dis`.

Eclipse dan VS Code berdampingan; proyek, paket, dan impor; serta *debugging* dari nol pada tagihan *laundry* yang Rp 16.200 terlalu mahal.

Sebelum kita bertemu: Soal Latihan 1.3, 1.4, 1.7, dan 1.9 — harga dengan tangan, isian mesin dengan tangan, sepasang pesan yang tertukar, dan `WashingMachine` buatan Anda sendiri. Penyelesaiannya ada di Subbab 1.14.

Buku: Subbab 1.13–1.14, lalu Bab 2.

Pemrograman Berorientasi Objek (EF234302)

Kuliah 2 — Lingkungan Pengembangan: Menyunting, Menjalankan, dan Men-debug

Buku: Irfan Subakti, *Pemrograman Berorientasi Objek dengan Java dan Python*, Bab 2

Dosen: Irfan Subakti — `yifana@gmail.com`

Departemen Teknik Informatika, Institut Teknologi Sepuluh Nopember, Surabaya

Hari ini kita membuka meja kerja itu sendiri, lalu memakainya untuk menemukan bug sungguhan langkah demi langkah, bukan dengan menebak. Buku: Bab 2.

Agenda

1. Apa itu IDE, dan apa yang sebenarnya terjadi ketika Anda menekan *Run*: `javac`, *bytecode*, JVM; kompilator tersembunyi Python.
2. Eclipse dan VS Code berdampingan: *workspace*, *view*, editor, *perspective*, proyek; folder, interpreter, `launch.json`.
3. Proyek, paket, modul, dan *import* — serta *import* yang terlupa di kedua bahasa.
4. *Debugging* dari nol: *breakpoint*, melangkah, tumpukan pemanggilan, variabel, *watch*, *breakpoint* bersyarat.
5. Tiga perburuan terbimbing: tagihan Rp 16.200 terlalu mahal, hasil sehari kurang Rp 6.000, dan program yang mogok.

Buku: Bab 2, Subbab 2.1–2.18.

Di Mana Kita Berhenti

Bab 1 menaruh meja kerja di hadapan Anda — JDK 25, Python 3.14, VS Code — menjalankan program pertama, mengamati sebuah gelung di *debugger*, dan membangun kelas `Order`.

Kini kita memperlambat langkah dan menengok meja kerja itu sendiri, karena Anda akan menghabiskan ratusan jam di sana.

Tiga pertanyaan: **Apa yang terjadi ketika program dijalankan? Bagaimana memakai perkakas dengan baik? Ketika program salah, bagaimana mengetahui sebabnya?**

> **WAWASAN KUNCI:** Anda tidak perlu mengingat setiap jalur menu. Simpanlah *kosakatanya* — *workspace*, proyek, paket, *perspective*, *breakpoint*, *step over*, *step into*, tumpukan pemanggilan — dan *kebiasaan* meraih *debugger* alih-alih menebak.

Buku: Subbab 2.1.

Apa Itu IDE

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.

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

Kata kuncinya terpadu: editor mengenal kompilator, debugger mengenal keduanya. Tabel 2.1 menamai kelima tugas; setiap menu termasuk salah satunya. Buku: Subbab 2.2.1–2.2.2 dan Tabel 2.1.

Dua Meja Kerja, Satu Perangkat Gagasan

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+.)

Simpan Tabel 2.2 di dekat Anda. Buku: Subbab 2.2.3 dan Tabel 2.2.

Kompilator, Interpreter, dan Kapan Masing-Masing Mengeluh

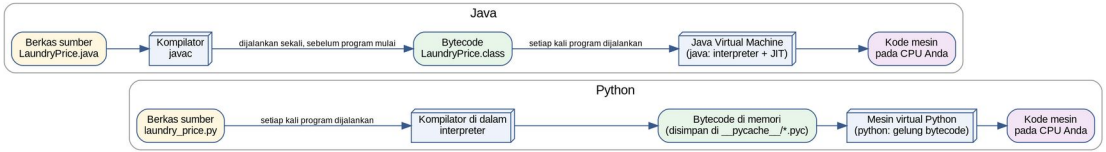
CPU hanya memahami **kode mesin**, dan kode x86-64 tidak berjalan di ARM64. Sesuatu harus menerjemahkan `total = total + line`.

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

Penerjemah buku menemukan kalimat yang tak masuk akal sebelum ada yang membacanya; penerjemah lisan baru menemukannya ketika pembaca sampai di sana.

*Java maupun Pytʜon lebih dahulu mengompilasi kode sumber menjadi **bytecode**, lalu sebuah **mesin virtual** menginterpretasikannya. Buku: Subbab 2.3.1–2.3.2.*

Dari Kode Sumber ke Prosesor



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.

Gambar 2.1: Java mengompilasi sekali dan JVM, dengan kompilator just-in-time untuk metode yang "panas", menjalankan bytecode setiap kali; Python mengompilasi di dalam interpreter setiap kali program dimulai. Buku: Subbab 2.3.2 dan Gambar 2.1.

Java: `javac`, Lalu `java`

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));
    }
}
```

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

```
2 + 3 = 5
```

Perintah pertama menulis `Add.class` dan tidak mencetak apa pun bila semuanya beres. Sejak Java 11, `java Add.java` mengompilasi Kode 2.1 di memori lalu menjalankannya, tanpa meninggalkan berkas `.class`. Buku: Subbab 2.3.3 dan Kode 2.1.

Membaca *Bytecode* dengan `javap -c`

```
  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
```

`iload_0`, `iload_1` mendorong `a` dan `b` ke sebuah tumpukan kecil; `iadd` mengambil keduanya dan mendorong jumlahnya; `ireturn` mengembalikannya. Itulah `return a + b;` bagi mesin bertumpukan.

JVM menjalankan berkas `.class`, bukan berkas `.java` Anda: menyunting tanpa mengompilasi ulang berarti menjalankan program lama. Daftar lengkapnya juga memuat konstruktor bawaan `Add()` yang tidak pernah Anda tulis. Buku: Subbab 2.3.3.

Python: Kompilator yang Tersembunyi

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

```python
"""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)
```

```
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
```

Kode 2.2 pada Python 3.11: `LOAD_FAST`, `LOAD_FAST`, `BINARY_OP (+)`, `RETURN_VALUE`. JVM memiliki satu instruksi tambah per tipe (`iadd`, `dadd`); `BINARY_OP` Python memutuskan saat berjalan apa arti "tambah". Modul yang diimpor disimpan di `__pycache__`. Buku: Subbab 2.3.4 dan Kode 2.2.

Kapan Kesalahan Tertangkap

`;` yang hilang di Java dan `:` yang hilang di Python sama-sama tertangkap **sebelum** apa pun berjalan — berkas, baris, tanda sisip, jenisnya:

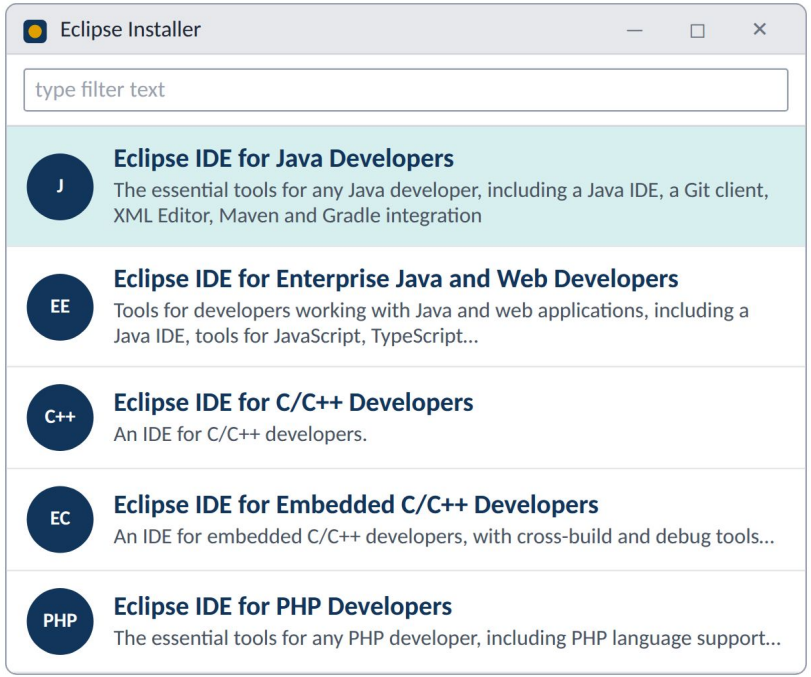
```
src/myfirstproject/MyFirstClass.java:6: error: ';' expected
        System.out.println("Hello Eclipse!")
                                            ^
1 error
```

Nama yang tidak ada lain ceritanya: kompilator Java memeriksa setiap nama dan tipe; Python baru mencari nama ketika barisnya dijalankan.

> **WAWASAN KUNCI:** *Java memeriksa nama dan tipe sebelum program berjalan; Python memeriksanya selagi program berjalan.* Mengetahui Anda berada di pihak mana memberi tahu Anda di mana mencari kesalahan.

Salah ketik nama di cabang Python yang jarang dijalankan dapat bersembunyi berbulan-bulan — salah satu alasan pengujian (Bab 6) lebih penting lagi di Python. Buku: Subbab 2.3.5.

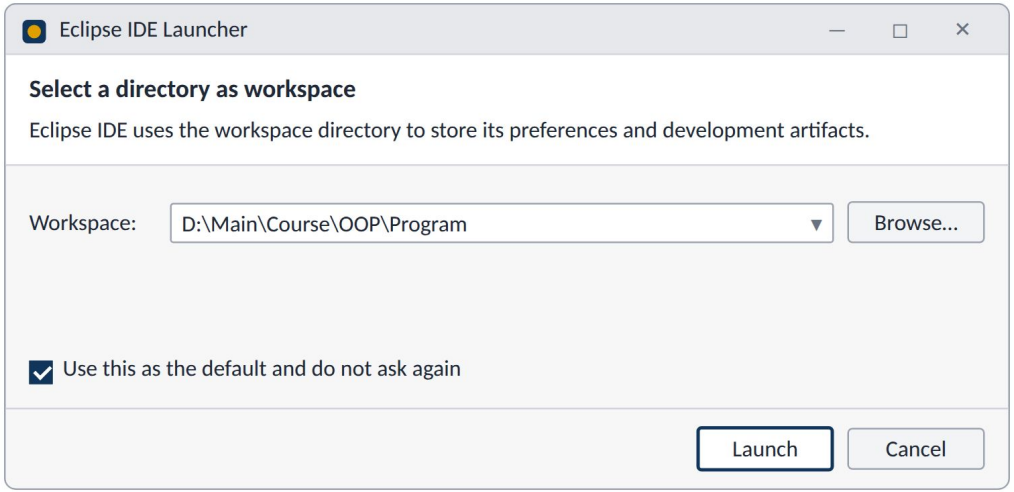
Memasang Eclipse



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

*Dari `eclipse.org/downloads`, Eclipse Installer: pilih **Eclipse IDE for Java Developers** (Gambar 2.2), biarkan runtime Java yang ditawarkan, pilih folder, **INSTALL**, lalu **LAUNCH**. Java yang menjalankan Eclipse dan JDK yang dipakai proyek adalah dua pengaturan terpisah. Buku: Subbab 2.4.1 dan Gambar 2.2–2.3.*

Workspace



Gambar 2.3 Eclipse IDE Launcher meminta sebuah folder workspace sebelum meja kerja terbuka.

Workspace Eclipse (Gambar 2.3) adalah folder berisi metadata Eclipse (di `.metadata`) dan, secara bawaan, proyek-proyek Anda. Satu workspace untuk setiap kursus menjaga pengaturan tetap terpisah; File > Switch Workspace menggantinya. Buku: Subbab 2.4.2 dan Gambar 2.4–2.5.

View, Editor, dan *Perspective*

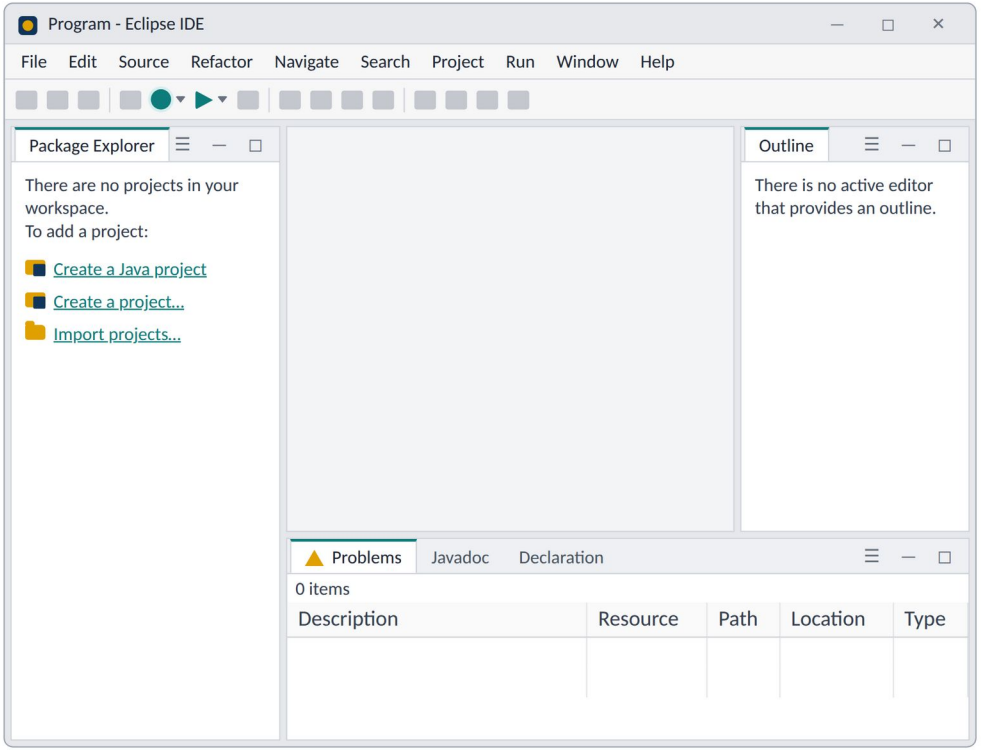
DEFINISI. *View* adalah panel yang menampilkan informasi terstruktur dan memungkinkan Anda bertindak langsung terhadapnya. *Editor* menampilkan isi satu berkas untuk Anda ubah; perubahannya tidak ditulis ke disk sampai Anda menyimpannya. *Perspective* adalah susunan *view* dan editor bernama yang cocok untuk satu jenis tugas.

Mengganti nama berkas di *view* langsung mengubahnya di disk; mengetik di editor tidak mengubah apa pun sampai **Ctrl+S**. Editor yang ***dirty*** menampilkan `*MyFirstClass.java` — dan *Run* menjalankan versi terakhir yang *tersimpan*.

Dua *perspective* mengerjakan hampir semuanya: **Java** dan **Debug**. *Window > Perspective > Reset Perspective…* menyelamatkan jendela yang berantakan.

Buku: Subbab 2.5.1–2.5.2 dan Gambar 2.7–2.8.

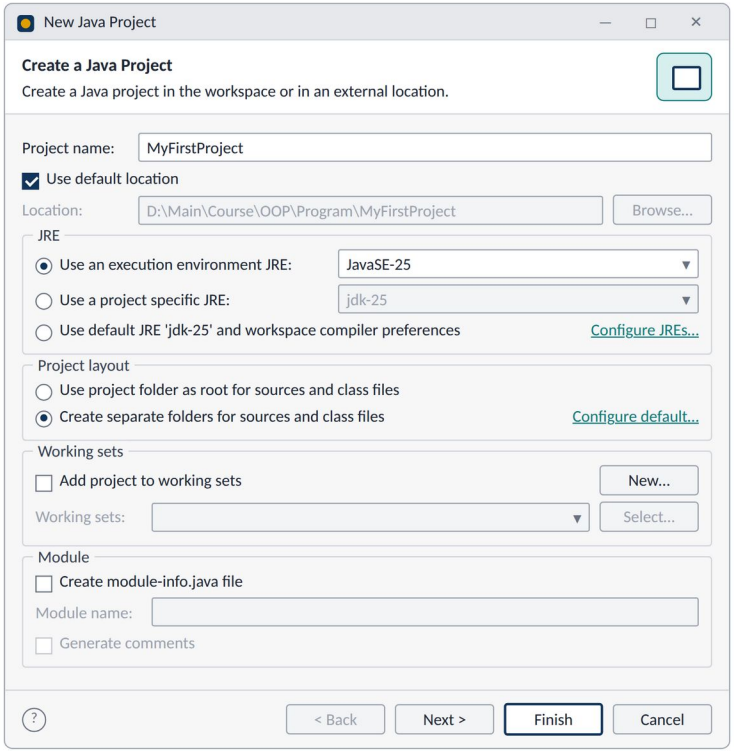
Perspective Java



Gambar 2.4 Perspective Java pada workspace kosong: Package Explorer di kiri, area editor di tengah, Outline di kanan, serta Problems, Javadoc, dan Declaration di bawah.

Pada Gambar 2.4: Package Explorer, area editor, Outline; Problems, Javadoc, dan Declaration di bawah. Nyalakan Show Line Numbers sekarang. Buku: Subbab 2.5.3–2.5.4 dan Gambar 2.9.

Sebuah Proyek, Sebuah Paket, Sebuah Kelas



Gambar 2.5 Wizard New Java Project, dengan folder terpisah untuk berkas sumber dan berkas kelas.

File > New > Java Project, beri nama `MyFirstProject`, pertahankan folder `src` dan `bin` yang terpisah (Gambar 2.5); lalu New > Package `myfirstproject` (huruf kecil), lalu New > Class `MyFirstClass` dengan kerangka `main`. Buku: Subbab 2.6.1–2.6.3 dan Gambar 2.14–2.17.

Program Pertama Proyek Itu

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

```java
package myfirstproject;

public class MyFirstClass {

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

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

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


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


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

Baris `package` pada Kode 2.3 harus sesuai dengan foldernya; `if __name__ == "__main__":` pada Kode 2.4 menjalankan `main()` hanya bila berkasnya dijalankan, bukan diimpor. Buku: Subbab 2.6.3 dan Kode 2.3–2.4.

Berjalan di Luar Eclipse: JAR

JAR (*Java ARchive*) adalah berkas ZIP berisi berkas `.class` ditambah sebuah *manifest*. *Export… > Java > JAR file*, pilih `myprogram1.jar`, **Finish**.

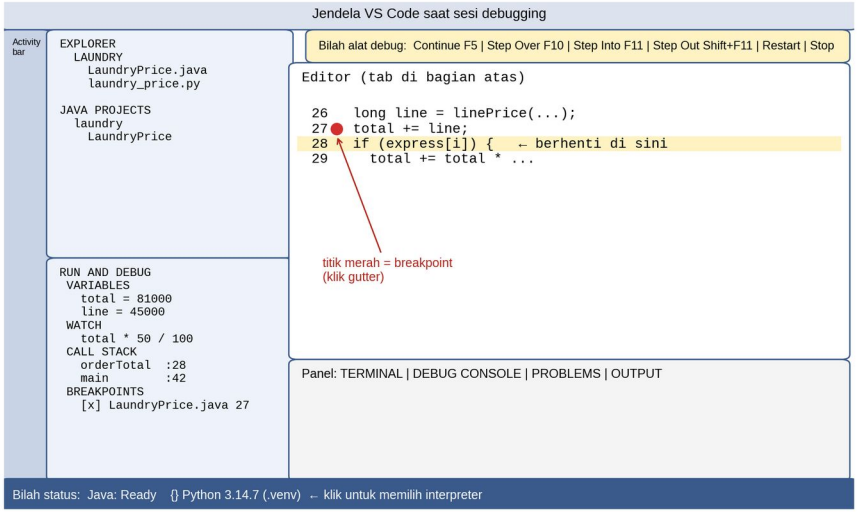
```
java -classpath myprogram1.jar myfirstproject.MyFirstClass
```

```
Hello Eclipse!
```

Classpath adalah tempat JVM mencari kelas; kelasnya disebut *beserta paketnya*. Paket itu telah menjadi folder di dalam JAR.

`jar tf myprogram1.jar` *mendaftar isinya; Runnable JAR berjalan dengan* `java -jar myprogram1.jar`*. Buku: Subbab 2.6.4 dan Gambar 2.20–2.22.*

Pekerjaan yang Sama di VS Code



Gambar 2.6 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.

*Gambar 2.6 adalah peta untuk sisa pertemuan ini. VS Code membuka **folder**, bukan workspace; view **Java Projects** berperan sebagai Package Explorer; CodeLens **Run** dan **Debug** ada di atas `main`. Buku: Subbab 2.7–2.7.2 dan Gambar 2.23.*

Python yang Mana? Dan Cara Menjalankan yang Tersimpan

Satu komputer sering memiliki beberapa Python. ***Status bar*** menunjukkan interpreter terpilih; *Python: Select Interpreter* menggantinya; *Python: Create Environment* membuat `.venv`.

Konfigurasi peluncuran di `.vscode/launch.json` menyimpan argumen, folder kerja, atau modul yang dijalankan dengan `python -m`:

```
{
    "type": "java",
    "name": "Tester (Java)",
    "request": "launch",
    "mainClass": "main.Tester"
},
```

F5 memulai konfigurasi terpilih di bawah debugger, Ctrl+F5 tanpanya; `debugpy` adalah tipe debugger Python. Panel Terminal selalu menampilkan perintah persis yang dijalankan. Buku: Subbab 2.7.3–2.7.5.

Paket Adalah Folder

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

```
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
```

Nama lengkap mencegah tabrakan: `util.Counter` *dan* `org.w3c.dom.css.Counter` *milik pustaka adalah dua kelas berbeda. Buku: Subbab 2.8.1–2.8.3.*

`Counter`: Paket, *Field*, *Getter*

Kode 2.5 `Counter.java`, bagian 1: kelas di dalam paket `util`, dengan satu *field* dan *getter*-nya.

```
package util;

public class Counter {

    private int result = 0;

    public int getResult() {
        return result;
    }
```

`package util;` harus sesuai dengan folder `util` di bawah `src`. *Field* `result` milik objek dan bertahan di antara pemanggilan — Soal Latihan 2.7 menunjukkan mengapa hal itu penting.

Versi templat `count` hanya berisi komentar `TODO` dan `return 0;`; Eclipse mengumpulkan setiap `TODO` di *view Tasks*.

Kode 2.5 membuka kelas yang akan diperiksa `Tester`. Buku: Subbab 2.8.3, Kode 2.5, dan Gambar 2.24.

`Counter.count`, dan Apa yang Harus Dikembalikannya

Kode 2.6 `Counter.java`, bagian 2: `count` menjumlahkan $1+2+\cdots+x$ dan melempar eksepsi di luar $1 \le x \le 255$.

```
    // 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;
    }
}
```

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

DENGAN TANGAN. $S(5) = 1+2+3+4+5 = 15 = \frac{5 \cdot 6}{2}$. Persamaan (2.1) memberi nilai yang diperiksa `Tester`.

Kode 2.6 menolak masukan yang buruk dengan melempar eksepsi (Bab 5). Buku: Subbab 2.8.3, Kode 2.5, dan Persamaan (2.1).

`Tester`: Kelas di Paket `main`

Kode 2.7 `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");
        }
    }
}
```

`import util.Counter;` *memungkinkan nama pendek di Kode 2.7. Buku: Subbab 2.8.3 dan Kode 2.6.*

Kembarannya dalam Python

Kode 2.8 `util/counter.py`, metode `count`: penolakannya berupa `ValueError`.

```
    # 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 2.9 `main/tester.py`, baris-baris pertamanya: *import* menyebut modulnya, lalu benda di dalamnya.

```
"""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)
```

Kode 2.8 dan Kode 2.9: jalankan dengan `cd python` *dan* `python3 -m main.tester`*, dari folder di atas paket;* `python3 main/tester.py` *gagal dengan* `ModuleNotFoundError`*. Buku: Subbab 2.8.3 dan Kode 2.7–2.8.*

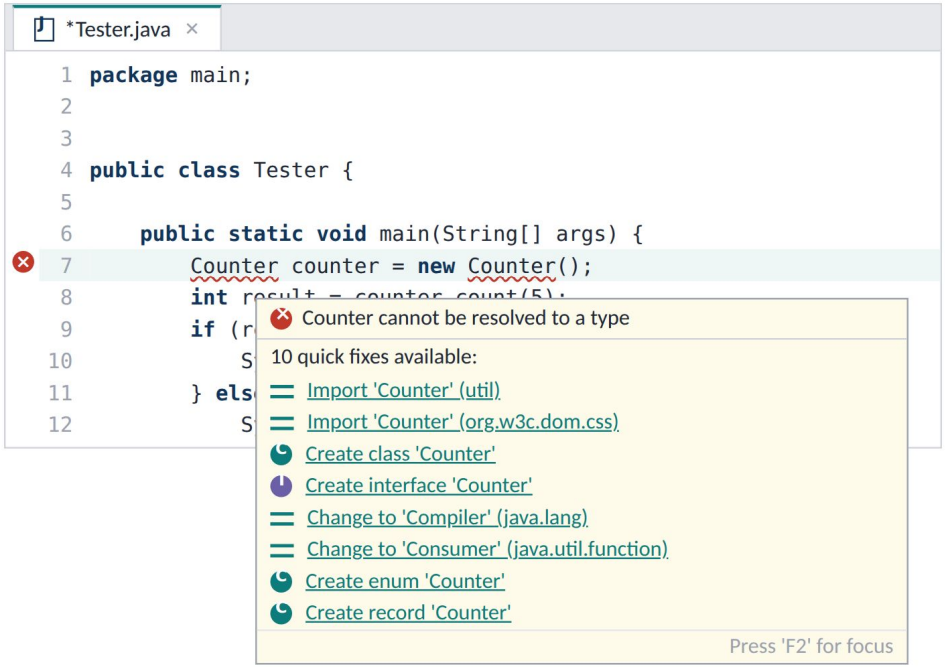
Import yang Terlupa, di Kedua Bahasa

```
src/main/Tester.java:7: error: cannot find symbol
        Counter counter = new Counter();
        ^
  symbol:   class Counter
  location: class Tester
  File "/home/student/counter/py/main/tester.py", line 5, in main
    counter = Counter()
              ^^^^^^^
NameError: name 'Counter' is not defined. Did you mean: 'counter'?
```

Java: `cannot find symbol` pada *waktu kompilasi* — tidak ada yang berjalan. **Python:** `NameError` pada *waktu jalan*, baru ketika baris 5 dicapai.

Pesan Java muncul dua kali, sekali untuk tipe dan sekali untuk konstruktor. Bingkai `<frozen runpy>` pada traceback Python yang lengkap milik `python -m`. Buku: Subbab 2.8.4.

IDE Menawarkan Perbaikannya



Gambar 2.7 Counter cannot be resolved to a type: kotak quick fix menawarkan import util.Counter, atau org.w3c.dom.css.Counter yang tidak berkaitan.

Pada Gambar 2.7 quick fix pertama menambahkan `import util.Counter;`; yang kedua menawarkan `Counter` pustaka yang tidak berhubungan — tabrakan nama yang menjadi alasan paket ada. Source > Organize Imports (Ctrl+Shift+O) menambah dan mengurutkan semua import sekaligus. Buku: Subbab 2.8.4 dan Gambar 2.25–2.26.

Mengompilasi Paket secara Manual

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

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

`-d bin` menulis satu subfolder per paket; `-cp bin` menaruh `bin` di *classpath*; kelasnya disebut dengan titik, `main.Tester` — bukan garis miring, bukan `.class`.

Dua baris ini persis yang dilakukan Eclipse dan VS Code ketika Anda menekan Run. Untuk berbagi proyek utuh: Export > Archive File dan Import > Existing Projects; di VS Code proyek hanyalah folder. Buku: Subbab 2.8.5–2.8.6.

Perkakas Andalan Penyuntingan

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: Eclipse: F11 memulai *debugging*, F5 melakukan *step into*. VS Code: sebaliknya.

Content assist, Quick Fix, Rename, Format: Tabel 2.3. Buku: Subbab 2.9 dan Tabel 2.3.

Debugging dari Nol

Bug adalah selisih antara apa yang dilakukan program dan apa yang seharusnya. Bug yang lantang berhenti dengan pesan; bug yang diam mencetak angka salah yang meyakinkan — Rp 109.350 tampak senyata Rp 93.150.

Debugging* dengan cetak** berjalan di mana saja, tetapi setiap pertanyaan berarti sunting dan jalankan ulang. ***Debugger berhenti di baris mana pun dan menampilkan setiap variabel tanpa menyunting.

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

Buku: Subbab 2.10.1.

Tujuh Kata untuk *Debugger* Apa Pun

***Breakpoint*:** berhenti *sebelum* menjalankan baris ini. ***Suspended*:** berhenti, barisnya disorot. **Melangkah:** *step over*, *into*, *out*. ***Resume*:** jalan sampai *breakpoint* berikutnya.

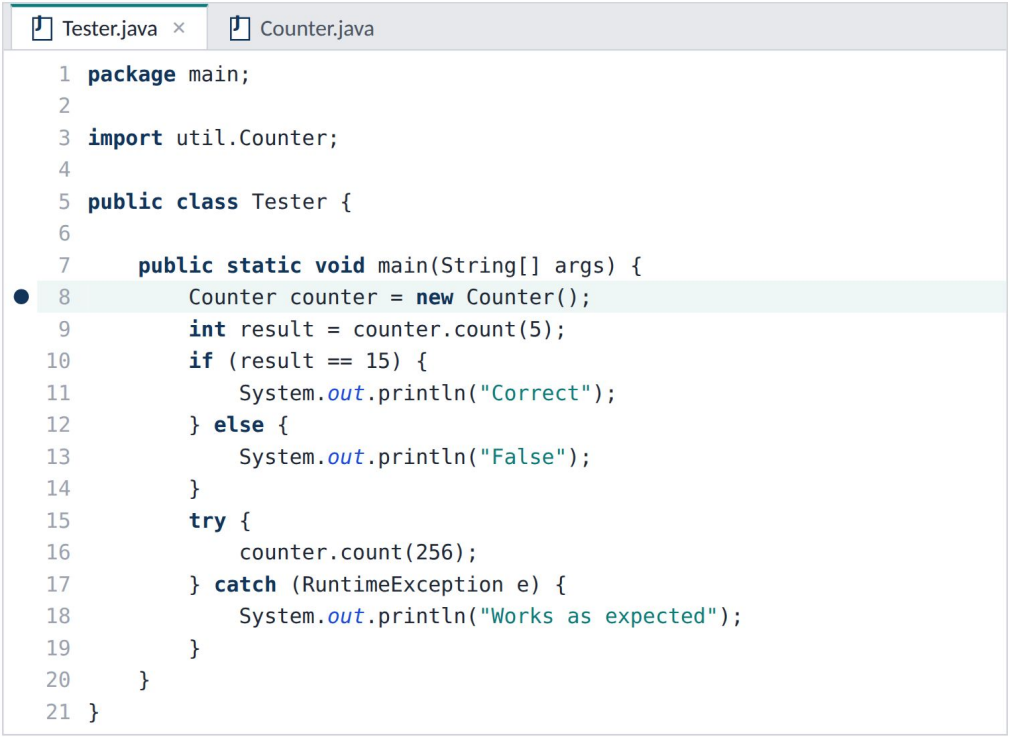
Variabel metode yang sedang berjalan; ***watch*** seperti `total * 50 / 100`, dievaluasi ulang di setiap perhentian.

Tumpukan pemanggilan: metode-metode yang sedang berjalan, masing-masing dipanggil oleh yang di bawahnya.

Watchpoint berhenti setiap kali sebuah *field* tertentu dibaca atau diubah, oleh baris mana pun.

Buku: Subbab 2.10.2.

Breakpoint di Eclipse



Gambar 2.8 Sebuah breakpoint pada baris 8, Counter counter = new Counter(); titik di ruler menandainya.

Klik ganda di ruler, Toggle Breakpoint, atau Ctrl+Shift+B (Gambar 2.8); lalu Debug As > Java Application. Di bawah Run As breakpoint diabaikan. Eclipse menawarkan pindah ke perspective Debug. Buku: Subbab 2.10.3 dan Gambar 2.30–2.34.

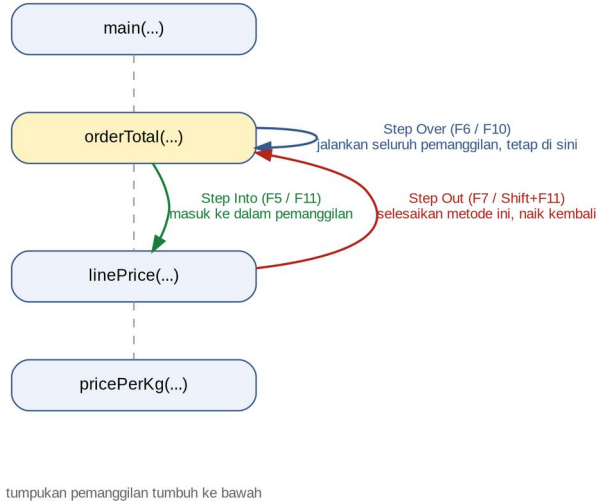
Resume, Step Over, Step Into, Step Out

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`

Tabel 2.4: over yang Anda percayai, into yang Anda curigai. Buku: Subbab 2.10.4 dan Tabel 2.4.

Melangkah, Dilihat pada Tumpukan Pemanggilan

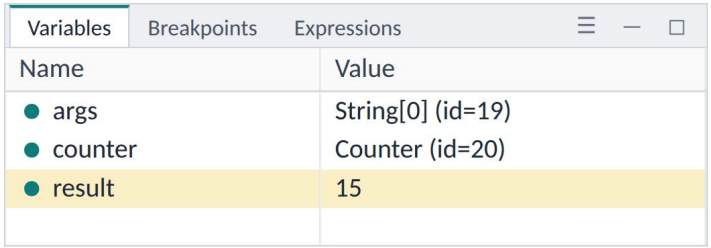


Gambar 2.9 Langkah-langkah debugger dilihat pada tumpukan pemanggilan. Step Into turun ke method yang dipanggil, Step Out kembali ke pemanggil, dan Step Over menjalankan seluruh pemanggilan dan tetap pada tingkat yang sama.

Pada Gambar 2.9, dari baris `orderTotal` yang memanggil `linePrice`, yang memanggil `pricePerKg`: Step Over tetap di tingkat yang sama, Step Into turun, Step Out naik kembali ke pemanggil. Buku: Subbab 2.10.4 dan Gambar 2.36.

Tumpukan Pemanggilan dan Variabel

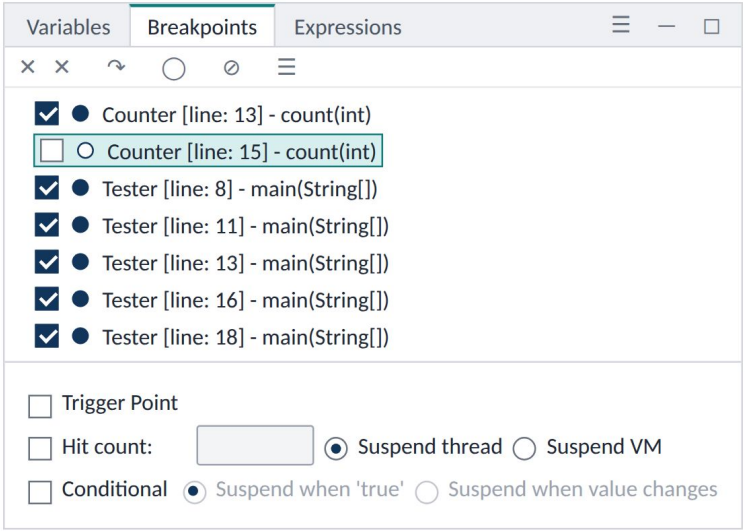
DEFINISI. *Tumpukan pemanggilan* (*call stack*) adalah daftar pemanggilan metode yang sudah dimulai tetapi belum selesai. Metode yang sedang dieksekusi berada paling atas; di bawahnya metode yang memanggilnya, lalu yang memanggil metode itu, terus sampai `main`.



Gambar 2.10 View Variables sesudah count(5) mengembalikan nilainya: args, counter, dan result = 15.

Klik bingkai yang lebih bawah untuk melihat baris dan variabel miliknya. View Variables (Gambar 2.10) menyorot apa yang diubah langkah terakhir; Anda bahkan dapat mengubah nilai untuk bertanya "bagaimana jika?". Buku: Subbab 2.10.5–2.10.6 dan Gambar 2.37–2.38.

Mengelola *Breakpoint*



Gambar 2.11 View Breakpoints mendaftar setiap breakpoint; kotak centangnya menyalakan atau mematikan breakpoint, dan bagian bawahnya memuat properti Hit count dan Conditional.

View Breakpoints (Gambar 2.11) menonaktifkan tanpa menghapus dan menyimpan dua sifat ampuh: ***Hit count*** *(berhenti pada kedatangan ke-*`n`*) dan* ***Conditional*** *(berhenti bila* `i == 7` *benar). Skip All Breakpoints (Ctrl+Alt+B) membungkam semuanya. Buku: Subbab 2.10.7 dan Gambar 2.42–2.43.*

Debugger yang Sama di VS Code

Klik *gutter* (atau F9) untuk titik merah; CodeLens **Debug** atau F5 untuk memulai. *Side bar* Run and Debug menampilkan **Variables**, **Watch**, **Call Stack**, **Breakpoints**.

Toolbar: Continue (F5), Step Over (F10), Step Into (F11), Step Out (Shift+F11), Restart, Stop. **Debug Console** mengevaluasi ekspresi apa pun.

Add Conditional Breakpoint…: Expression, Hit Count, atau **Log Message** — *logpoint* mencetak tanpa berhenti.

> **WAWASAN KUNCI:** Setiap *debugger* grafis menawarkan seperangkat kecil kendali yang sama. Pelajari gagasannya sekali, pada satu alat, dan Anda dapat memakai *debugger* mana pun dalam hitungan menit.

Buku: Subbab 2.10.8.

Perburuan Terbimbing: Tagihan yang Terlalu Mahal

wash Rp 6.000/kg, *iron* Rp 4.000/kg, *wash+iron* Rp 9.000/kg; di bawah 3 kg dihitung 3 kg; *express* +50% pada item itu; anggota diskon 10% untuk tagihan minimal Rp 100.000.

$$p_i = \left(1 + \frac{e_i}{2}\right) \cdot r(s_i) \cdot \max(w_i, 3), \tag{2.2}$$

$$T = \begin{cases} \frac{9}{10} P & \text{jika pelanggan adalah anggota dan } P \geq 100\,000, \\ P & \text{selain itu,} \end{cases} \quad \text{dengan } \sum_{i=1}^{n} p_i. \tag{2.3}$$

Bu Sari, seorang anggota: 4 kg *wash*, 2 kg *iron*, 5 kg *wash+iron* ekspres. Program berkata **Rp 109.350**; struknya berkata **Rp 93.150**.

Persamaan (2.2) menghargai satu item dan Persamaan (2.3) seluruh tagihan. Buku: Subbab 2.11.1 dan Persamaan (2.2)–(2.3).

Kalkulator Harga: Harga per Baris

Kode 2.10 `LaundryPrice.java`, bagian 1: konstanta, daftar harga sebagai `switch`, dan harga satu baris.

```
// 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));
    }
```

Kode 2.10: konstanta `static final` dan minimum 3 kg. Buku: Subbab 2.11.1 dan Kode 2.9.

Kalkulator Harga: Totalnya, dengan Bug

Kode 2.11 `LaundryPrice.java`, bagian 2: `orderTotal`, tempat semua harga item bertemu. Versi ini mengandung bug.

```
    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;
    }
```

```
Customer : Bu Sari (member)
Total    : Rp 109350
Expected : Rp 93150
```

`main` meneruskan ketiga item ke Kode 2.11 dan mencetak totalnya di samping yang diharapkan. Versi Python mencetak total salah yang sama. Buku: Subbab 2.11.1 dan Kode 2.9–2.10.

Langkah 1: Ketahui Jawaban yang Seharusnya

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 2.5 memberi lebih dari jawabannya: total berjalan seharusnya melewati 24.000, lalu 36.000, lalu 103.500. Tempat pertama program menyimpang dari kolom ini adalah bugnya. Buku: Subbab 2.11.2 dan Tabel 2.5.

Langkah 2–3: Hipotesis dan Tur Terbimbing

```
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]);
```

Terlalu mahal Rp 16.200: diskonnya tidak hilang (itu akan memberi 103.500), jadi carilah di tempat harga-harga bertemu — breakpoint di baris 26. Di `jdb`: `step` masuk ke `pricePerKg`, `where` menampilkan empat bingkai, dua `step up` naik kembali. Item 1 dihargai benar: `line = 24000`. Buku: Subbab 2.11.3–2.11.4.

Langkah 4: Amati Total Berjalan

Breakpoint di `total += line;`, *watch* pada `total` dan `total * EXPRESS_PERCENT / 100`, lalu *Resume* dari item ke item.

Perhentian 1: `line = 24000`, `total = 0`. Perhentian 2: `line = 12000`, `total = 24000`. Keduanya benar.

Perhentian 3: `total = 81000` setelah ditambah 45.000; *watch* menunjukkan biaya tambahan $81{,}000 \times 50/100 = 40{,}500$, padahal Tabel 2.5 menyebut $22{,}500$. Selangkah kemudian `total` menjadi **121.500**, bukan 103.500.

> **PENJELASANNYA.** Biaya tambahan mengalikan total berjalan *semua* item: item 1 dan 2 membayar $36{,}000 \times 50/100 = 18{,}000$ terlalu banyak, dan $18{,}000 \times 90/100 = 16{,}200$ setelah diskon — persis kelebihan tagihannya.

Terjelaskan sampai rupiah terakhir: begitulah Anda tahu bahwa inilah sang bug, bukan sekadar sebuah bug. Buku: Subbab 2.11.5.

Bug Itu sebagai Rumus

$$t_i = (t_{i-1} + p'_i)\left(1 + \frac{e_i}{2}\right), \tag{2.4}$$

Di sini $p'_i = r(s_i)\max(w_i, 3)$, sedangkan Persamaan (2.2) dan Persamaan (2.3) meminta $t_i = t_{i-1} + p'_i\left(1 + \frac{e_i}{2}\right)$: faktornya diterapkan pada hal yang salah.

Persamaan (2.4) juga menyatakan kapan bug itu tak terlihat: bila satu-satunya item ekspres berada *pertama*, $t_{i-1} = 0$ dan keduanya sama. Uji dengan satu item ekspres tidak pernah menangkapnya.

Buku: Subbab 2.11.5 dan Persamaan (2.4).

Perburuan yang Sama di `pdb`, dengan `display`

```
(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
```

`display` berperan sebagai watch: di setiap perhentian `pdb` mencetak ulang setiap ekspresi yang nilainya berubah, beserta nilai lamanya. Baris terakhir mengatakan semuanya: `121500  [old: 81000]`. Versi `jdb` dari perburuan ini ada di buku. Buku: Subbab 2.11.5.

Langkah 5: Perbaiki, lalu Periksa

Kode 2.12 `LaundryPriceFixed.java`, satu-satunya metode yang berubah: biaya tambahan kini mengalikan `line`, dan `total += line` pindah ke sesudahnya.

```
    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;
    }
```

```
Customer : Bu Sari (member)
Total    : Rp 93150
Expected : Rp 93150
```

Dengan Kode 2.12 kedua bahasa mencetak Rp 93.150. Jalankan perbaikannya di bawah debugger sekali lagi dan pastikan 24.000, 36.000, dan 103.500: total yang benar dengan alasan yang salah adalah bug yang menunggu pelanggan berikutnya. Buku: Subbab 2.11.6 dan Kode 2.11–2.12.

Perbaikannya dalam Python

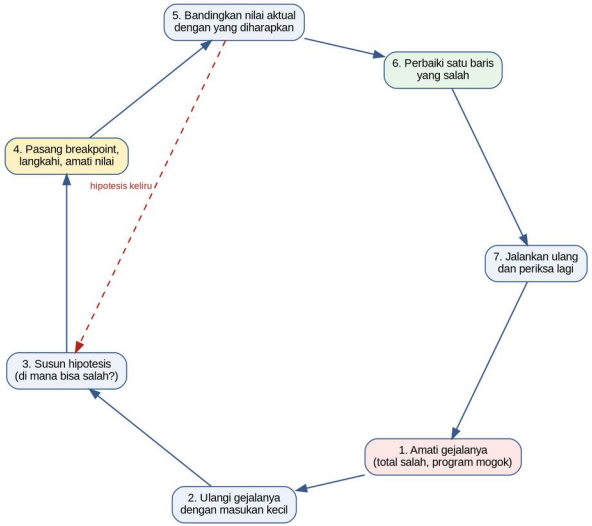
Kode 2.13 `laundry_price_fixed.py`, `order_total` yang sudah diperbaiki.

```
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
```

Perubahan satu baris yang sama: biaya tambahan pada `line`, dengan `//` untuk rupiah bulat, lalu `total += line`.

Kode 2.13 mencetak `Total     : Rp 93150`. Bab 6 mengubah pemeriksaan seperti ini menjadi pengujian otomatis. Buku: Subbab 2.11.6 dan Kode 2.12.

Metode di Balik Perburuan



Gambar 2.12 Siklus debugging. Bila perbandingan pada langkah 5 membantah hipotesis, kembalilah ke langkah 3 dengan membawa apa yang sudah Anda pelajari, bukan kembali menebak-nebak.

Gambar 2.12: amati, reproduksi, berhipotesis, pasang breakpoint, bandingkan dengan hitungan tangan, perbaiki, jalankan ulang. Hipotesis yang terbantah pun sebuah kemajuan. Buku: Subbab 2.11.7 dan Gambar 2.44.

Perburuan Terbimbing: Hasil Sehari Kurang Rp 6.000

Kode 2.14 `DailyOrders.java`, metode `linePrice`: sepuluh pesanan melewatinya, dan total hari itu kurang Rp 6.000.

```
    static long linePrice(String service, double kg) {
        long billedKg = (long) Math.max(kg, MIN_KG);
        return billedKg * pricePerKg(service);
    }
```

```
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
```

Berat yang ditagih tidak boleh kurang dari berat sebenarnya, maka Kode 2.14 diberi breakpoint bersyarat `billedKg < kg` pada `return`-nya. Buku: Subbab 2.12 dan Kode 2.13–2.14.

Hanya Dua Perhentian

```
(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)
```

7,5 kg ditagih 7, 3,5 kg ditagih 3: $0.5 \times 6{,}000 + 0.5 \times 6{,}000 = 6{,}000$. *Cast* memotong beratnya; ubahlah harganya, bukan beratnya.

Pesanan 9 (1,5 kg) tidak berhenti: minimum menaikkannya ke 3,0 lebih dulu. Buku: Subbab 2.12.

Membaca *Stack Trace* Java

Kode 2.15 `UnknownService.java`, `main`-nya: sebuah pesanan menyebut *dry-clean*, yang tidak dikenal daftar harga.

```
    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)
```

Baca **dari atas**: tidak tertangkap, di *thread* `main`; *tipe* `IllegalArgumentException`; *pesannya*; lalu pemanggilan, **yang terbaru lebih dulu** — baris 11 melempar, dipanggil dari baris 17, dipanggil dari baris 24.

Baris yang melempar tidak selalu baris yang salah: `pricePerKg` benar menolak, dan pertanyaan sebenarnya adalah mengapa `main` memintanya. Pada Kode 2.15 dan kembaran Python-nya, carilah bingkai yang menyebut kode Anda sendiri. Buku: Subbab 2.13–2.13.1 dan Kode 2.15.

Membaca *Traceback* Python

```
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
```

`Traceback (most recent call last)`: baca **dari bawah** — tipe dan pesan, lalu bingkai terdalam (baris 13), lalu para pemanggilnya. Tanda sisipnya menunjuk pemanggilan yang tepat.

> **WAWASAN KUNCI:** *Stack trace* adalah tumpukan pemanggilan yang dicetak saat program mogok: tipe eksepsi, pesan, rantai pemanggilan. Bacalah ketiganya sebelum mengubah satu baris pun.

Dengan breakpoint eksepsi, atau `pdb` dalam mode post-mortem, program berhenti pada `raise` dengan setiap bingkai masih dapat diperiksa. Buku: Subbab 2.13.2 dan Kode 2.16.

Profitina dalam Praktik

Reproduksi sebelum memperbaiki: skor yang “tampak salah” lebih dulu direproduksi pada salinan uji, seperti langkah 2 pada Gambar 2.12; bug yang diperbaiki menjadi pengujian otomatis, dan lebih dari seribu pengujian menjaga setiap rilis.

***Stack trace* adalah data:** server web tidak dapat berhenti di *breakpoint* di depan pengguna, maka ia mencatat *traceback* — tanpa data pribadi, sejalan dengan UU PDP 27/2022 — untuk dibaca dari bawah.

Ketahui mesin mana menjalankan apa: interpreter mana, model mana (Qwen2.5 secara lokal melalui Ollama), konfigurasi mana. *Endpoint* kesehatan yang melaporkan versi terpasang adalah *status bar* dunia produksi.

Aplikasi menyarankan dan pengguna memutuskan; debugging membuat sarannya benar, tidak pernah menjadikannya jaminan. Buku: Subbab 2.14.

Kesalahan yang Sering Terjadi

Menjalankan program lama (editor belum disimpan, atau tanpa `javac`). **Python yang salah** — periksa *status bar*.

Menjalankan modul paket sebagai skrip — pakai `python3 -m main.tester`. **Paket dan folder tidak cocok** — buat dan pindahkan kelas melalui IDE.

Run alih-alih Debug; `java main/Tester.class` alih-alih `java -cp bin main.Tester`.

Masuk ke segala hal dengan *step into*; **berhenti di nilai salah pertama tanpa menjelaskannya** sampai rupiah terakhir.

Membaca *trace* dari ujung yang salah — Java terbaru di atas, Python terbaru di bawah. **Meninggalkan cetakan *debug*** — utamakan *watch* dan *logpoint*.

Buku: Subbab 2.15.

Ringkasan — Lima Hal untuk Diingat

1. IDE memadukan penyuntingan, pembangunan, eksekusi, *debugging*, dan penjelajahan; Tabel 2.2 menerjemahkan antara Eclipse dan VS Code.

2. Java mengompilasi sekali ke *bytecode* dan JVM menjalankannya; Python mengompilasi di dalam interpreter setiap kali mulai. Java menemukan nama yang hilang saat kompilasi, Python saat berjalan.

3. Paket adalah folder; jalankan `java -cp bin paket.Kelas` dan `python -m paket.modul`.

4. *Breakpoint*, *step over/into/out*, variabel, *watch*, tumpukan pemanggilan; *breakpoint* bersyarat, *hit count*, dan eksepsi berhenti hanya di tempat yang penting.

5. *Debug* dengan metode (Gambar 2.12); tagihan *laundry* terlalu mahal karena biaya tambahan mengalikan total berjalan (Persamaan (2.4)).

Buku: Subbab 2.16.

Pertemuan Berikutnya

Bab 3 — Kelas, Objek, String, Tipe, Percabangan, dan Perulangan.

Setiap bagian kelas — *field*, konstruktor, `this` dan `self`, *getter*, *setter*, `toString` — dan gambar memori tentang referensi.

String, delapan tipe primitif, tiga jebakan aritmetika, `if` dan `switch`, perulangan dan tabel penelusuran; `Customer` dan `Service` milik Profitina Laundry.

Sebelum kita bertemu: Soal Latihan 2.2, 2.5, 2.7, dan 2.9 — kapan galat tertangkap, jalur melangkah, penghitung yang terus menghitung, dan bug yang bersembunyi bila item ekspres berada pertama. Penyelesaiannya ada di Subbab 2.18.

Buku: Subbab 2.17–2.18, lalu Bab 3.

Pemrograman Berorientasi Objek (EF234302)

Kuliah 3 — Kelas, Objek, String, Tipe, Percabangan, dan Perulangan

Buku: Irfan Subakti, *Pemrograman Berorientasi Objek dengan Java dan Python*, Bab 3

Dosen: Irfan Subakti — `yifana@gmail.com`

Departemen Teknik Informatika, Institut Teknologi Sepuluh Nopember, Surabaya

Hari ini kita membuka sebuah program dan menjajarkan setiap bagiannya di meja kerja, lalu membangun dua kelas pertama Profitina Laundry. Buku: Bab 3.

Agenda

1. Di dalam kelas: *field*, konstruktor, metode, `this` dan `self`; membuat objek; memeriksa bahwa kelas bekerja.

2. Referensi dan gambar memori: nilai, panah, alias, identitas dan kesamaan.

3. *String* dan tipe: posisi, imutabilitas, format; delapan tipe primitif; pengetikan statis dan dinamis.

4. Tiga jebakan aritmetika — luapan, *floating point*, pembagian bilangan bulat — lalu `if`, `switch`, perulangan, dan tabel penelusuran.

5. Latihan terbimbing, serta `Customer` dan `Service` milik Profitina Laundry, diperiksa dengan tangan.

Buku: Bab 3, Subbab 3.1–3.17.

Titik Berangkat Kita

Bab 1 menyiapkan meja kerja; Bab 2 membuatnya nyaman — proyek, paket, dan *debugger*. Program-programnya sendiri masih mungil.

Dua kelompok bagian: **struktur** — kelas dan objek; dan **apa yang terjadi di dalam metode** — nilai, keputusan, pengulangan.

> **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.

Pemrogram yang belum menguasai kelompok kedua menulis kelas yang indah tetapi menghitung jawaban yang salah. Buku: Subbab 3.1.

Hello, Sekali Lagi, Kata demi Kata

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!");
    }
}
```

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()
```

*Pada **Kode 3.1** semua kode tinggal di dalam kelas, kurung kurawal menandai blok, dan huruf `S`, titik, tanda kutip, serta titik koma semuanya wajib. **Kode 3.2** menandai blok dengan **indentasi** dan memanggil `main` hanya bila berkasnya dijalankan langsung. Buku: Subbab 3.2.1 dan Kode 3.1–3.2.*

Membaca Kata dari Baris Perintah

Kode 3.3 `Hello2.java`: mencetak kata pertama yang diketik di baris perintah.

```java
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!");
        }
    }
}
```

```
java Hello2 Welcome to Profitina Laundry
```

```
She said, Welcome!
```

*Pada **Kode 3.3**, `args[0]` adalah kata pertama — penomoran dimulai dari 0 — dan `+` **menyambung** teks. Tanpa uji `args.length > 0`, ketiadaan kata akan memicu `ArrayIndexOutOfBoundsException`. Di Python kata pertama adalah `sys.argv[1]`, karena `sys.argv[0]` adalah nama skripnya. Buku: Subbab 3.2.2 dan Kode 3.3.*

Nama, Konvensi, dan Komentar

Java: kelas `LaundryDemo`, variabel dan metode dalam **camel case** `aLineOfText`, konstanta `MAX_VALUE`. **Python** (PEP 8): kelas `LaundryService`, variabel dan fungsi dalam **snake case** `a_line_of_text`.

Aturan, bukan konvensi: `String`, bukan `string`; `print`, bukan `Print`.

Komentar: baris (`//`, `#`), blok (`/* … */`), dan dokumentasi (Javadoc `/** … */`, *docstring* Python). Jelaskan *mengapa*, bukan *apa*:

```
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 di atas mengulang kodenya dan tidak menambah apa-apa; “anggota mendapat diskon 10%” adalah komentar yang baik. Buku: Subbab 3.2.3–3.2.4 dan Kode 3.5–3.6.

Cetak Biru dan Rumah

Kelas adalah cetak biru; **objek** adalah rumah yang dibangun darinya — benda nyata dengan perabotnya sendiri. *Objek* dan *instance* bermakna sama.

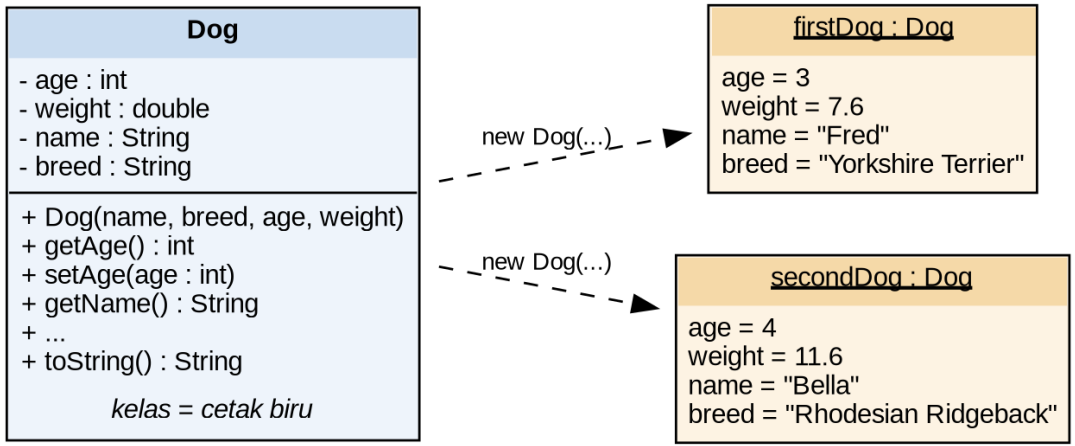
Kelas `Dog` menyatakan bahwa setiap anjing memiliki nama, ras, umur, dan berat. Fred dan Bella adalah dua *instance*.

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**: baris judul adalah kelasnya, setiap baris di bawahnya satu objek. Buku: Subbab 3.3.1 dan Tabel 3.1.*

Kelas dan Dua Objek dalam UML



Gambar 3.1 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 pada **Gambar 3.1** mudah terlewat: setiap objek memiliki salinan sendiri dari setiap field. `age` milik Fred dan `age` milik Bella adalah dua kotak berbeda. Buku: Subbab 3.3.1 dan Gambar 3.3.*

`Dog.java`: *Field* dan Konstruktor

Kode 3.4 `Dog.java`, bagian 1: empat *field* privat dan satu konstruktor.

```
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;
    }
```

Field menyimpan data setiap objek; `private` menyembunyikannya di balik metode — **enkapsulasi**. **Konstruktor** bernama sama dengan kelas dan tanpa tipe kembalian; **parameternya** menerima nilai dari pemanggil. Tanpa `main`: `Dog` adalah cetak biru, bukan program.

Kode 3.4 *membuka kelas lengkap yang ada di buku. Buku: Subbab 3.3.2 dan Kode 3.7.*

`Dog.java`: *Getter*, *Setter*, dan `toString`

Kode 3.5 `Dog.java`, bagian 2: *getter* dan *setter* untuk `age`; berat, nama, dan ras mengikuti pola yang sama.

```
    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }
```

Kode 3.6 `Dog.java`, bagian 3: `toString`, yang menggambarkan seluruh anjing sebagai teks.

```
    @Override
    public String toString() {
        return "Dog name: " + getName() + "; breed: " + getBreed()
                + "; age: " + getAge() + "; weight: " + getWeight() + " kg";
    }
}
```

Getter *mengembalikan satu field,* ***setter*** *mengubah satu — dan menjadi gerbang tempat* `if (age < 0) return;` *kelak melindungi setiap pemanggil sekaligus (**Kode 3.5**).* `@Override` *pada **Kode 3.6** menangkap salah ketik* `tostring`*. Buku: Subbab 3.3.2 dan 3.3.4 serta Kode 3.7.*

Bagian-Bagian Kelas, Berdampingan

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

***Tabel 3.2**: bagian-bagian Java dan padanannya di Python. Buku: Subbab 3.3.2 dan Tabel 3.2.*

`dog.py`: Kelas yang Sama dalam Python

Kode 3.7 `dog.py`, bagian 1: *field* diciptakan di dalam `__init__`; sebuah *getter* dan sebuah *setter*.

```
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
```

*Pada **Kode 3.7**: tanpa deklarasi field, tanpa `private` (`_age` hanyalah konvensi), tanpa tipe; `__str__` berperan sebagai `toString`. Buku: Subbab 3.3.2 dan 3.3.4 serta Kode 3.8.*

Kata `this`, dan Kata `self`

```
public Dog(String name, String breed, int age, double weight) {
    this.name = name;
    ...
}
```

Ada dua hal bernama `name`: *field* dan parameter. `name = name;` menyalin parameter ke dirinya sendiri — bug diam-diam yang membiarkan *field* tetap `null`.

> **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.

Python mewajibkan `self`: `first_dog.get_age()` sebenarnya adalah `Dog.get_age(first_dog)`. Buku: Subbab 3.3.3.

Membuat Objek dengan `new`

Kode 3.8 `DogDemo.java`, `main`-nya: dua objek `Dog` dibuat, dicetak, dan diubah melalui *setter*.

```
    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
    }
```

```
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
```

*`new` memanggil konstruktor dengan **argumen** dan mengembalikan **referensi**; perubahan pada Fred di **Kode 3.8** tidak menyentuh Bella. Buku: Subbab 3.3.5 dan Kode 3.9.*

Demonstrasinya dalam Python

Kode 3.9 `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)

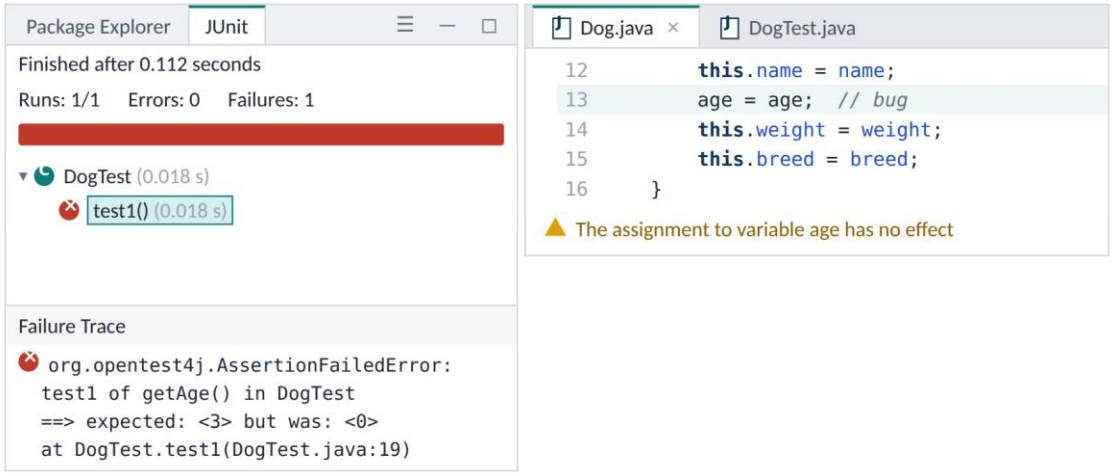
    print(first_dog)
    print(str(second_dog))  # efeknya sama dengan baris di atas

    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()
```

*Tanpa `new`: kelas dipanggil seperti fungsi. `+` di Java mengubah bilangan menjadi teks; Python menuntut `str(...)`. Keluaran **Kode 3.9** sama, empat baris. Buku: Subbab 3.3.5 dan Kode 3.10.*

Memeriksa Bahwa Kelas Bekerja



Gambar 3.2 Bilah merah: `test1` gagal dengan harapan 3 tetapi ternyata 0, karena konstruktor `Dog` versi ini menulis `age = age;` alih-alih `this.age = age;`.

*`DogCheck` membandingkan nilai harapan dan nilai nyata lalu mencetak PASS atau FAIL (berat dalam toleransi 10^{-9}). JUnit melakukannya dalam skala besar: pada **Gambar 3.2** konstruktor yang menulis `age = age;` menghasilkan `expected: <3> but was: <0>`; dengan `this.age = age;` bilahnya menjadi hijau. Setiap uji: arrange, act, assert. Buku: Subbab 3.3.6, Kode 3.11–3.12, dan Gambar 3.6–3.8.*

Nilai dan Panah

Kode 3.10 `ReferenceDemo.java`, paruh referensinya: sebuah alias, kembaran berisi sama, dan `null`.

```
        // 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());

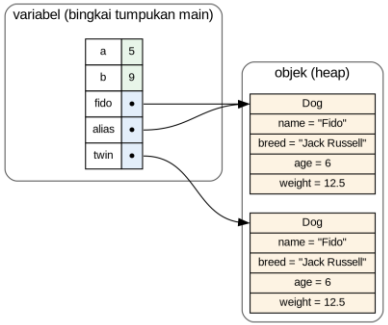
        Dog twin = new Dog("Fido", "Jack Russell", 6, 12.5);
        System.out.println("fido == alias: " + (fido == alias));
        System.out.println("fido == twin:  " + (fido == twin));

        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);
    }
```

Variabel **primitif** menyimpan nilai; variabel **referensi** menyimpan panah ke objek di ***heap***. `Dog alias = fido;` menyalin *panahnya*: dua **alias**, satu anjing.

Kode 3.10 *diawali dengan* `int b = a; b = 9;`*, yang tidak menyentuh* `a`*. Buku: Subbab 3.4.1 dan Kode 3.13.*

Gambar Memorinya



Gambar 3.3 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.

```
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 3.3**: `==` pada referensi bertanya "objek yang sama?"; memanggil melalui `null` memicu `NullPointerException`. Buku: Subbab 3.4.1 dan Gambar 3.10.*

Python: Setiap Variabel Adalah Nama

Kode 3.11 `reference_demo.py`, cuplikan: `is` menguji identitas.

```
# 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))
```

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`

Kode 3.11 *mencetak apa yang dicetak Java;* ***Tabel 3.3*** *mengeja kedua pertanyaan itu dengan cara berlawanan. Buku: Subbab 3.4.2–3.4.3, Kode 3.14, dan Tabel 3.3.*

String: Posisi dan Potongan

Kode 3.12 `StringBasics.java`, cuplikan: `substring`, `charAt`, dan `length`.

```
        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());
```

Kode 3.13 `string_basics.py`, cuplikan yang sama: *slice*, pengindeksan, dan `len`.

```
    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)))
```

Awal disertakan, akhir tidak: ***Kode 3.12*** *dan* ***Kode 3.13*** *mencetak* `Yarik / Hello`, `d`, `5`. *Buku: Subbab 3.5.1–3.5.2 dan Kode 3.15–3.16.*

Operasi *String* Sehari-hari

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()`

Tugas	Java (`String s`)	Python (`str s`)
Ulangi	`s.repeat(3)`	`s * 3`
Isi sama	`s.equals(t)`	`s == t`

Tabel 3.4 *adalah ringkasan untuk selalu di tangan. Buku: Subbab 3.5.2 dan Tabel 3.4.*

String Tidak Pernah Berubah

Kode 3.14 `StringImmutability.java`, intinya: metode mengembalikan *string* baru; `==` membandingkan panah, `equals` membandingkan karakter.

```
        String s = "laundry";
        String t = s.toUpperCase();   // membangun string BARU; s tidak berubah
        System.out.println("s = " + s + ", t = " + t);

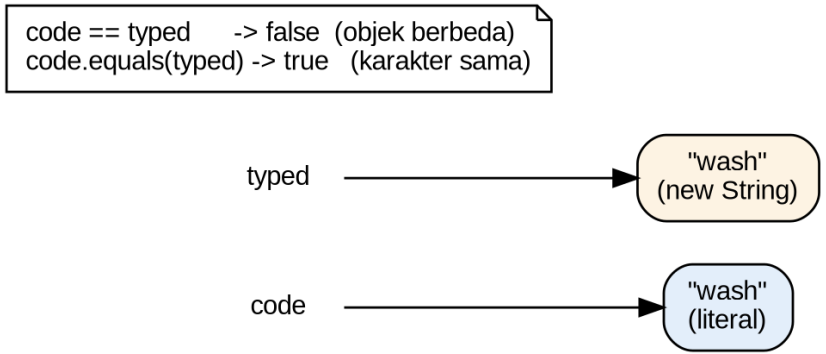
        s = s + " day";               // s kini menunjuk ke string baru yang lain
        System.out.println("s = " + s);

        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));
```

```
s = laundry, t = LAUNDRY
s = laundry day
code == typed:      false
code.equals(typed): true
```

*`toUpperCase` membangun string baru, dan `s = s + " day"` memindahkan panahnya. Untuk banyak potongan pakai `StringBuilder` (Python: `"".join(parts)`). Pada **Kode 3.14**, `code == typed` bernilai `false`. Buku: Subbab 3.5.3, Kode 3.17, dan Gambar 3.12.*

Karakter Sama, Dua Objek



Gambar 3.4 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 `==`. `"wash" == "wash"` kebetulan `true` untuk literal, tetapi teks yang diketik, dibaca, atau dibangun saat berjalan adalah objek baru.

***Gambar 3.4**: Python terhindar dari jebakan ini karena `==`-nya membandingkan isi. Buku: Subbab 3.5.3 dan Gambar 3.13.*

Memformat Keluaran, dan Mengurai Masukan

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%`

`printf` mencetak, `String.format` mengembalikan; di Python *f-string* `f"{weight_kg:.2f}"`. `%,d` di Java mengikuti ***locale*** komputer: `24.192` pada pengaturan Indonesia.

Uraikan teks lebih dulu: `Integer.parseInt`, `Double.parseDouble`; Python `int(...)`, `float(...)`. `"12" + "3"` adalah `"123"`; `Integer.parseInt("12") + 3` adalah `15`.

Tabel 3.5 *mendaftar penampung yang dipakai buku;* `abc` *gagal dengan* `NumberFormatException` *atau* `ValueError`*. Buku: Subbab 3.5.4–3.5.5 dan Tabel 3.5.*

Delapan Tipe Primitif Java

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`

$$-2^{b-1} \leq v \leq 2^{b-1} - 1 \tag{3.1}$$

*Menurut Persamaan (3.1), `byte` menampung -2^7 sampai $2^7 - 1$ dan `int` -2^{31} sampai $2^{31} - 1$. Pilih `int`/`long` dan `double`; Python punya satu `int` tanpa batas dan satu `float` (**Tabel 3.6**). Buku: Subbab 3.6.1, Tabel 3.6, dan Persamaan (3.1).*

Pengetikan Statis di Java

Kode 3.15 `StaticTyping.java`: di Java setiap variabel memiliki satu tipe seumur hidupnya; `var` membiarkan kompilator menyimpulkan tipenya.

```
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
    }
}
```

```
StaticTyping.java:11: error: incompatible types: String cannot be converted to int
        count = "three";
                ^
1 error
```

*Pada **Kode 3.15** sebuah `int` **diperlebar** menjadi `double`; `var` meminta kompilator menyimpulkan `String`; `7000L` adalah `long`. Buka komentar `count = "three";` dan kompilator menolak, seperti di atas. Buku: Subbab 3.6.2 dan Kode 3.23.*

Pengetikan Dinamis, tetapi Kuat, di Python

Kode 3.16 `dynamic_typing.py`, `main`-nya: sebuah nama boleh diikat ulang ke tipe lain; tipenya diperiksa ketika `+` 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)
```

```
3 int
three str
7000 int
21000 int True
TypeError: can only concatenate str (not "int") to str
```

***Kode 3.16**: **type hint** tidak dipaksakan; diperiksa belakangan, tetapi tetap diperiksa. Buku: Subbab 3.6.2 dan Kode 3.24.*

Karakter, Nilai Kebenaran, dan Operator

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`

***Tabel 3.7**: `*` sebelum `+`; Java tidak punya operator pangkat (`Math.pow`), Python menulis `**`. `char` adalah bilangan di baliknya (`'B'` adalah 66). `&&` dan `||` bersifat **short-circuit**, sehingga `note != null && note.length() > 0` aman. Buku: Subbab 3.6.3–3.7.1 dan Tabel 3.7.*

Jebakan Pertama: Luapan Bilangan Bulat

Kode 3.17 `IntegerTypes.java`, luapannya: x, y, z, maupun jawabannya tidak ada yang dekat batas, tetapi hasilnya negatif.

```
        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));
```

```
x * y / z as int : -294967
x * y as int     : -1474836480
x * y / z as long: 4000000
MAX_VALUE + 1    : -2147483648
```

*Pada **Kode 3.17** hasil kali antara 2×10^{10} melampaui $2^{31} - 1$. Obatnya: `(long) x * y / z` — cast sebelum perkalian. Buku: Subbab 3.7.2 dan Kode 3.27.*

Luapan Itu, Dihitung dengan Tangan

$$\text{tersimpan}(v) = ((v + 2^{31}) \bmod 2^{32}) - 2^{31} \tag{3.2}$$

$2^{32} = 4\,294\,967\,296$; $v + 2^{31} = 20\,000\,000\,000 + 2\,147\,483\,648 = 22\,147\,483\,648$.

Dibagi 2^{32} masuk 5 kali, $5 \times 4\,294\,967\,296 = 21\,474\,836\,480$; sisanya 672 647 168.

$672\,647\,168 - 2\,147\,483\,648 = -1\,474\,836\,480$ — baris “x * y as int”. Dibagi 5 000 dan pecahannya dibuang: $-294\,967$.

> **PERINGATAN:** Luapan terjadi diam-diam. Kalikan uang dalam `long`, atau pakai `Math.multiplyExact`. `int` Python tidak pernah meluap; `wrap_int32`-nya meniru Java.

Persamaan (3.2) hanya menyimpan 32 bit terbawah, seperti odometer yang berputar kembali. Buku: Subbab 3.7.2, Persamaan (3.2), dan Kode 3.28.

Jebakan Kedua: Persepuluhan yang Tak Genap Satu

Kode 3.18 `FloatingPoint.java`, intinya: sepuluh kali sepersepuluh bukan satu, penjumlahan tidak asosiatif, dan `==` gagal.

```
        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

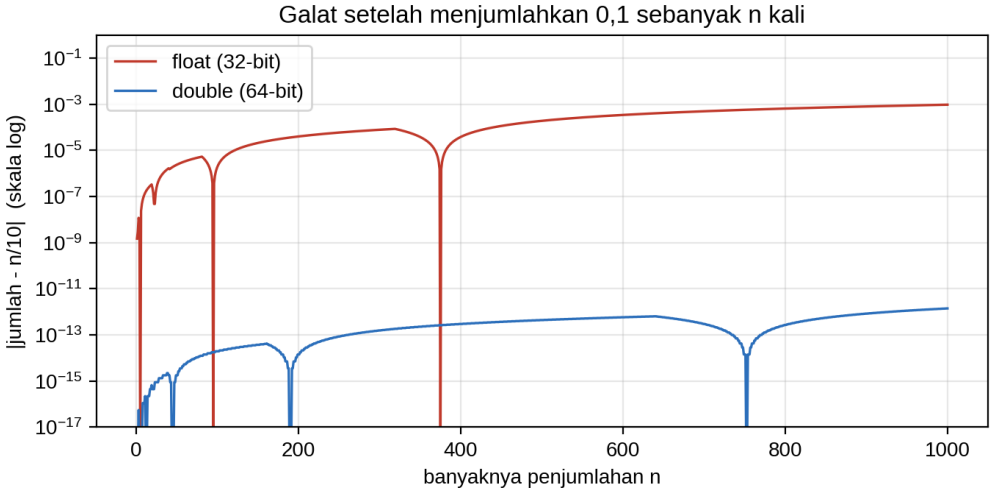
        System.out.println("0.1 + 0.2 + 0.3 = " + (0.1 + 0.2 + 0.3));
        System.out.println("0.2 + 0.3 + 0.1 = " + (0.2 + 0.3 + 0.1));

        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
```

*0.1 tidak punya bentuk biner yang tepat, seperti $\frac{1}{3}$ tidak punya bentuk desimal yang tepat; **Kode 3.18** menunjukkan akibatnya. Buku: Subbab 3.7.3 dan Kode 3.29.*

Bagaimana Galatnya Tumbuh



Gambar 3.5 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.

*Pada **Gambar 3.5** galat `float` kira-kira semiliar kali galat `double`; lembah-lembahnya adalah saat galat pembulatan saling meniadakan. Buku: Subbab 3.7.3 dan Gambar 3.14.*

Bandingkan dengan Toleransi

$$|a - b| < \varepsilon \tag{3.3}$$

Java: `Math.abs(a - b) < 1e-9`. Python: sama, atau `math.isclose(a, b)`; `Decimal("0.1") * 10` tepat `1.0`.

> **WAWASAN KUNCI:** Uang tidak boleh disimpan sebagai `double`. Simpanlah rupiah bulat dalam `long` (Java) atau `int` (Python), seperti kelas-kelas Profitina Laundry, atau pakailah `BigDecimal` / `Decimal`.

Persamaan (3.3) persis yang dipakai `DogCheck` untuk berat. Buku: Subbab 3.7.3, Persamaan (3.3), dan Kode 3.30.

Jebakan Ketiga: Pembagian Bilangan Bulat

$$a = (a/b) \cdot b + (a \bmod b) \tag{3.4}$$

Kode 3.19 `IntegerDivision.java`, intinya: `x / y` antarbilangan bulat, *cast* sebelum dan sesudah, dan operand negatif.

```
        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));
```

```
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
```

*Persamaan (3.4) mengikat `/` dan `%`; `n % 10` adalah digit terakhir. `(double) (x / y)` membagi dulu dan sudah terlambat. Pada **Kode 3.19** Java memotong ke arah nol. Buku: Subbab 3.7.4 dan Kode 3.31.*

Pembagian dan Pembulatan, Berdampingan

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**: `/` Python selalu menghasilkan desimal dan `//` membulatkan ke bawah, sehingga `-7 // 2` adalah `-4`; `round(2.5)` adalah `2` (setengah ke genap), sedangkan `Math.round(2.5)` adalah `3`. Kedua bahasa mematuhi Persamaan (3.4). Buku: Subbab 3.7.4 dan Tabel 3.8.*

Melakukan Sesuatu Hanya Bila Kondisinya Terpenuhi

Kode 3.20 `IfDemo.java`, bagian 1: sebuah `if` dengan `else`. Apa yang dicetaknya, dan bagaimana jika `i` bernilai 1?

```
        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);
        }
```

```
i is less than or equal to 0
j is 3
```

Kondisinya harus `boolean`; tepat satu blok yang dijalankan. Dengan `i = 1`: `i is greater than 0!` dan `j is 2`. Selalu tulis kurung kurawalnya. ***Kode 3.20*** *membuka teka-teki di buku. Buku: Subbab 3.8.1 dan Kode 3.33.*

Rantai Keputusan

Kode 3.21 `if_demo.py`, bagian 2: rantai `elif` yang dicoba untuk $k = 2, \ldots, 6$.

```
    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")
```

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`

*Kondisi benar pertama yang menang (**Tabel 3.9**); **Kode 3.21** adalah rantainya dalam Python. Buku: Subbab 3.8.2, Kode 3.34, Tabel 3.9, dan Gambar 3.15.*

`switch` dan Jebakannya

Kode 3.22 `SwitchDemo.java`, bagian 1: `switch` klasik tanpa `break`.

```
        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");
            }
        }
```

```
-- without break --
Input was b
Input was c
```

*Label `case` hanyalah pintu masuk: dari `case 'b':` eksekusi **jatuh terus** (fall-through) ke `case 'c':`. Obatnya `break`, ditambah `default` untuk yang tak cocok (**Kode 3.22**). Buku: Subbab 3.8.3, Kode 3.35, dan Gambar 3.16.*

Bentuk Panah, dan `match` di Python

Kode 3.23 `SwitchDemo.java`, bagian 3: bentuk panah (Java 14+), `switch` yang menghasilkan nilai.

```
        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);
```

Kode 3.24 `switch_demo.py`, `describe`: `match` tidak pernah jatuh terus; `case _:` adalah bawaannya.

```
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"
```

*Utamakan bentuk panah (**Kode 3.23**); **Kode 3.24** memerlukan Python 3.10+. Buku: Subbab 3.8.3 dan Kode 3.35–3.36.*

`while`, `for`, dan `do-while`

Kode 3.25 `LoopBasics.java`: perulangan `while` yang sudah diperbaiki, perulangan `for` yang setara, dan perulangan `do-while`.

```
public class LoopBasics {

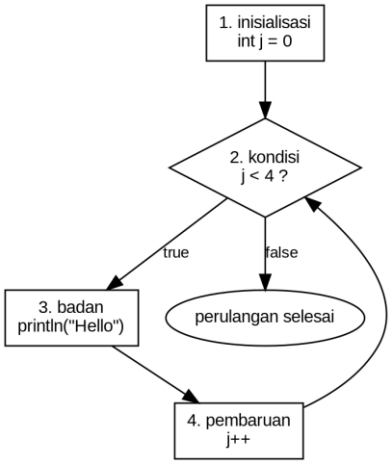
    public static void main(String[] args) {
        int i = 0;
        while (i < 4) {
            System.out.println("Hello (while, i = " + i + ")");
            i++;                        // tanpa baris ini perulangan tidak pernah berhenti
        }

        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);
    }
}
```

*Tanpa `i++` gelung `while` pada **Kode 3.25** tidak pernah berhenti — **gelung tak berujung**. `do-while` berjalan sekali walaupun `k < 4` salah. Buku: Subbab 3.9.1 dan Kode 3.37.*

Empat Bagian Gelung `for`



Gambar 3.6 Empat bagian `for (int j = 0; j < 4; j++)`: inisialisasi sekali, lalu kondisi, badan, dan pembaruan dalam satu siklus sampai kondisinya salah.

***Gambar 3.6**: inisialisasi sekali; lalu kondisi, badan, pembaruan, sampai kondisinya salah — jadi `j` bernilai 0, 1, 2, 3. `for j in range(4):` di Python sama; `while True:` dengan `break` menggantikan `do-while`. Buku: Subbab 3.9.1 dan Gambar 3.17.*

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

Empat deret: `for (i = 1; i <= 10; i++)`, `for (i = 10; i >= 1; i--)`, `i += 3` sampai 15, dan `i * i` untuk kuadrat — Python `range(1, 11)`, `range(10, 0, -1)`, `range(3, 16, 3)`. `i < 10` alih-alih `i <= 10` adalah **galat kelebihan-atau-kekurangan satu** (*off-by-one*).

***Tabel 3.10**: pilih gelung yang bentuknya memberi tahu pembaca jenis pengulangannya. Buku: Subbab 3.9.2–3.9.4, Kode 3.39–3.40, dan Tabel 3.10.*

Perulangan For-Each

Kode 3.26 `ArgsEcho.java`: mengunjungi setiap kata baris perintah melalui indeks dan dengan perulangan for-each.

```
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();
    }
}
```

```
You gave 3 word(s).
args[0] = wash
args[1] = dry
args[2] = iron
WASH DRY IRON
```

Jalankan dengan `java ArgsEcho wash dry iron`*. Gelung for-each pada* ***Kode 3.26*** *tidak punya indeks yang bisa salah;* `for word in args:` *di Python sama, dan* `enumerate` *memberi indeks sekaligus nilai. Buku: Subbab 3.9.3 dan Kode 3.41.*

Tabel Penelusuran

Kode 3.27 `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);
    }
}
```

*`n % 10` adalah digit terakhir dan `n / 10` membuangnya (Python `n // 10`). **Kode 3.27** mencetak `digit sum = 12` dan `reversed = 1704`. Buku: Subbab 3.10 dan Kode 3.43.*

Menelusuri $n = 4071$ dengan Tangan

Tabel 3.11 Menelusuri **Kode 3.27** 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

Satu kolom per variabel yang berubah ditambah kondisinya; baris awal; jalankan *berurutan*, dengan nilai *sebagaimana saat itu*; berhenti ketika kondisinya salah.

***Tabel 3.11**: satu putaran per digit; nol di ujung 4071 menjadi nol di depan yang lenyap. Bab 4 mengerjakan soal-soal utuh semacam ini. Buku: Subbab 3.10 dan Tabel 3.11.*

Latihan Terbimbing: Luas Lingkaran

$$A = \pi r^2 \tag{3.5}$$

Kode 3.28 `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);
    }
}
```

*Keduanya `double`, karena π tidak bulat; **Kode 3.28** mencetak `78.53981633974483` dan, setelah diformat, `78.54` — dengan tangan* $25 \times 3.14159 \ldots$*, sesuai Persamaan (3.5). Buku: Subbab 3.11.1 dan Kode 3.45.*

Latihan Terbimbing: Satuan Imperial ke Kilogram

$$W_{\text{lb}} = 2240t + 112h + 28q + 14s + p + \frac{o}{16} + \frac{d}{256} + \frac{g}{7000} \tag{3.6}$$

$$W_{\text{kg}} = 0.45359237 \cdot W_{\text{lb}} \tag{3.7}$$

> **DENGAN TANGAN.** 11 *stone* dan 6 *pound*: $14 \times 11 + 6 = 160$ lb menurut Persamaan (3.6); $160 \times 0.45359237 = 72.5747792$ kg menurut Persamaan (3.7). Programnya mencetak `72.57477920000001`.

Beri nama faktornya (`poundsPerStone`), jangan sebar **bilangan ajaib**; tulis `1 / 16.0`, karena `1 / 16` bernilai 0.

Tabel satuannya adalah Tabel 3.12; programnya Kode 3.47–3.48. Buku: Subbab 3.11.2 dan Persamaan (3.6)–(3.7).

Latihan Terbimbing: Lima Ratus Tahun Bunga

$$\text{total} = \text{modal} \cdot (1 + 0.01 \cdot \text{suku bunga})^{\text{tahun}} \tag{3.8}$$

Tabel 3.12 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

*Persamaan (3.8): setiap tahun mengalikan dengan 1.023, jadi n tahun mengalikan dengan 1.023^n. **Tabel 3.12** memeriksa lima tahun pertama dengan tangan; versi berulang enam pernyataan adalah Kode 3.49. Buku: Subbab 3.11.3 dan Tabel 3.13.*

Menghapus Pengulangan dengan Gelung

Kode 3.29 `InvestmentLoop.java`, `main`-nya: saldo tahun demi tahun, mencetak lima tahun pertama dan setiap tahun keseratus.

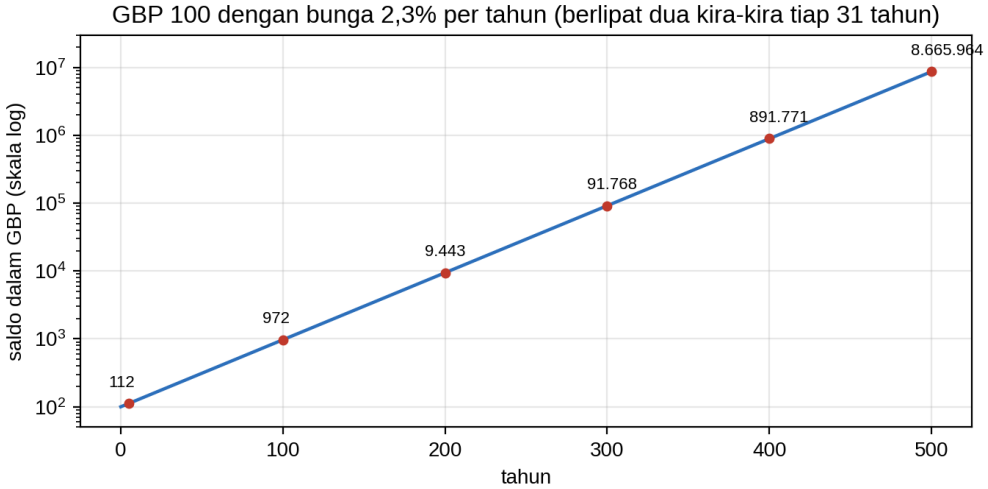
```
        double capital = 100;       // GBP (pound sterling)
        double interestRate = 2.3;  // persen per tahun
        double factor = 1 + 0.01 * interestRate;

        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);
```

```
 100            971.77
 200           9443.36
 300          91767.76
 400         891771.29
 500        8665963.51
```

*Pada **Kode 3.29** 500 perkalian sepakat dengan satu `Math.pow` sampai ke sen: `8665963.51`. Buku: Subbab 3.11.3 dan Kode 3.51.*

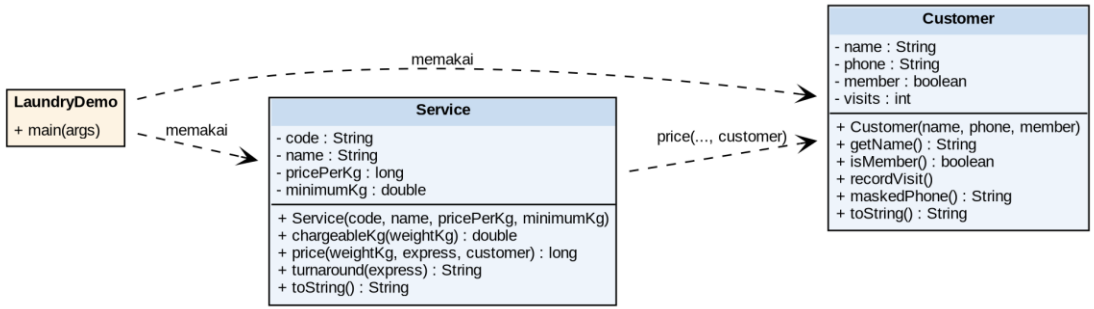
Tanda Tangan Pertumbuhan Majemuk



Gambar 3.7 Saldo GBP 100 dengan bunga 2,3% per tahun selama 500 tahun, pada skala logaritmik; garis lurus adalah ciri khas pertumbuhan majemuk.

*Pada sumbu logaritmik **Gambar 3.7**, pertumbuhan dengan faktor yang sama setiap tahun adalah garis lurus; saldo berlipat dua kira-kira setiap 31 tahun. Buku: Subbab 3.11.3 dan Gambar 3.18.*

Profitina Laundry: `Customer` dan `Service`



Gambar 3.8 Dua kelas pertama Profitina Laundry dan program yang memakainya.

Wash & Fold Rp7.000/kg, Iron Only Rp5.000/kg, Wash, Dry & Iron Rp10.000/kg; berat minimum; kelipatan setengah kilogram dibulatkan ke atas; ekspres +50%; anggota diskon 10%.

*Pada **Gambar 3.8**: nama dan telepon tidak pernah berubah, jadi tanpa setter dan `final`; `recordVisit()` sebagai ganti setter; `maskedPhone()` menampilkan tiga digit. Buku: Subbab 3.12–3.12.1 dan Gambar 3.19.*

Aturan Harga sebagai Rumus

$$c = \frac{\lceil 2 \cdot \max(w, m) \rceil}{2} \tag{3.9}$$

$$p = \begin{cases} c \cdot r & \text{reguler} \\ \frac{3}{2}\, c \cdot r & \text{ekspres} \end{cases}, \qquad \text{harga} = \begin{cases} p & \text{tamu} \\ p - \lfloor p/10 \rfloor & \text{anggota} \end{cases} \tag{3.10}$$

DENGAN TANGAN. $w = 4.2$, $m = 3$: $2w = 8.4$, dibulatkan ke atas 9, dibagi dua 4.5 kg menurut Persamaan (3.9). Dengan Rp7.000: 31 500; anggota membayar $31\,500 - 3\,150 = 28\,350$ menurut Persamaan (3.10).

Sesudah Persamaan (3.9) setiap langkah adalah aritmetika bilangan bulat yang tepat dalam rupiah bulat; floor-nya adalah pembagian bilangan bulat. Buku: Subbab 3.12.2, Persamaan (3.9)–(3.10), dan Gambar 3.20.

`Service`: Berat Tertagih dan Harga

Kode 3.30 `Service.java`, dua metode yang menghargai satu kantong: berat tertagih dan harganya.

```
    /** 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;
    }
```

*Pada **Kode 3.30** `price` menerima sebuah `Customer` dan menanyakan `isMember()` — objek-objek bekerja sama — serta mengalikan sebelum membagi dalam `total * 3 / 2`. Buku: Subbab 3.12.3 dan Kode 3.54.*

Dua Metode yang Sama dalam Python

Kode 3.31 `service.py`, `chargeable_kg` dan `price`: `math.ceil`, dan `//` di mana pun Java membagi bilangan bulat.

```
    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
```

`int(...)` alih-alih `round` Python, yang membulatkan setengah ke genap; $c \cdot r$ selalu bulat di sini, jadi tidak ada yang hilang. Keluaran Python identik, karakter demi karakter.

Kode 3.31 *melengkapi pasangannya;* `Customer` *dalam kedua bahasa adalah Kode 3.53 dan 3.56. Buku: Subbab 3.12.3 dan Kode 3.57.*

Empat Kantong, Diperiksa dengan Tangan

```
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))
```

Tabel 3.13 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

***Tabel 3.13**: 2 kg milik Budi ditagih 3 kg; 5,75 kg menjadi 6 kg; keempat harga berjumlah 178 950. Program demonstrasinya adalah Kode 3.55. Buku: Subbab 3.12.4 dan Tabel 3.14.*

Profitina dalam Praktik

Kelas untuk benda-benda domain: setiap jenis sinyal yang diperiksa pemindai adalah kelasnya sendiri yang menilai dan menjelaskan dirinya.

***String* di mana-mana:** halaman web adalah teks; penyebutan merek, `robots.txt`, schema.org — semuanya membandingkan *isi*, tidak pernah identitas (**Tabel 3.3**).

Bilangan dengan cermat: skor dibandingkan dengan toleransi Persamaan (3.3); jumlah koin PROFIT sebagai satuan terkecil yang bulat, seperti rupiah.

Data pribadi: `maskedPhone` mencerminkan UU PDP 27/2022 — olah hanya yang diperlukan tujuannya. **Pengujian sebagai gerbang**, lebih dari seribu, seperti `DogCheck`.

Platform menyarankan; penggunanya yang memutuskan. Buku: Subbab 3.13.

Kesalahan yang Sering Terjadi

`name = name;` alih-alih `this.name = name;` · `==` pada *string* Java · `name.trim();` saja · mengira `Dog b = a;` menyalin · memanggil melalui `null`.

Luapan di tengah perhitungan — *cast sebelum*: `(long) x * y` · desimal dibandingkan dengan `==` · `1 / 16` bernilai 0 · uang dalam `double`.

`break` yang hilang · rantai `else if` yang urutannya salah · gelung `while` yang tak pernah berhenti · batas *off-by-one*.

`sys.argv[0]` adalah nama skrip · komentar yang mengulang kode.

Buku: Subbab 3.14.

Ringkasan — Lima Hal untuk Diingat

1. Kelas adalah cetak biru; objek dibangun dengan `new` (atau dengan memanggil kelasnya), masing-masing dengan *field*-nya sendiri; `this`/`self` adalah objek saat ini.

2. Variabel primitif menyimpan nilai, variabel referensi menyimpan panah; identitas adalah `==`/`is`, kesamaan `equals`/`==`. Gambarlah diagram objeknya.

3. *String* bersifat imutabel dan berindeks dari 0; Java bertipe statis, Python dinamis tetapi kuat.

4. Luapan (Persamaan (3.2)), *floating point* (bandingkan menurut Persamaan (3.3)), pembagian bilangan bulat — tiga jebakan, tiga kebiasaan.

5. `if`/`switch`/`match`, `for`/`while`/for-each; tabel penelusuran menunjukkan persis apa yang dilakukan gelung. `Customer` dan `Service` menghargai kantong menurut Persamaan (3.9) dan Persamaan (3.10).

Buku: Subbab 3.15.

Pertemuan Berikutnya

Bab 4 — Evaluasi dan Pembahasan I: Dari Spesifikasi Sederhana ke Kelas yang Teruji.

Tanpa fitur baru: empat belas soal utuh atas Bab 1–3, masing-masing diselesaikan perlahan dengan delapan langkah yang sama, dalam kedua bahasa, beserta pengujiannya.

Pecahan, jam, akun, bilangan kompleks, berat, pertarungan, rupiah, waktu, struk, nama, kata sandi, gelung yang ditelusuri, dan tahun kabisat.

Sebelum kita bertemu: Soal Latihan 3.2, 3.5, 3.6, dan 3.10 — referensi, ekspresi, gelung yang ditelusuri, dan pecahan uang dengan pembagian dan sisa. Penyelesaiannya ada di Subbab 3.17.

Buku: Subbab 3.16–3.17, lalu Bab 4.

Pemrograman Berorientasi Objek (EF234302)

Kuliah 4 — Evaluasi dan Pembahasan I: Dari Spesifikasi Sederhana ke Kelas yang Teruji

Buku: Irfan Subakti, *Pemrograman Berorientasi Objek dengan Java dan Python*, Bab 4

Dosen: Irfan Subakti — `yifana@gmail.com`

Departemen Teknik Informatika, Institut Teknologi Sepuluh Nopember, Surabaya

Tidak ada fitur bahasa baru hari ini: empat belas soal utuh atas Bab 1–3, masing-masing diselesaikan dari nol, dalam kedua bahasa, beserta pengujiannya. Buku: Bab 4.

Agenda

1. Metode untuk soal pemrograman kecil apa pun: delapan langkah, uji lebih dulu, dan susunan setiap soal.

2. Tujuh soal klasik: kalkulator pecahan, sudut jarum jam, akun komputer, bilangan kompleks, berat, kelas pecahan, pemain dan musuh.

3. Tujuh soal sehari-hari: rupiah, waktu dalam sehari, nota *laundry*, nama yang rapi, aturan kata sandi, perulangan yang ditelusuri dengan tangan, tahun kabisat.

4. Untuk setiap soal: contoh dengan tangan, rancangan, Java dan Python, sebuah *penyelesaian keliru yang menggoda*, dan *uji diri*.

5. Keempat belas soal berdampingan, Profitina, kesalahan yang sering terjadi, ringkasan.

Sebelum melihat setiap penyelesaian, tutuplah dan cobalah sepuluh menit. Buku: Subbab 4.1–4.22.

Posisi Kita Saat Ini

Bab 1: meja kerja, dan gagasan pokoknya — objek membundel data dengan operasi yang diizinkan atasnya.

Bab 2: menyunting, menjalankan, dan men-*debug* — *breakpoint*, melangkah, *watch*. Beberapa penyelesaian keliru hari ini adalah bug yang ditemukan *debugger* dalam semenit.

Bab 3: kelas, *field*, konstruktor, metode; `int`, `long`, `double` beserta jebakannya; `String`, `equals`; `if`, `switch`, `while`, `for`.

> **WAWASAN KUNCI:** Menyebut perkakas yang diperlukan sebuah soal sudah separuh pekerjaan.

Buku: Subbab 4.1.

Empat Belas Soal, Satu Kotak Perkakas

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

Soal	Topik	Perkakas utama dari Bab 1–3
4.14	Tahun kabisat	ekspresi Boolean, `switch`, metode yang mengembalikan objek berikutnya

Tabel 4.1 *adalah peta: untuk perulangan, mulai dari 4.11–4.13; untuk kelas, dari 4.3 dan 4.5. Buku: Subbab 4.1 dan Tabel 4.1.*

Cara Menggarap Soal Pemrograman



Gambar 4.1 Delapan langkah yang dipakai untuk setiap soal dalam bab ini; panah putus-putus adalah putaran antara menulis kode dan menjalankan uji.

1 nyatakan ulang dengan kata-kata sendiri · **2** kerjakan dua atau tiga contoh dengan tangan · **3** kata benda → kelas dan *field*, kata kerja → metode · **4** gambar kotak kelasnya

5 tulis ujinya lebih dulu · **6** kodekan sedikit demi sedikit · **7** jalankan, bandingkan, *debug* · **8** tengok ke belakang: masukan yang belum diuji, cara keliru yang menggoda, sebuah varian

Gambar 4.1 *merentangkan empat gerakan Pólya — pahami, rencanakan, laksanakan, tengok kembali — menjadi delapan langkah; panah putus-putus adalah putaran antara kode dan uji. Buku: Subbab 4.2–4.2.1 dan Gambar 4.1.*

Uji, dan Bentuk Setiap Soal

Java: *driver* uji sederhana dengan `check(label, expected, actual)` yang mencetak PASS atau FAIL — membandingkan dengan `equals`, persis seperti `assertEquals` di JUnit.

Python: `unittest` dari pustaka standar; `python3 file.py` menjalankan demonstrasi lalu ujinya.

Setiap soal: soalnya · dengan tangan · kelas, *field*, dan metode · Java · Python · baris demi baris · penyelesaian keliru yang menggoda · uji diri.

> **MENGAPA CONTOH DENGAN TANGAN DIDAHULUKAN.** Program hanya dapat sebenar pemahaman Anda atas soalnya. Setiap nilai harapan dalam uji hari ini lebih dulu dihitung dengan tangan.

Hanya baris `Ran 4 tests in 0.000s` yang boleh berbeda di komputer Anda. Buku: Subbab 4.2.2–4.2.3.

Soal 4.1 — Kalkulator Pecahan

SOAL 4.1. Dengan `e1`, `d1` dan `e2`, `d2` mewakili $\frac{e_1}{d_1}$ dan $\frac{e_2}{d_2}$, tulislah `Q11Fraction` yang menghitung `es`, `ds` serta `ep`, `dp` untuk jumlah dan hasil kali, memakai Persamaan (4.1) dan Persamaan (4.2). Ujilah $\frac{1}{2}+\frac{1}{3}$, $\frac{1}{3}+\frac{3}{4}$, $\frac{1}{2}\cdot\frac{2}{3}$, $\frac{1}{4}\cdot\frac{2}{3}$.

$$\frac{e_1}{d_1}+\frac{e_2}{d_2}=\frac{e_1d_2+e_2d_1}{d_1d_2} \tag{4.1}$$

$$\frac{e_1}{d_1}\cdot\frac{e_2}{d_2}=\frac{e_1e_2}{d_1d_2} \tag{4.2}$$

Dengan tangan: $1\cdot3+1\cdot2=5$ per 6; $4+9=13$ per 12; $\frac{2}{6}$; $\frac{2}{12}$ — tidak disederhanakan, sesuai permintaan. Tepi: $-\frac{1}{2}+\frac{1}{3}=\frac{-1}{6}$.

Jawabannya sepasang bilangan bulat, bukan 0.8333*. Buku: Subbab 4.3.1–4.3.2 dan Persamaan (4.1)–(4.2).*

4.1: Satu Rumus per Metode

Kode 4.1 `Q11Fraction.java`, keempat rumusnya: metode statis, satu untuk setiap bilangan yang dihitung.

```
    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;
    }
```

```
1/2 + 1/3 = 5/6
1/3 + 3/4 = 13/12
1/2 * 2/3 = 2/6
1/4 * 2/3 = 2/12
```

***Kode 4.1**: metode static, karena belum ada yang diingat; Soal 4.6 menyembuhkan bilangan-bilangan lepas itu. Buku: Subbab 4.3.3–4.3.6 dan Kode 4.1–4.3.*

4.1: Penyelesaian Keliru yang Menggoda

Kode 4.2 `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));
    }
}
```

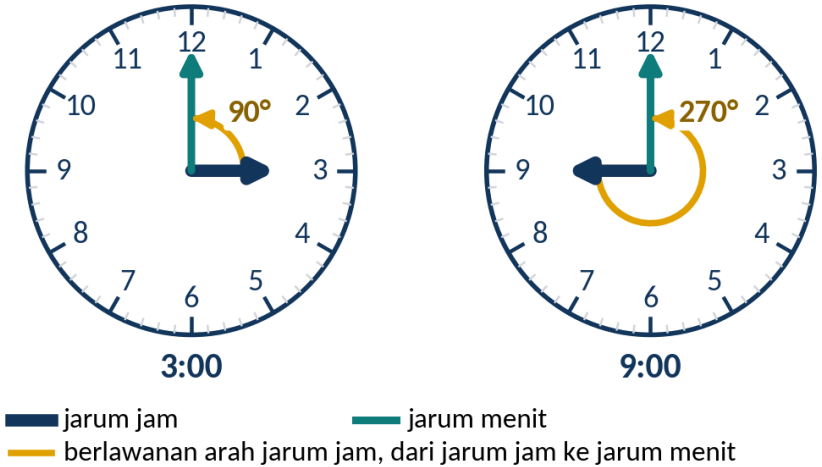
```
tops and bottoms added: 2/5
int division:           0
double division:        0.8333333333333333
```

UJI DIRI. `printDifference(1, 3, 3, 4)`? — $\frac{e_1 d_2 - e_2 d_1}{d_1 d_2}$: $4 - 9 = -5$ per 12, jadi `1/3 - 3/4 = -5/12`.

Kode 4.2*: menjumlahkan pembilang dan penyebut begitu saja salah;* `1 / 2` *bernilai* `0` *dalam* `int`*;* `double` *mendekati tetapi bukan yang diminta. Debugger langsung memperlihatkan yang kedua. Buku: Subbab 4.3.7 dan Kode 4.4.*

Soal 4.2 — Sudut Jarum Jam Analog

SOAL 4.2. Diberikan `hours` (0–23) dan `minutes` (0–59), `Q12Time` menghitung sudut **berlawanan arah jarum jam dari jarum jam ke jarum menit**, dibulatkan dan dinormalkan ke 0–359. Uji 9:00, 3:00, 18:00, 1:00, 2:30, dan 4:41: 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°.

***Gambar 4.2**: pukul 3:00 sudutnya 90°, pukul 9:00 270°. Satu menit 6°, satu jam 30°. Buku: Subbab 4.4.1 dan Gambar 4.2.*

4.2: Memahaminya dengan Tangan

$$\theta_{\text{jam}} = 30\left(h \bmod 12 + \frac{m}{60}\right) \tag{4.3}$$

$$\theta_{\text{menit}} = 6m \tag{4.4}$$

$$\alpha = \text{round}((\theta_{\text{jam}} - \theta_{\text{menit}}) \bmod 360) \bmod 360 \tag{4.5}$$

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

Jarum jam merayap (Persamaan (4.3)); berlawanan arah jarum jam berarti jam dikurangi menit; normalkan, lalu bulatkan (Persamaan (4.4), Persamaan (4.5)). ***Tabel 4.2****: pukul 4:41 sudut tepatnya* 254.5° *dan jawabannya 254 —* ***pembulatan setengah ke genap****. Buku: Subbab 4.4.2 dan Tabel 4.2.*

4.2: Normalkan, lalu Bulatkan

Kode 4.3 `Q12Time.java`, metode `angle`: Java memerlukan `+ 360` dan `Math.rint`.

```java
    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
    }
```

Kode 4.4 `q12_time.py`, fungsi `angle`: `%` dan `round` Python sudah berperilaku sesuai kebutuhan.

```python
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
```

```
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.
```

*Di Java `-105.0 % 360` adalah `-105.0`, maka **Kode 4.3** menambah 360; di Python `%` dengan pembagi positif tidak pernah negatif (**Kode 4.4**). `minutes / 60.0` menghindari pembagian bilangan bulat; pada 2:11,* 359.5 *dibulatkan menjadi 360, dan `% 360` menjadikannya 0. Buku: Subbab 4.4.4–4.4.6 dan Kode 4.6–4.8.*

4.2: Tiga Kekeliruan, dan Satu Jawaban Kebetulan

```
difference:         -105.5
no +360:            -105
Math.round:         255
minutes / 60 int:   234
difference:         -105.5
round, then %:      254
round half up:      255
minutes // 60:      234
```

Java: tanpa `+ 360` hasilnya -105; `Math.round` membulatkan -105.5 *ke atas* dan memberi 255; `minutes / 60` membekukan jarum jam: 234.

`round(-105.5)` Python adalah -106 (genap), dan $-106 \bmod 360 = 254$ — benar, tetapi *kebetulan*: ia membulatkan sebelum menormalkan.

> **UJI DIRI.** Searah jarum jam pada 4:41? $246 - 140.5 = 105.5$, setengah ke genap: 106. Pukul 3:00: 270. (Periksa: $254.5 + 105.5 = 360$.)

Program yang benar karena kebetulan akan salah setelah perubahan kecil berikutnya. Buku: Subbab 4.4.7 dan Kode 4.9–4.10.

Soal 4.3 — Akun Komputer

SOAL 4.3. `Q13ComputerAccount` dibangun dari `realName`, `userName`, dan `password`, dengan `printRealName`, `printUserName`, `printPassword`, dan `changePassword(String)`; ujilah di `Q13ComputerAccountTest`.

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`

*Objek pertama yang mengingat. Baris terakhir **Tabel 4.3** adalah inti objek: satu kelas, banyak objek yang saling bebas. Buku: Subbab 4.5.1–4.5.2 dan Tabel 4.3.*

4.3: *Field* Privat, Pemeriksaan yang Terkendali

Kode 4.5 `Q13ComputerAccount.java`, bagian 1: tiga *field* privat dan konstruktornya.

```
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;
    }
```

Kode 4.6 `Q13ComputerAccount.java`, bagian 2: `hasPassword`, cara terkendali untuk memeriksa kata sandi.

```
    // Menjawab "inikah kata sandinya?" tanpa pernah menyerahkan kata sandi itu.
    public boolean hasPassword(String attempt) {
        return password.equals(attempt);
    }
```

*Pada **Kode 4.5**, lupa `this.` membuat field-nya tetap `null`. Sengaja tidak ada `getPassword`: **Kode 4.6** menjawab dengan `equals`. Buku: Subbab 4.5.3–4.5.6 dan Kode 4.11–4.13.*

4.3: `==` Bukanlah `equals`

Kode 4.7 `WrongPasswordCheck.java`: `==` membandingkan identitas objek, `equals` membandingkan karakter.

```java
/** 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
```

> **UJI DIRI.** `changePassword(old, new)` yang mengembalikan `boolean`: `("wrong", "x")` mengembalikan `false`; `("sesame", "x")` mengembalikan `true`.

*Pada **Kode 4.7** pengguna yang mengetik kata sandi yang benar justru ditolak; `==` tampak berhasil pada literal, sehingga bug ini bersembunyi dalam uji cepat. Python: `is` adalah identitas, `==` isi. Buku: Subbab 4.5.7 dan Kode 4.14.*

Soal 4.4 — Bilangan Kompleks

SOAL 4.4. `Q14ComplexNumber` dengan konstruktor dan *getter*, serta metode untuk menjumlahkan, mengurangkan, dan mengalikan dua bilangan kompleks dan menuliskannya sebagai `"a + bi"`, memakai Persamaan (4.6) untuk hasil kali.

$$(a + bi)(c + di) = (ac - bd) + (ad + bc)i \tag{4.6}$$

$c_1 = 1 + 2i$, $c_2 = 3 + 4i$: jumlah $4 + 6i$; selisih $-2 - 2i$; hasil kali $1 \cdot 3 - 2 \cdot 4 = -5$ dan $1 \cdot 4 + 2 \cdot 3 = 10$, jadi $-5 + 10i$. Tepi: $i \cdot i = -1 + 0i$.

Keputusan rancangan: `add` menerima bilangan kompleks *lain* dan mengembalikan yang *baru*, seperti $c_1 + c_2$ dalam matematika. Kita mencetak `-2.0 - 2.0i`, bukan `-2.0 + -2.0i`.

Buku: Subbab 4.6.1–4.6.3 dan Persamaan (4.6).

4.4: Operasi yang Mengembalikan Objek Baru

Kode 4.8 `Q14ComplexNumber.java`, ketiga operasinya: masing-masing mengembalikan objek baru.

```
    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);
    }
```

Kode 4.9 `q14_complex_number.py`, `__str__`: tanda minus dan bagian positifnya bila bagian imajinernya negatif.

```
    def __str__(self):
        if self._imaginary < 0:
            return f"{self._real} - {-self._imaginary}i"
        return f"{self._real} + {self._imaginary}i"
```

*Pada **Kode 4.8**, `other.real` diizinkan: privasi milik kelas. **Kode 4.9** menyamai `toString` Java. Buku: Subbab 4.6.4–4.6.6 dan Kode 4.16–4.18.*

4.4: Penyelesaian Keliru yang Menggoda

Kode 4.10 `WrongComplex.java`: mengalikan bagian demi bagian, dan mencetak bagian negatif secara naif.

```java
/** 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
```

UJI DIRI. `c1.multiply(c1.conjugate())` untuk $1 + 2i$? — `5.0 + 0.0i`, karena $(a + bi)(a - bi) = a^2 + b^2$.

*Mengalikan bagian demi bagian pada **Kode 4.10** melupakan suku silang dan $i \cdot i = -1$. Buku: Subbab 4.6.7 dan Kode 4.19.*

Soal 4.5 — Konversi Berat

SOAL 4.5. `Q15Weight(double p)` membuat berat dalam *pound*, dengan `getPounds` dan `getKilograms`, memakai Persamaan (4.7); ujilah di `Q15WeightTest`.

$$1 \text{ pon} = 0.45359237 \text{ kilogram} \tag{4.7}$$

Dengan tangan: $150 \times 0.45359237 = 68.0388555$; $140 \times 0.45359237 = 63.5029318$; 11 st 6 lb $= 160$ lb $= 72.5747792$ kg.

Simpan *pound* dan kilogram sekaligus? Jangan: metode kelak bisa mengubah yang satu dan melupakan yang lain, dan objeknya akan bertentangan dengan dirinya sendiri.

Faktor pada Persamaan (4.7) tepat menurut kesepakatan internasional, jadi kedelapan digitnya ditulis. Buku: Subbab 4.7.1–4.7.3 dan Persamaan (4.7).

4.5: Simpan Sekali, Turunkan Sisanya

Kode 4.11 `Q15Weight.java`: satu konstanta, satu *field*, sebuah konstruktor, dan dua *getter*.

```java
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 4.11 *hanya menyimpan pound dan menghitung kilogram pada setiap pemanggilan, dengan biaya satu perkalian.* `static final` *berarti satu salinan, tak pernah berubah. Buku: Subbab 4.7.4–4.7.6 dan Kode 4.21–4.23.*

4.5: Toleransi, dan Nilai yang Menjadi Basi

```
John: 150.0 lb = 68.0388555 kg
Mary: 140.0 lb = 63.502931800000006 kg
11 st 6 lb: 160.0 lb = 72.57477920000001 kg
```

Kode 4.12 `WrongWeight.java`, *setter*-nya: ia melupakan *field* kedua.

```
    public void setPounds(double p) {
        pounds = p; // lupa memperbarui kilogram
    }
```

```
140.0 lb = 68.0388555 kg ?
```

*`63.502931800000006` tidak salah: bandingkan dengan `Math.abs(expected - actual) < 1e-9` (Python `assertAlmostEqual`). **Kode 4.12** mengklaim 140 lb sama dengan 68,04 kg. Simpan setiap fakta sekali, dan hitung segala yang mengikutinya. Buku: Subbab 4.7.4–4.7.7 dan Kode 4.24.*

Soal 4.6 — Kelas Pecahan

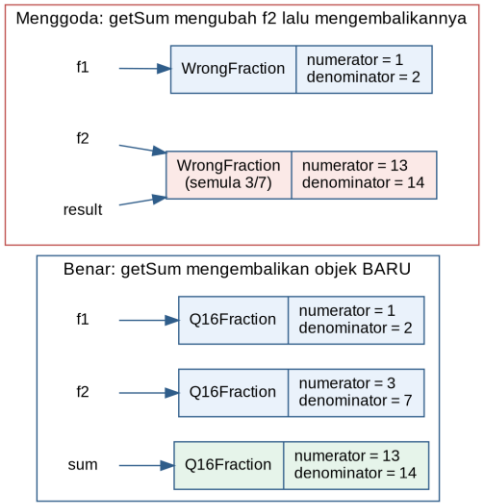
SOAL 4.6. `Q16Fraction` dengan `getNumerator`, `getDenominator`, `toString`, serta `getSum` dan `getProduct` yang mengembalikan `Q16Fraction`. Dengan $f_1 = \frac{1}{2}$ dan $f_2 = \frac{3}{7}$: `f1.toString()` adalah `"1/2"`, `f2.getProduct(f1)` mencetak `3/14`, `f2.getSum(f1)` mencetak `13/14`.

Dengan tangan: $\frac{3 \cdot 1}{7 \cdot 2} = \frac{3}{14}$; $\frac{3 \cdot 2 + 7 \cdot 1}{14} = \frac{13}{14}$; dan sesudahnya f_2 harus tetap $\frac{3}{7}$.

Berantai: $(f_1 + f_1) \cdot (f_2 + f_2) = \frac{4}{4} \cdot \frac{42}{49} = \frac{168}{196}$, yang tak lain adalah $\frac{6}{7}$.

Soal 4.1 lagi, tetapi setiap pecahan kini satu objek. Buku: Subbab 4.8.1–4.8.2.

4.6: Objek Baru, atau Objek yang Diubah?



Gambar 4.3 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.

***Gambar 4.3**, kotak bawah: `getSum` yang benar mengembalikan objek ketiga. Objek yang tidak pernah berubah disebut imutabel — seperti `String` Java. Buku: Subbab 4.8.3 dan Gambar 4.4.*

4.6: Pecahan yang Imutabel

Kode 4.13 `Q16Fraction.java`, kedua operasinya: masing-masing membangun dan mengembalikan pecahan baru.

```
    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 4.14 `q16_fraction.py`, kedua operasi yang sama dalam Python.

```
    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())
```

*Tanpa setter, **Kode 4.13** menjadi imutabel; **Kode 4.14** menerjemahkannya baris demi baris. Driver-nya mencetak `1/2`, `3/14`, `13/14`. Buku: Subbab 4.8.4–4.8.6 dan Kode 4.26–4.28.*

4.6: `getSum` yang Mengubah `this`

Kode 4.15 `WrongFraction.java`, `getSum` yang menggoda: ia menimpa *field*-nya sendiri dan mengembalikan `this`.

```
    public WrongFraction getSum(WrongFraction other) {
        numerator = numerator * other.denominator + denominator * other.numerator;
        denominator = denominator * other.denominator;
        return this;
    }
```

```
f2 + f1 = 13/14
f2 is now 13/14
f2 + f1 = 40/28 (same call, new answer)
```

> **UJI DIRI.** Sederhanakan di konstruktor dengan FPB Euklides: `new Q16Fraction(2, 6)` mencetak `1/3`, dan `f2.getSum(f2)` mencetak `6/7`.

*Pemanggilan yang sama, `f2.getSum(f1)`, memberi jawaban baru pada kali kedua: kotak atas **Gambar 4.3**. **Kode 4.15** menjebak setiap pemakainya. Buku: Subbab 4.8.7 dan Kode 4.29.*

Soal 4.7 — Pemain dan Musuh

SOAL 4.7. `Player` dan `Enemy`, masing-masing dengan nama, kesehatan, kekuatan, dan pertahanan, konstruktor, *getter*, dan *setter*; `attack(lawan)` membuat lawan memanggil `takeDamage(attack)`, yang mengurangi kesehatan sebesar (serangan − pertahanan) dan mencetak `"{name} died!"` di bawah nol.

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!”

*Setiap objek bertanggung jawab atas field-nya sendiri. Soalnya diam soal kerusakan negatif (kita anggap nol) dan agak meleset pada kesehatan tepat 0 (kita anggap mati): tuliskan keputusan seperti itu (**Tabel 4.4**). Buku: Subbab 4.9.1–4.9.3 dan Tabel 4.4.*

4.7: Satu Pesan, dan yang Diserang yang Memutuskan

Kode 4.16 `Player.java`, pertarungannya: `attack` mengirim satu pesan; `takeDamage` menghitung kerusakan dari pertahanannya sendiri.

```
    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!");
        }
    }
```

```
Round 1: Villain has 40 health left
Round 2: Villain has 20 health left
Villain died!
Round 3: Villain has 0 health left
```

Kurangi dulu, baru periksa. `Villain died!` *muncul di dalam serangan ketiga, sebelum* `Round 3` *dicetak: urutan keluaran mencatat urutan pemanggilan.* `Enemy.java` *adalah bayangan cermin* ***Kode 4.16****; Bab 7 membuat keduanya berbagi kode. Buku: Subbab 4.9.4–4.9.6 dan Kode 4.31–4.33.*

4.7: Menjangkau ke Dalam Objek Lain

Kode 4.17 `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());
    }
}
```

```
Cheater.java:5: error: health has private access in Enemy
        villain.health = 0;
               ^
1 error
```

> **UJI DIRI.** Tambahkan `isAlive()` dan biarkan tokoh yang sudah mati mengabaikan serangan: satu `hero.attack(villain)` lagi tidak mencetak apa pun dan kesehatannya tetap 0.

*Galat kompilasi untuk **Kode 4.17** adalah hal baik. `villain._health = 0` di Python lolos diam-diam — dan tidak ada "died!", karena satu-satunya metode yang tahu soal kematian dilewati. Buku: Subbab 4.9.7 dan Kode 4.34–4.35.*

Soal 4.8 — Uang dalam Rupiah

SOAL 4.8. `Rupiah` menyimpan rupiah bulat, dengan `getAmount`, serta `plus`, `times`, dan `discount(percent)` (dibulatkan ke bawah) yang mengembalikan objek baru; `toString` mengelompokkan digit dari kanan: `Rp1.250.000`, `Rp999`, `-Rp5.000`.

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`

Uang adalah `long`, tidak pernah `double`. $4 \cdot 7000 + 12500 = 40500$*; diskon 10% adalah* $\frac{40500 \cdot 90}{100} = 36450$*; kalikan sebelum membagi.* ***Tabel 4.5*** *mengelompokkan 1250000. Buku: Subbab 4.10.1–4.10.2 dan Tabel 4.5.*

4.8: Mengelompokkan dari Kanan

Kode 4.18 `Rupiah.java`, `toString`: prosedur penelusuran tangan ditulis sebagai gelung yang menghitung mundur.

```
    @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;
    }
```

```
Order: Rp40.500
After 10% off: Rp36.450
```

*`count % 3 == 0 && i > 0`: tanpa `i > 0`, `999` menjadi `.999`. Python menelusuri `range(len(digits) - 1, -1, -1)` dengan cara yang sama. **Kode 4.18** imutabel seperti `Q16Fraction`. Buku: Subbab 4.10.3–4.10.6 dan Kode 4.36–4.38.*

4.8: Dari Kiri, dan dalam `double`

Kode 4.19 `wrong_rupiah_format.py`: dua kesalahan yang sama dalam Python.

```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.** Rp100.000 dibagi 3 orang: masing-masing Rp33.333 dan sisa Rp1 — tanpa `remainder`, satu rupiah lenyap diam-diam.

Kelompok tiga digit milik ribuan dan jutaan, dihitung dari kanan; 1.1 *tidak punya bentuk biner yang tepat.* ***Kode 4.19*** *memperlihatkan kedua kesalahan; versi Java mencetak hal yang sama. Buku: Subbab 4.10.7 dan Kode 4.40.*

Soal 4.9 — Waktu dalam Sehari

> **SOAL 4.9.** `TimeOfDay(hours, minutes)` pada jam 24, dengan *getter*, `plusMinutes(m)` yang melingkar melewati tengah malam ke dua arah, `isBefore`, `minutesUntil`, dan `toString` seperti `08:05`.

22:30 + 45 menit = 23:15 — menitnya *disimpan* ke jam. 22:30 + 150 menit = 01:00 — jamnya *melingkar*. 00:10 − 20 menit = 23:50.

Satu gagasan: simpan satu bilangan, menit sejak tengah malam (0–1439). $22 \cdot 60 + 30 = 1350$; $+150 = 1500$; $1500 \bmod 1440 = 60$, yaitu 01:00. Dan -10 melingkar menjadi 1430: 23:50.

Dunia luar berbicara dalam jam dan menit; bagian dalam memakai representasi yang memudahkan aritmetika. Memilihnya, lalu menyembunyikannya, itulah buah enkapsulasi. Buku: Subbab 4.11.1–4.11.3.

4.9: Satu Bilangan Internal, Tersembunyi

Kode 4.20 `TimeOfDay.java`, intinya: konstruktor, `wrap` yang privat, dan aritmetika atas menit.

```
    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);
    }
```

Kode 4.20 *mencetak* `Dropped off at 22:30, ready at 01:00`*;* `%` *Python tidak memerlukan* `+ 1440`*. Buku: Subbab 4.11.4–4.11.6 dan Kode 4.41–4.43.*

4.9: Tanpa Simpanan, Tanpa Lingkaran, Tanpa Nol di Depan

Kode 4.21 `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);
    }
}
```

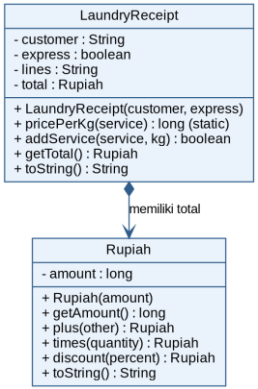
```
no carry:   22:75
no wrap:    25:0
no padding: 8:5
```

> **UJI DIRI.** Buka 07:00–21:00: 06:59 tutup, 07:00 buka, 20:59 buka, 21:00 tutup — menit pembukaan disertakan, menit penutupan tidak.

*Setiap baris **Kode 4.21** adalah bug sungguhan dari sistem pemesanan sungguhan; ketiganya lenyap dengan satu bilangan yang selalu dijaga dalam rentang. Buku: Subbab 4.11.7 dan Kode 4.44.*

Soal 4.10 — Nota *Laundry*

SOAL 4.10. `LaundryReceipt(customer, express)`; `addService(service, kg)` menambah satu baris — wash Rp7.000, iron Rp5.000, wash+iron Rp10.000 per kilogram *yang dimulai*, ekspres +50% — atau mengembalikan `false` untuk layanan yang tak dikenal atau berat yang tidak positif. Totalnya sebuah `Rupiah`.



Gambar 4.4 Diagram kelas Soal 4.10: sebuah `LaundryReceipt` memiliki total bertipe `Rupiah`; belah ketupat terisi menandai bahwa total itu milik nota.

Sari: $4 \cdot 7000 + 2 \cdot 5000 = 38000$*; Budi:* $50000 + 25000 = 75000$*.* ***Gambar 4.4****: nota memiliki sebuah* `Rupiah` *—* ***komposisi****. Buku: Subbab 4.12.1–4.12.3 dan Gambar 4.5.*

4.10: Tolak Lebih Awal, Tagih dengan *Ceiling*

Kode 4.22 `LaundryReceipt.java`, `addService`: tolak lebih awal, tagih per kilogram yang dimulai, jumlahkan dalam `Rupiah`.

```
    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);
        // 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;
    }
```

*`pricePerKg` adalah `switch` atas string, yang membandingkan dengan `equals`. `Locale.ROOT` menjaga `3.2` agar tidak menjadi `3,2` di komputer berbahasa Indonesia. **Kode 4.22** dan kembaran Python-nya mencetak nota yang sama. Buku: Subbab 4.12.4–4.12.6 dan Kode 4.46–4.48.*

4.10: Notanya, dan Tiga Cara Salah Memperlakukan 3,2

```
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
int(kg):       3 kg -> Rp21000
round(kg):     3 kg
round(2.5):    2 kg
math.ceil(kg): 4 kg
```

UJI DIRI. Biaya minimum Rp20.000 tempatnya di `getTotal()`; 1,5 kg setrika berharga Rp10.000, jadi totalnya Rp20.000.

`int(3.2)` merugikan Rp7.000 pada satu kantong; `round` juga memberi 3, dan `round(2.5)` adalah 2 (setengah ke genap). "Per kilogram yang dimulai" berarti ceiling. Buku: Subbab 4.12.7 dan Kode 4.49–4.50.

Soal 4.11 — Merapikan Nama

SOAL 4.11. `PersonName` menyimpan teks mentah `"  siti  NUR aisyah "` dan menawarkan `wordCount()`, `initials()` (`"S.N.A."`), dan `tidy()` (`"Siti Nur Aisyah"`), hanya dengan `length`, `charAt`, dan metode karakter.

```
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?    ^              ^           ^
```

Sebuah karakter memulai kata bila ia bukan spasi dan merupakan karakter pertama atau didahului spasi: posisi 2, 9, dan 13. Kasus tepi: nama kosong, nama berisi tiga spasi. Buku: Subbab 4.13.1–4.13.2.

4.11: Satu Aturan Privat, Dipakai Tiga Kali

Kode 4.23 `PersonName.java`, aturannya: di mana sebuah kata dimulai.

```
    private boolean startsWord(int i) {
        return raw.charAt(i) != ' ' && (i == 0 || raw.charAt(i - 1) == ' ');
    }
```

Kode 4.24 `PersonName.java`, `tidy`: huruf besar di awal kata, huruf kecil di tempat lain, spasi dibuang.

```
        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 4.23 *bersandar pada* `||` *yang* ***short-circuit****;* ***Kode 4.24*** *menambah spasi sebelum setiap kata kecuali yang pertama. Buku: Subbab 4.13.3–4.13.6 dan Kode 4.51–4.53.*

4.11: `split(" ")` dan Kata Kosong

Kode 4.25 `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
```

*Pada **Kode 4.25** setiap spasi berlebih menghasilkan kata kosong `''`, dan `''[0]` gagal; Java berhenti dengan `StringIndexOutOfBoundsException`. Jalan pintasnya: Python `split()`, Java `trim().split("\\s+")`. Uji diri: `lastNameFirst()` memberi `"Aisyah, Siti Nur"`. Buku: Subbab 4.13.7 dan Kode 4.55.*

Soal 4.12 — Aturan Kata Sandi

> **SOAL 4.12.** `PasswordChecker.problems(password)` mendaftar setiap aturan yang dilanggar, dipisah `"; "`: sedikitnya 8 karakter, sebuah digit, sebuah huruf kapital, sebuah huruf kecil; `isStrong` bernilai benar tepat ketika tidak ada pelanggaran.

`sesame` → `too short; no digit; no capital letter`. `1sesame` → `too short; no capital letter`. `Sesame2026` → kuat. Batas: `Abcdefg1` (8) kuat, `Abcdef1` (7) terlalu pendek.

"Ada karakter yang berupa digit" dapat diputuskan pada digit pertama; "tidak ada karakter kapital" baru setelah karakter *terakhir*.

Pedoman modern (NIST SP 800-63B) mengutamakan kata sandi panjang yang dicocokkan dengan daftar kata sandi umum, dan sistem nyata hanya menyimpan hash bergaram. Buku: Subbab 4.14.1–4.14.3.

4.12: Hitung di Dalam Gelung, Putuskan Sesudahnya

Kode 4.26 `PasswordChecker.java`, `problems`: satu gelung menghitung, pernyataan `if` sesudahnya memutuskan.

```
        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");
        }
```

*Pada **Kode 4.26**, rantai `else if` karena satu karakter paling banyak satu jenis; pembantu `add` menaruh `"; "` di antara pelanggaran; `isStrong` didefinisikan melalui `problems`, sehingga keduanya tak pernah berselisih. Buku: Subbab 4.14.4–4.14.6 dan Kode 4.56–4.58.*

4.12: Vonis Setelah Satu Karakter

Kode 4.27 `WrongPassword.java`, `hasDigit`: `else` mengembalikan nilai di dalam gelung.

```
    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;
    }
```

```
1sesame:  true
Laundry7: false
```

UJI DIRI. Aturan kelima, `"no symbol"`: `Sesame2026` kini melaporkan `no symbol`; `Sesame-2026` kuat.

Kode 4.27 *hanya melihat karakter pertama — sebenarnya bukan gelung; telusuri* `hasDigit("Laundry7")` *dan lihat* `i` *tak pernah mencapai 1. Pindahkan* `return false` *ke sesudah gelung. Buku: Subbab 4.14.7 dan Kode 4.59.*

Soal 4.13 — Menelusuri Perulangan dengan Tangan

SOAL 4.13. Tanpa menjalankan program, telusuri (a) jumlah digit $n = 4071$ dan (b) algoritma Euklides untuk $a = 84$, $b = 36$; lalu tulis program yang jejaknya mereproduksi tabel Anda.

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

Tabel 4.6*: empat putaran, satu per digit;* $4 + 0 + 7 + 1 = 12$*. Untuk* $n = 0$ *gelungnya berjalan nol kali. Tabel penelusuran adalah versi kertas dari jendela watch debugger. Buku: Subbab 4.15.1–4.15.2 dan Tabel 4.6.*

4.13: Algoritma Euklides dengan Tangan

$$\gcd(a, b) = \gcd(b,\ a \bmod b), \qquad \gcd(a, 0) = a \tag{4.8}$$

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

$\gcd(84,36) = 12$: $84 = 7 \cdot 12$, $36 = 3 \cdot 12$. Mulai dengan $\gcd(36,84)$ hanya menukar kedua bilangan pada putaran pertama.

*Persamaan (4.8): apa pun yang membagi a dan b juga membagi a mod b; **Tabel 4.7** menerapkannya dua kali. Buku: Subbab 4.15.2, Persamaan (4.8), dan Tabel 4.7.*

4.13: Gelung yang Mencetak Jejaknya Sendiri

```
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
```

`n = n / 10` adalah pembagian bilangan bulat di Java; Python *harus* menulis `n // 10`. Dengan `/`, gelung Python berjalan 328 putaran dan menjumlahkan pecahan menjadi `13.33…`.

`while (n > 9)` terasa benar dan berhenti satu putaran terlalu cepat: 8 alih-alih 12 — tabelnya akan berbaris tiga, bukan empat.

*Baris-baris yang tercetak adalah **Tabel 4.6** dan **Tabel 4.7**. Uji diri: gcd(1071,462) perlu tiga putaran dan hasilnya 21. Buku: Subbab 4.15.4–4.15.7 dan Kode 4.61–4.65.*

Soal 4.14 — Tahun Kabisat dan Hari Berikutnya

SOAL 4.14. `SimpleDate` dengan `isLeapYear` dan `daysInMonth` yang statis, `isValid()`, `nextDay()` yang mengembalikan tanggal baru, dan `toString` seperti `2024-02-29`; hitung pula tahun kabisat dari 1901 sampai 2100.

$$\text{kabisat}(y) = (4 \mid y \ \wedge \ 100 \nmid y) \ \vee \ 400 \mid y \tag{4.9}$$

2024 kabisat, 2023 bukan, 1900 bukan ($100 \mid 1900$, $400 \nmid 1900$), 2000 kabisat. Dari 1901 sampai 2100: 50 kelipatan 4, dikurangi 2100: **49**.

`nextDay`: bukan hari terakhir → hari + 1; selain itu bukan Desember → tanggal 1 bulan berikutnya; selain itu 1 Januari tahun berikutnya.

Persamaan (4.9) adalah aturan Gregorian sebagai satu ekspresi Boolean. Buku: Subbab 4.16.1–4.16.3 dan Persamaan (4.9).

4.14: Aturan sebagai Satu Ekspresi Boolean

Kode 4.28 `SimpleDate.java`, fakta-fakta kalender: tahun kabisat, dan panjang bulan dengan `switch`.

```
    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;
        }
    }
```

*Fakta-fakta itu statis: milik kalender, bukan milik satu tanggal (**Kode 4.28**); beberapa label `case` berbagi satu badan. Buku: Subbab 4.16.4–4.16.6 dan Kode 4.66–4.68.*

4.14: Hari Berikutnya, dan Jebakan Julian

Kode 4.29 `simple_date.py`, `next_day`: tiga kasus, diperiksa berurutan, masing-masing mengembalikan tanggal baru.

```
    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)
```

```
1900 leap? true
Leap years from 1901 to 2100: 50
```

> **UJI DIRI.** `dayOfYear()` dengan gelung `for` atas bulan-bulan sebelumnya: 2024-03-01 adalah hari ke-61; 2023-12-31 hari ke-365.

Kode 4.29 *memberi 2024-02-29 sesudah 2024-02-28. "Habis dibagi 4" saja adalah aturan Julian, benar untuk 1901–2099 dan salah untuk 1900 dan 2100, seperti terlihat pada keluarannya. Buku: Subbab 4.16.7 dan Kode 4.69.*

Tujuh Soal Klasik Berdampingan

Tabel 4.8 Soal 4.1–4.7: 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

***Tabel 4.8**: setiap kelas yang mewakili sebuah nilai dibuat imutabel. Buku: Subbab 4.17 dan Tabel 4.8.*

Tujuh Soal Baru Berdampingan

Tabel 4.9 Soal 4.8–4.14: gagasan rancangan yang diajarkan masing-masing, dan jebakan yang diperagakan penyelesaian kelirunya.

Soal	Gagasan rancangan	Jebakan yang diperlihatkan penyelesaian keliru
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

***Tabel 4.9**: hampir setiap jebakan tak terlihat pada contoh yang “manis” dan baru tampak di tepi. Menguji tepi adalah menguji. Buku: Subbab 4.17 dan Tabel 4.8.*

Profitina dalam Praktik

Kelas kecil, masing-masing diuji tersendiri: setiap sinyal pemindai adalah satu kelas kecil; lebih dari seribu pengujian harus hijau sebelum rilis — PASS/FAIL dalam ukuran penuh.

Uang dan bilangan yang tepat: koin PROFIT, dengan enam desimal, ditangani sebagai satuan terkecil yang bulat, dan ditampilkan kepada pengguna Indonesia dengan titik pengelompokan.

Kata sandi tidak pernah dicetak: disimpan hanya sebagai *hash* Argon2id, diubah hanya dengan bukti kata sandi lama (UU PDP 27/2022).

Waktu dan tanggal: tren skor mingguan dan pemindaian ulang terjadwal menemui jebakan Soal 4.9 dan 4.14.

Skor memberi tahu pemilik merek di mana mencari; pemiliknya yang memutuskan apa yang diubah. Buku: Subbab 4.18.

Kesalahan yang Sering Terjadi

Membagi dua bilangan bulat dan mengharapkan pecahan · memercayai `%` dengan bilangan negatif di Java · `==` pada *string* · membulatkan dengan aturan yang salah.

Membandingkan `double` dengan `==` · menyimpan fakta yang sama dua kali · mengubah `this` di dalam `getSum` · menjangkau ke dalam objek lain.

Memutuskan di dalam gelung apa yang hanya dapat diputuskan di ujungnya · kondisi *off-by-one* · lupa `this.` di konstruktor.

Hanya menguji contoh dalam soal. Tambahkan nol, bilangan negatif, batas, *string* kosong, menit ganjil, tahun abad.

Setiap penyelesaian keliru hari ini lolos setidaknya satu contoh yang "manis". Buku: Subbab 4.19.

Ringkasan — Lima Hal untuk Diingat

1. Delapan langkah: nyatakan ulang, contoh dengan tangan, kata benda dan kata kerja, kotak kelas, uji lebih dulu, sedikit demi sedikit, jalankan dan *debug*, tengok kembali.

2. *Driver* Java dengan `check` dan `unittest` Python mengubah contoh tangan menjadi pemeriksaan otomatis; keduanya berpindah ke JUnit 6 secara mekanis.

3. Kelas nilai — pecahan, bilangan kompleks, uang, waktu, tanggal — bersifat imutabel; operasinya mengembalikan objek baru.

4. Simpan setiap fakta sekali; pilih representasi internal yang baik lalu sembunyikan; biarkan setiap objek menjaga *field*-nya sendiri.

5. Kuasai aritmetikanya: pembagian bilangan bulat, `%` Java, setengah ke genap (Persamaan (4.5)), *ceiling*, toleransi; hitung di dalam gelung, putuskan sesudahnya.

Buku: Subbab 4.20.

Pertemuan Berikutnya

Bab 5 — Larik, List, Masukan, Eksepsi, dan Berkas.

Nota yang dapat memuat baris sebanyak apa pun sebagai objek; masukan dari papan ketik; eksepsi yang dapat ditangkap, seperti `StringIndexOutOfBoundsException` pada Soal 4.11; berkas.

Sebelum kita bertemu: Soal Latihan 4.2, 4.5, 4.8, dan 4.12 — sudut jarum jam, `toString` yang ditelusuri, jumlah digit yang ditelusuri, dan delapan langkah secara mandiri. Penyelesaiannya ada di Subbab 4.22.

Buku: Subbab 4.21–4.22, lalu Bab 5.

Kuliah 5 — Larik, List, Masukan, Eksepsi, dan Berkas

Pemrograman Berorientasi Objek (EF234302)

Buku: Irfan Subakti, *Pemrograman Berorientasi Objek dengan Java dan Python*, Bab 5

Dosen: Irfan Subakti — yifana@gmail.com

Hari ini program kita berhenti beranggapan bahwa ia tahu berapa banyak benda yang akan ditanganinya, dan bahwa tidak akan ada yang salah. Buku: Bab 5, Subbab 5.1–5.12.

Cakupan Pertemuan Ini

1. **Larik** — sederet slot bernomor yang ukurannya tetap; indeks 0 sampai $n - 1$; referensi, salinan, dan `args`.
2. **List** — `ArrayList` dan `list` Python, operasi sehari-hari, cara list tumbuh, *boxing*.
3. **Masukan** — `Scanner` dan `input()`, sisa pindah baris, perulangan dengan *sentinel*.
4. **Eksepsi** — *stack trace*, `try`/`catch`/`finally`, `throw`, eksepsi terperiksa dan tak terperiksa, eksepsi buatan sendiri.
5. **Berkas** — *stream* dan *reader*, menutup apa yang dibuka, menulis teks, CSV.
6. **Profitina Laundry** — memuat pesanan dari berkas CSV yang rusak, menolak setiap baris buruk dengan tepat, menyimpan berkas yang bersih.

Urutannya mengikuti urutan ketergantungan gagasan-gagasannya: larik, list, masukan, eksepsi, berkas. Buku: Subbab 5.1.

Di Mana Kita Berhenti: Tiga Celah

Bab 3 memberi kita kelas, objek, string, tipe, percabangan, dan perulangan; Bab 4 membahas soal-soal dengan semua itu. Setiap variabel menyimpan **satu** nilai.

Banyak nilai. Pada suatu Senin binatu itu menerima empat pesanan, mungkin juga empat puluh; kita tidak dapat mendeklarasikan empat puluh variabel.

Masukan. Program yang hanya mencetak adalah pidato, bukan percakapan.

Kegagalan. Pelanggan mengetik “tiga” untuk 3; sebuah berkas dipindahkan atau disunting dengan tangan. Program yang menyambutnya dengan setumpuk teks merah barulah purwarupa, belum program yang selesai.

Eksepsi dan berkas memang sejalan: hampir segala sesuatu yang menyentuh berkas dapat gagal, dan Java bahkan tidak mau mengompilasi kode berkas sebelum kita menyatakan apa yang terjadi bila gagal. Buku: Subbab 5.1.

Larik: Rak Kotak Surat

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 $0,1,\ldots,n-1$.

Indeks adalah **jarak** dari awal: jika larik dimulai pada alamat b dan setiap slot selebar s bita, slot i dimulai pada alamat Persamaan (5.1).

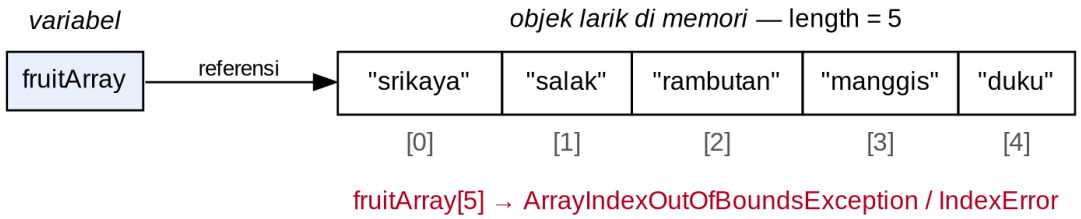
$$\text{alamat}(i) = b + i \cdot s \tag{5.1}$$

Satu perkalian dan satu penjumlahan mencapai slot mana pun, sepanjang apa pun lariknya: *akses acak*.

INGAT. Larik sepanjang n **tidak** mempunyai elemen berindeks n.

Kita sudah pernah berjumpa dengan larik: `String[] args` *pada setiap* `main`*. Buku: Subbab 5.2.1.*

Larik di dalam Memori



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.

*Pada **Gambar 5.1** variabel `fruitArray` tidak menyimpan kelima buah itu; ia menyimpan referensi ke objek larik, yang juga mencatat panjangnya. Buku: Subbab 5.2.1.*

Membuat dan Mengisi Larik

Kode 5.1 membangun lima slot `String` dengan `new String[5]` lalu mengisinya satu indeks demi satu indeks.

Kode 5.1 Dari `main` di `FruitArray.java`: `new`, nilai bawaan, penugasan menurut indeks, dan perulangan pencacah.

```
        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]);
        }
```

Larik baru tidak pernah kosong: setiap slot berisi nilai bawaan, `null` untuk objek, `0` untuk `int`, `false` untuk `boolean`. Tulislah selalu `i < fruitArray.length`, jangan angka 5 yang ditulis mati. Buku: Subbab 5.2.2.

Bentuk Singkat dan `for` yang Disempurnakan

Kode 5.2 adalah sisa `main`: bentuk kurung kurawal menghitung nilainya untuk kita, dan `for (String fruit : …)` mengunjungi setiap nilai tanpa indeks.

Kode 5.2 Sisa `main` di `FruitArray.java`: penginisialisasi singkat, `for` yang disempurnakan, dan larik `int`.

```
        // 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));
```

```
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]
```

`length` adalah field larik, tanpa tanda kurung; string memakai metode `length()`. `Arrays.toString` mengubah larik menjadi teks yang mudah dibaca. Buku: Subbab 5.2.2.

Rak yang Sama di Python

Python tidak mempunyai larik berukuran tetap di dalam bahasanya; **Kode 5.3** menirunya dengan list berisi lima `None`.

Kode 5.3 Paruh pertama `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])
```

`len(x)` memerankan `.length`; `range(n)` menghasilkan $0, 1, \ldots, n-1$*; Python tidak mengisi list dengan nol kecuali kita menulis `[0] * 10`. Keluarannya sama dengan keluaran Java baris demi baris. Buku: Subbab 5.2.2.*

Melangkah Melewati Ujung

Kode 5.4 menulis perulangan selisih-satu yang klasik: `<=` di tempat yang semestinya `<`.

Kode 5.4 `ArrayBounds.java`: perulangan yang berjalan satu langkah terlalu jauh dan menulis melewati ujung larik lima elemen.

```java
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 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)
```

Java melempar eksepsi, menyebut barisnya (baris 8), dan meninggalkan sisa `main`. Python melempar `IndexError` di tempat yang sama; dan karena `a[-1]` adalah elemen terakhir di Python, −1 yang terhitung keliru gagal secara diam-diam di sana. Buku: Subbab 5.2.3.

Larik Berisi Larik

Kode 5.5 menjumlahkan muatan tiga mesin selama tiga hari dengan `int[][]`: `washes[m][d]` adalah baris `m`, kolom `d`.

Kode 5.5 Dari `LoadStats.java`: jumlah per baris sebuah larik dua dimensi.

```
        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");
        }
```

```
Machine 1 washed 36 loads
Machine 2 washed 33 loads
Machine 3 washed 33 loads
Machine 2 on day 3: 14
```

Paruh pertama program menyimpan total berjalan dan maksimum berjalan yang dimulai dari elemen pertama, bukan dari 0; rata-ratanya $27.0/5 = 5.40$*. "Mesin 2 pada hari 3" adalah* `washes[1][2]` *karena kedua indeks dimulai dari 0. Buku: Subbab 5.2.4.*

Variabel Larik Menyimpan Referensi

Kode 5.6 memperlihatkan bahwa penugasan menyalin **anak panahnya**, bukan lariknya, dan bagaimana membuat salinan sungguhan.

Kode 5.6 Dari `ArrayCopy.java`: alias, `Arrays.copyOf`, serta perbandingan dengan `==` dan `Arrays.equals`.

```
        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[] twin = {99, 2, 3};
        System.out.println("original == twin? " + (original == twin));
        System.out.println("Arrays.equals(original, twin)? " + Arrays.equals(original, twin));
```

`original` menjadi `[99, 2, 3]` melalui aliasnya; `==` bernilai `false`, `Arrays.equals` bernilai `true`. Larik tidak dapat tumbuh: `Arrays.copyOf(original, 6)` membuat larik yang lebih besar lalu menyalin. Di Python, `is` bertanya "objek yang sama?" dan `==` membandingkan isi. Buku: Subbab 5.2.5.

Larik di dalam `main`: `args`

`java Hello2 one two three` memasukkan tiga string ke `args` (**Kode 5.7**); Python memasukkannya ke `sys.argv`, sesudah nama skrip itu sendiri (**Kode 5.8**).

Kode 5.7 `Hello2.java`: mencetak argumen baris perintah.

```java
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]);
        }
    }
}
```

Kode 5.8 `hello2.py`: mencetak argumen baris perintah dari `sys.argv`.

```python
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])
```

Argumen baris perintah membawa apa yang berubah dari satu eksekusi ke eksekusi lain: nama berkas, tanggal, mode. Di VS Code, tambahkan entri `"args"` ke `launch.json`. Buku: Subbab 5.2.6.

List yang Tumbuh

Pada Senin pagi binatu itu tidak tahu berapa pesanan yang akan datang. Ukuran larik tetap mana pun keliru: terlalu kecil, program macet; terlalu besar, memori terbuang dan kita harus menghitung slot yang terpakai.

> **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.

`ArrayList<String>` dibaca “sebuah `ArrayList` yang menampung `String`, dan hanya `String`”. Bab 9 menjelaskan tipe *generik* ini selengkapnya.

Java menyediakan list yang dapat tumbuh sebagai kelas pustaka, `java.util.ArrayList`; Python membangunnya ke dalam bahasa. Buku: Subbab 5.3.1.

Operasi `ArrayList` Sehari-hari

Kode 5.9 menambahkan enam buah, membaca satu, lalu menanyakan letak dua di antaranya.

Kode 5.9 Dari `FruitList.java`: `add`, `size`, `get`, dan `indexOf`.

```
        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));
        int positionOfPeach = fruit.indexOf("peach");
        int positionOfKiwi = fruit.indexOf("kiwi");
        System.out.println("peach at " + positionOfPeach + ", kiwi at " + positionOfKiwi);
```

```
List: [mango, banana, peach, plum, durian, plum], size 6
First fruit: mango
peach at 2, kiwi at -1
```

`indexOf` mengembalikan −1 untuk "tidak ada", yang tidak mungkin menjadi indeks sah. `contains` menjawab pertanyaan ya-atau-tidak. Tiga ejaan untuk "berapa banyak": `array.length`, `string.length()`, `list.size()`. Buku: Subbab 5.3.2.

Mengganti, dan Dua Macam `remove`

Kode 5.10: `remove(3)` menghapus elemen pada *posisi* 3 dan mengembalikannya; `remove("plum")` menghapus elemen pertama yang sama dan mengembalikan `true` atau `false`.

Kode 5.10 Dari `FruitList.java`: `set`, serta penghapusan menurut posisi dan menurut nilai.

```
        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"));
```

```
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
```

Elemen-elemen sesudahnya bergeser ke kiri, sehingga indeksnya berubah. `index` dan `remove` di Python ***melempar*** *`ValueError` untuk nilai yang tidak ada, alih-alih mengembalikan* -1 *atau `false`; penghapusan menurut posisi ditulis `pop(3)` atau `del fruit[3]`. Buku: Subbab 5.3.2.*

Larik, `ArrayList`, `list`: Membaca dan Menulis

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`

Tabel 5.1 *adalah paruh pertama perbandingan berdampingan itu; simpanlah di dekat Anda sepanjang semester ini. Buku: Subbab 5.3.2.*

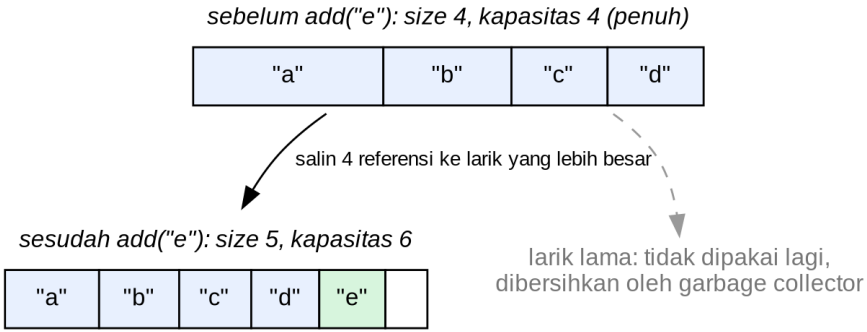
Larik, `ArrayList`, `list`: Tumbuh dan Mencari

Tabel 5.2 Operasi sehari-hari pada larik Java, `ArrayList` Java, dan `list` Python.

Operasi	Larik Java `a`	`ArrayList<T> list` Java	`list` Python
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`

Tabel 5.2 *adalah paruh keduanya: larik tidak dapat menambah di akhir, menyisipkan, atau menghapus, dan baris terakhir menyebut eksepsi yang dilempar setiap wadah untuk indeks yang buruk. Buku: Subbab 5.3.2.*

Bagaimana List Tumbuh



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.

*Di dalam setiap `ArrayList` ada larik biasa dengan slot cadangan; list menyimpan size (ukuran) dan capacity (kapasitas)-nya. Ketika lariknya penuh, `add` melakukan apa yang diperlihatkan **Gambar 5.2**. Buku: Subbab 5.3.3.*

Tumbuh Satu, atau Berlipat Dua?

Tumbuh satu slot: menambahkan n elemen menyalin sebanyak Persamaan (5.2); untuk $n = 1000$ itu 499,500 salinan.

$$0 + 1 + 2 + \cdots + (n-1) = \frac{n(n-1)}{2} \tag{5.2}$$

Berlipat dua ketika penuh: penyalinan terjadi pada ukuran 1,2,4, … , 2^k, dan Persamaan (5.3) membatasi totalnya; untuk $n = 1000$ totalnya 1023.

$$1 + 2 + 4 + \cdots + 2^k = 2^{k+1} - 1 < 2n \tag{5.3}$$

Kurang dari dua salinan per elemen secara rata-rata: menambah di akhir, secara rata-rata, semurah menulis ke dalam larik.

Java tumbuh dengan faktor 1.5*, CPython dengan faktor yang sedikit lebih kecil; faktor apa pun yang lebih besar dari 1 memberikan argumen yang sama dengan konstanta yang berbeda. Soal Latihan 5.4 mengukur kedua strategi. Buku: Subbab 5.3.3.*

List Berisi Bilangan: *Boxing*

`ArrayList<int>` tidak dapat dikompilasi: parameter tipe harus berupa kelas, jadi kita menulis `ArrayList<Integer>`. **Kode 5.11** memperlihatkan satu jebakan yang terkenal.

Kode 5.11 Dari `BoxingDemo.java`: *boxing*, dan dua makna `remove` pada list berisi `Integer`.

```
        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);
```

Keluarannya: `[5, 10, 3, 7]`, lalu `[5, 10, 3]`, lalu `[5, 10]`. `remove(3)` menyerahkan sebuah `int`, yang cocok persis dengan `remove(int index)`. Python menghindari kerancuan ini dengan dua kata: `del weights[3]` untuk posisi, `weights.remove(3)` untuk nilai. Larik cocok untuk ukuran tetap dan tipe primitif; bila ragu, pilihlah `ArrayList`. Buku: Subbab 5.3.4 dan 5.3.5.

Membaca Papan Ketik dengan `Scanner`

Kode 5.12 membaca satu baris, satu bilangan bulat, satu bilangan desimal, dan satu baris lagi.

Kode 5.12 Isi `SquareInput.java`: membaca dengan `Scanner`.

```
        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 + "\".");
```

`nextInt()`, `nextDouble()`, dan `next()` membaca satu token; `nextLine()` membaca sisa baris. Agar dapat dijalankan ulang persis sama, alihkan sebuah berkas: `java SquareInput < SquareInput.input.txt`. Buku: Subbab 5.4.1.

Sisa Pindah Baris

`nextDouble()` membaca `3.5` lalu berhenti **tepat sesudahnya**, meninggalkan pindah baris yang belum terbaca. `nextLine()` yang langsung menyusul akan mengembalikan `""`, dan catatan untuk kurir tidak terbaca oleh siapa pun.

> **ATURAN.** Sesudah `nextInt()`, `nextDouble()`, atau `next()`, panggillah `nextLine()` sekali sebelum Anda menginginkan satu baris utuh — baris 14 pada **Kode 5.12**.

`input()` di Python selalu membaca satu baris utuh dan mengembalikan `str`; kita mengonversinya dengan `int(...)` atau `float(...)`, seperti pada **Kode 5.13**. Tidak ada sisa, tidak ada jebakan.

Kode 5.13 Akhir `square_input.py`: tidak ada yang perlu ditelan, karena `input()` membaca baris utuh.

```
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 + '".')
```

Jika pengguna mengetik `twelve`, `nextInt()` maupun `int()` menolak dan program macet; Subbab 5.5 dikhususkan untuk menanganinya dengan lebih baik. Buku: Subbab 5.4.1.

Membaca sampai Bertemu *Sentinel*

Kode 5.14 membaca nama buah sampai `#stop#`: masukan, perulangan `while`, dan list yang tumbuh sekaligus.

Kode 5.14 Isi `FruitCollector.java`: membaca baris-baris ke dalam `ArrayList` sampai bertemu *sentinel* `#stop#`.

```
        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?");
```

```
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?
```

Pembacaan awal terjadi sebelum `while`; string dibandingkan dengan `equals`, bukan `==`. Tanpa `#stop#`, `nextLine()` melempar `NoSuchElementException`. Buku: Subbab 5.4.2.

Apa Itu 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.

Yang sudah kita jumpai: `ArrayIndexOutOfBoundsException`, `IndexError`, `ValueError`. Segera menyusul: `InputMismatchException`, `NullPointerException`.

Salah satu masalah pemrograman adalah pengguna: kita hanya dapat menduga apa yang mungkin salah lalu meredam akibatnya. Buku: Subbab 5.5.1.

Program Berjumpa dengan Pengguna

Kode 5.15 membaca bilangan bulat sampai 99 dan memercayai pengguna sepenuhnya. Masukannya `1`, `2`, `b`, `3`, `99`.

Kode 5.15 Isi `InputTest.java`: membaca bilangan bulat sampai 99, tanpa perlindungan terhadap masukan yang buruk.

```
        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());
```

```
Enter some numbers!
Adding 1
Adding 2
Exception in thread "main" java.util.InputMismatchException
    at InputTest.main(InputTest.java:12)
```

Empat baris di tengah jejak itu (di dalam `Scanner`) dihilangkan di sini; buku mencetak semuanya. Buku: Subbab 5.5.1.

Membaca *Stack Trace*

Apa: baris pertama menyebut tipenya, `java.util.InputMismatchException`.

Di mana: telusuri ke bawah sampai baris pertama yang menyebut kelas *Anda*: `at InputTest.main(InputTest.java:12)`, yaitu `nextInt()` kedua, yang membaca `b`.

> **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.

Traceback Python berjalan ke arah sebaliknya, "most recent call last": bacalah dari bawah ke atas. `ValueError: invalid literal for int() with base 10: 'b'` bahkan mengutip teks yang buruk itu.

Nama berkas dan nomor baris milik pustaka (`Scanner.java:947`) dapat berbeda antarversi Java; nomor baris Anda sendiri tidak. Buku: Subbab 5.5.2.

Propagasi: Tiga Panggilan ke Dalam

Pada **Kode 5.16**, `main` → `loadAll` → `readOrder` → `parseWeight`, yang melempar eksepsi untuk berat `-2`.

Kode 5.16 Dua metode terbawah `PropagationDemo.java`: eksepsi dilemparkan tiga panggilan ke dalam.

```
    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");
    }
```

`loadAll` memanggil `readOrder` untuk setiap `"Sari,3.5"`, `"Budi,-2"`, `"Tono,4"`, dan `main` memanggil `loadAll` di dalam `try` yang menangkap `IllegalArgumentException`. Buku: Subbab 5.5.2.

Buku Harian Perjalanan Eksepsi

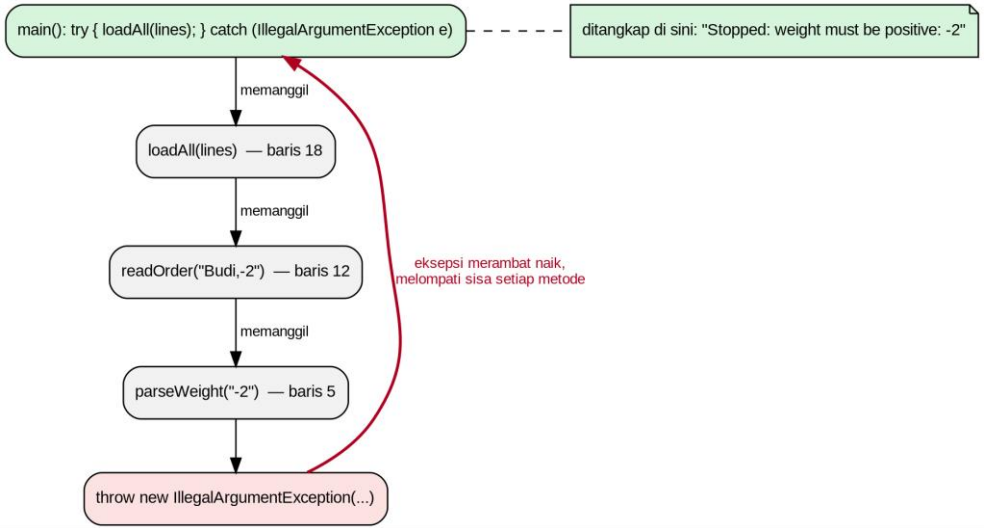
Stack trace yang dicetak **Kode 5.16** mendaftar setiap metode yang ditinggalkan, yang terbaru lebih dahulu:

```
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)
```

`Tono,4` tidak pernah dibaca, dan “All lines loaded.” tidak pernah dicetak: setiap metode ditinggalkan di titik panggilannya.

Baris 5 di `parseWeight`, dipanggil dari baris 12 di `readOrder`, dari baris 18 di `loadAll`, dari baris 26 di `main`. `traceback.print_exc` di Python memberikan jejak yang sama, dengan panggilan terbaru di akhir. Buku: Subbab 5.5.2.

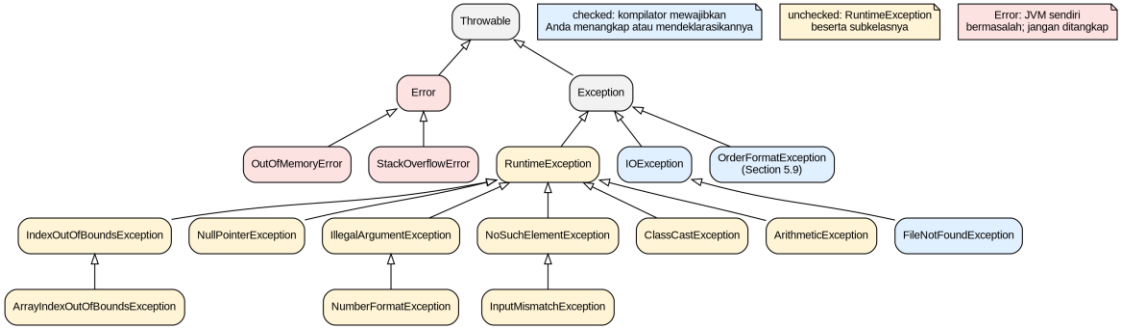
Propagasi dalam Gambar



Gambar 5.3 Propagasi. Panggilan bergerak turun dari `main`; eksepsi berjalan kembali ke atas, meninggalkan setiap metode yang dilaluinya, sampai mencapai `catch` yang cocok.

***Gambar 5.3**: perjalanan kembali menyusuri rantai panggilan inilah propagasi, dan ia berhenti pada `catch` pertama yang tipenya cocok. Buku: Subbab 5.5.2.*

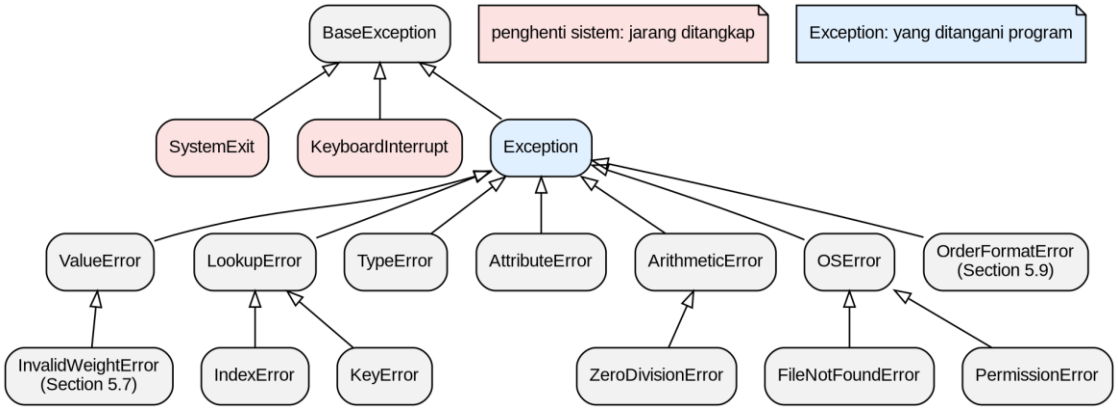
Silsilah Kegagalan di Java



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.

*Pada **Gambar 5.4**, `Error` berarti Java tidak dapat pulih (memori habis, stack overflow): kita tidak menangkapnya. Di bawah `Exception`, pembelahan antara `RuntimeException` dan sisanya adalah pokok bahasan Subbab 5.6. Buku: Subbab 5.5.3.*

Silsilah di Python



Gambar 5.5 Sebagian hierarki eksepsi Python. Program menangkap subkelas-subkelas `Exception`; eksepsi-eksepsi di sampingnya meminta program berhenti.

***Gambar 5.5**: jaring berlabel sebuah kelas juga menangkap setiap keturunannya, sehingga `except LookupError` menangkap `IndexError` maupun `KeyError`. Buku: Subbab 5.5.3.*

`try` dan `catch`

Tanpa eksepsi: seluruh blok `try` dijalankan dan `catch` dilewati. Dengan eksepsi: sisa `try` dilewati; jika tipenya cocok (**atau subkelasnya**), `catch` dijalankan; jika tidak, eksepsi terus berpropagasi.

Kode 5.17 memperbaiki `InputTest` untuk masukan `1 2 b 3 ten 99`:

Kode 5.17 Perulangan `SafeInput.java`: menangkap `InputMismatchException` agar masukan yang buruk dilaporkan dan dilewati.

```
        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.");
            }
        }
```

Python menulis `try:` … `except ValueError as ve:`, dan aturannya sama. Buku: Subbab 5.5.4.

Tiga Rincian yang Membuatnya Berhasil

`s.next()` membuang token yang buruk. `nextInt()` yang gagal tidak mengonsumsi `b`; tanpa `next()`, perulangan akan gagal pada `b` yang sama selamanya.

Jangan membuat `Scanner` baru di dalam `catch`. `Scanner` lama mungkin sudah menyangga masukannya; dengan berkas yang dialihkan, `Scanner` baru sering tidak menemukan apa-apa (Soal Latihan 5.12).

`while (s.hasNext())` berakhir dengan rapi jika masukan berhenti sebelum `99`.

```
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]
```

Versi Python, `for line in sys.stdin`, berhenti dengan rapi di akhir masukan, dan teks yang buruk sudah terkonsumsi ketika `int()` gagal; keluarannya tampak di atas. Buku: Subbab 5.5.4.

Jaring yang Makin Lama Makin Besar

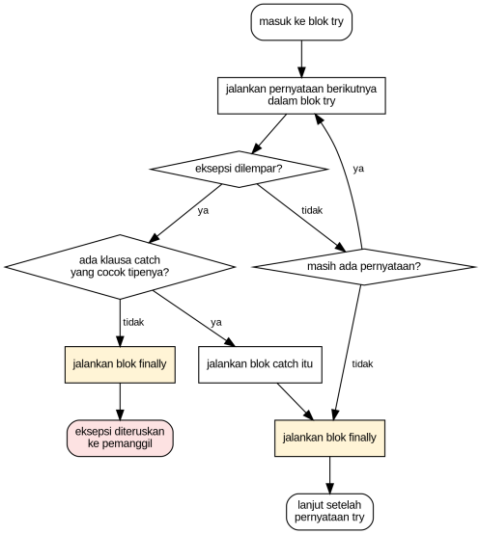
Kode 5.18 membagi 100 dengan bilangan yang diambil dari `{"12", "x7", "0"}` pada posisi 0, 1, 2, dan 5.

Kode 5.18 Perulangan `CatchOrder.java`: tiga klausa `catch`, dari yang paling spesifik ke yang paling umum.

```
        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);
            }
        }
```

Posisi 0: $100/12 = 8$. *Posisi 1:* `"x7"`, *jaring pertama. Posisi 2:* `100 / 0`, *jaring terlebar. Posisi 5: tidak ada, jaring kedua.* `catch (Exception e)` *di urutan pertama membuat klausa lain tak terjangkau, dan Java menolak mengompilasinya. Multi-catch:* `catch (A | B e)`*; Python:* `except (ValueError, ZeroDivisionError) as e:`*. Buku: Subbab 5.5.5.*

`finally`: Blok yang Selalu Dijalankan



Gambar 5.6 Jalur-jalur yang melewati pernyataan `try`–`catch`–`finally`. Setiap jalur melewati blok `finally`.

***Gambar 5.6**: sesudah akhir yang normal, sesudah `catch`, ketika eksepsi lolos, dan bahkan sesudah `return` — setiap jalur melewati `finally`. Di situlah berkas ditutup dan mesin dimatikan. Buku: Subbab 5.5.6.*

Contoh Terkerjakan 1: `finally` dan `return`

Kode 5.19 mempunyai `try` dan `finally` tetapi tanpa `catch`.

Kode 5.19 Metode `divide` di `FinallyDemo.java`: `finally` dijalankan sesudah `return` yang normal dan ketika eksepsi lolos.

```
    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");
        }
```

```
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.
```

Kasus 1: 5 dihitung, `finally` mencetak, lalu `main` menerima 5. Kasus 2: `1 / 0` melempar eksepsi, `finally` mencetak dalam perjalanan keluar, lalu `catch` di `main` menanganinya; `result =` tidak pernah tercetak. Buku: Subbab 5.5.6.

Contoh Terkerjakan 1, Lanjutan: Telusurilah

Soal Latihan 5.5: `f(x)` mencetak `A`, menghitung `10 / x`, mencetak `B`, mengembalikan `y`; `catch` mencetak `C` dan mengembalikan -1; `finally` mencetak `D`.

`f(0)`: `A`; `10 / 0` melempar eksepsi, jadi `B` dilewati; `catch` mencetak `C` dan menyiapkan -1; `finally` mencetak `D`; hasilnya -1.

`f(1)`: `A`; `y = 10`; `B`; `return` menyiapkan 10; `finally` mencetak `D`; hasilnya 10.

Python menambahkan `else`, yang dijalankan hanya jika `try` tidak melempar apa-apa; urutannya `try`, `except`, `else`, `finally`.

Blok `finally` dijalankan sesudah nilai yang akan dikembalikan dihitung dan sebelum nilai itu benar-benar dikembalikan. Buku: Subbab 5.5.6 dan Penyelesaian 5.5.

Melempar Eksepsi Sendiri

Kode 5.20 melarang nilai *grayscale* di atas 255 dengan `throw new IllegalArgumentException(…)`; Python menulis `raise ValueError(…)`.

`main` memanggilnya dengan 270 di dalam `try`; `catch` mencetak "270 is an invalid Grayscale value", menyetel ulang nilainya menjadi 255, dan pemrosesan berlanjut.

Kode 5.20 Metode `isExceeded` di `IllegalArgumentExceptionExample.java`: melempar `IllegalArgumentException` untuk nilai di atas 255.

```
    public static boolean isExceeded(int x) {
        boolean result = false;
        if (x > 255) {
            throw new IllegalArgumentException("Value exceeds 255");
        } else {
            result = true;
        }
        return result;
    }
```

Nama metode itu berbohong: `isExceeded` mengembalikan `true` ketika nilainya baik. Namailah `checkGrayscale`, jangan kembalikan apa-apa, dan lemparkan eksepsi untuk masukan yang buruk. Lemparkan eksepsi hanya untuk kesalahan, jangan sebagai cara cerdik keluar dari perulangan. Buku: Subbab 5.5.7.

Eksepsi Tak Terperiksa: Kesalahan Kita Sendiri

Kode 5.21 memancing tiga yang paling terkenal; semuanya memperluas `RuntimeException`.

Kode 5.21 Isi `UncheckedDemo.java`: tiga eksepsi tak terperiksa akibat kesalahan pemrograman.

```
        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);
        }
```

*Indeks 6 pada larik sepanjang 6; `length()` pada slot yang `null`; `(String)` pada sebuah `Integer`. Ini kesalahan program: tanggapan yang tepat adalah **membetulkan kodenya**, bukan menangkapnya. Padanannya di Python: `IndexError`, `TypeError`, `AttributeError`. Buku: Subbab 5.6.1.*

Eksepsi Terperiksa: Kompilator Bersikeras

Berkas yang tidak ada, disk yang penuh, *drive* jaringan yang terputus: bukan kesalahan pemrograman siapa pun, dan tidak ada kecermatan yang dapat mencegahnya. Program yang benar hanya dapat **bersiap** menghadapinya.

`new FileReader("filename.txt")` saja tidak dapat dikompilasi:

```
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
```

Eksepsi terperiksa memperluas `Exception` tetapi tidak memperluas `RuntimeException`: kotak-kotak biru pada silsilah Java. Buku: Subbab 5.6.2.

Tangkap, atau Deklarasikan

Kode 5.22: `firstLine` **mendeklarasikan** `throws IOException`, karena ia tidak dapat mengetahui arti berkas yang hilang; `main` **menangkap**, karena ia tahu bahwa ia menginginkan sebuah pesan.

Kode 5.22 Dari `OpenFile.java`: `firstLine` mendeklarasikan `IOException` dengan `throws`; `main` menangkapnya.

```
    static String firstLine(String fileName) throws IOException {
        BufferedReader br = new BufferedReader(new FileReader(fileName));
        String line = br.readLine();
        br.close();
        return line;
    }
            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.
```

Jaring spesifik lebih dahulu. `throws` pada `main` dapat dikompilasi, lalu membuat program macet. Python tidak mengenal eksepsi terperiksa: ia mendokumentasikannya di bagian `Raises:` pada docstring. Buku: Subbab 5.6.2–5.6.4.

Eksepsi Buatan Sendiri

Eksepsi buatan sendiri menyatakan **jenis** masalahnya (kelasnya), **persisnya** apa (pesannya), dan **dengan data** apa (*field*-nya dan penyebabnya). **Kode 5.23** adalah salah satunya.

Kode 5.23 Kelas bersarang `InvalidWeightException` di `WeightCheck.java`: eksepsi terperiksa yang membawa teks penyebab masalah.

```
    // 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;
        }
    }
```

Terperiksa (memperluas `Exception`) karena data buruk dari luar adalah hasil yang dapat diduga; tak terperiksa (`RuntimeException`) akan menandakan kesalahan pemrograman. `super(message, cause)` memakai ulang apa yang sudah disimpan `Exception`. Buku: Subbab 5.7.

Merangkai: Bungkus, Jangan Hilangkan

Pada **Kode 5.24** `NumberFormatException` menjadi **penyebab** eksepsi kita: satu pesan setingkat binatu, tanpa ada yang hilang.

Kode 5.24 Metode `parseWeight` di `WeightCheck.java`: menerjemahkan eksepsi tingkat rendah, dan menolak berat di luar 0–50 kg.

```
    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;
    }
```

Untuk −1 dan 80 tidak ada eksepsi yang mendasarinya, jadi penyebabnya `null`. Python menulis `raise InvalidWeightError(…) from ve`, dan eksepsi aslinya adalah `e.__cause__`. Buku: Subbab 5.7.

Menutup yang Dibuka, Cara Lama

Setiap berkas yang terbuka memegang pegangan dari sistem operasi, dan data yang ditulis baru dijamin sampai di disk sesudah `close()`. **Kode 5.25** menutup di dalam `finally`.

Kode 5.25 Blok `finally` di `CopyFileClassic.java`: menutup kedua *stream*, cara sebelum Java 7.

```
        } 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 */ }
        }
```

Kedua stream dideklarasikan sebelum `try` dan diberi `null`; `finally` memeriksa `null` dan membungkus setiap `close()` dengan `try`-nya sendiri. Hampir separuh program mengurusi penutupan. Buku: Subbab 5.8.2.

`try`-with-resources

Apa pun yang `AutoCloseable` dan dibuka di dalam tanda kurung sesudah `try` ditutup ketika bloknya berakhir, bagaimanapun berakhirnya, dalam urutan terbalik. **Kode 5.26** adalah salinan versi modern.

Kode 5.26 Metode `copy` di `CopyFile.java`: `try`-with-resources menutup kedua *stream* secara otomatis.

```
    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;
    }
```

```
Copied 102 bytes from notes.txt to notes_copy2.txt
Could not copy nothing.txt: nothing.txt (No such file or directory)
```

`copy` mendeklarasikan, `main` menangkap. `with open(a, "rb") as source, open(b, "wb") as target:` di Python memberikan jaminan yang sama. Dalam kode yang baru, bukalah setiap berkas dengan cara ini. Buku: Subbab 5.8.3.

Menulis dan Menambahkan Teks

`FileWriter` menimpa berkas; `FileWriter(name, charset, true)` **menambahkan** di akhir. Python: `open` dengan mode `"w"` atau `"a"`.

Susun teks sekali saja: `text = text + more` di dalam perulangan adalah jebakan kuadratik Persamaan (5.2); pakailah `StringBuilder`, atau kumpulkan potongannya lalu `"".join(parts)`.

Nyatakan pengodeannya: `StandardCharsets.UTF_8` dan `encoding="utf-8"`, atau "José" yang ditulis di satu mesin bisa kacau di mesin lain.

Baca sampai habis: `while ((line = in.readLine()) != null)` di Java; `for line in f` di Python.

Stream membawa bita, reader dan writer membawa karakter, dan `BufferedReader` menyerahkan baris utuh. Program struk menulis tiga butir dan totalnya, menambahkan "Thank you for choosing Profitina Laundry!", lalu membaca kembali berkasnya dengan nomor baris. Buku: Subbab 5.8.1 dan 5.8.4.

CSV: Dua Jebakan `split`

Kode 5.27 mencetak setiap *field* di antara `<` dan `>` agar spasi dan *field* kosong terlihat.

Kode 5.27 Dari `CsvSplit.java`: jebakan pertama ketika memecah satu baris CSV di Java.

```
        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));
```

```
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><>
```

Jebakan 1 (hanya Java): `split(",")` diam-diam membuang field kosong di ujung; `split(",", -1)` mempertahankannya. Jebakan 2 (semua bahasa): field bertanda kutip seperti `"Santoso, Budi"` memuat koma; `csv.reader` di Python menanganinya, Java memerlukan pustaka. Buku: Subbab 5.8.5.

Profitina Laundry: Alurnya



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.

***Gambar 5.7**: baris yang buruk adalah masalah setempat, jadi kita mencatatnya lalu melanjutkan; berkas yang tidak ada atau header yang salah adalah masalah menyeluruh, jadi pemuat melempar eksepsi dan aplikasilah yang memutuskan. Buku: Subbab 5.9.*

Contoh Terkerjakan 2: Berkas yang Rusak

`orders.csv`, dirusak dengan tujuh cara, dengan 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
```

Aturannya: tepat lima field; id dan nama tidak kosong; layanan yang dikenal; berat dalam (0,50] *kg;* `yes` *atau* `no`*; id tidak berulang. Buku: Subbab 5.9.*

Harga dalam Rupiah Bulat

`WASH` Rp 7,000, `IRON` Rp 5,000, `WASH_IRON` Rp 10,000 per kilogram; ekspres 50% lebih mahal (Persamaan (5.4)).

$$p = \begin{cases} \text{round}(w \cdot r) & \text{pesanan biasa,} \\ \text{round}(w \cdot r) + \lfloor \text{round}(w \cdot r)/2 \rfloor & \text{pesanan ekspres.} \end{cases} \tag{5.4}$$

Sari: $\text{round}(3.5 \times 7000) = 24{,}500$. Budi, cuci-setrika ekspres: $50{,}000 + 25{,}000 = 75{,}000$.

Uang dihitung dengan bilangan bulat: $0.1 + 0.2$ tidak persis sama dengan 0.3 dalam *floating-point* biner.

`Order` menyimpan layanan dan tarifnya dalam dua larik paralel (Java) atau sebuah dictionary (Python), dan `rateFor` mengembalikan -1 untuk kode yang tidak dikenal. Buku: Subbab 5.9 dan 5.9.1.

Kelas `Order` Mengetahui Harganya

Kode 5.28 mengimplementasikan Persamaan (5.4) dengan pencarian di dalam dua larik paralel.

Kode 5.28 Dari `Order.java`: tarif, `rateFor`, dan `price`.

```
    static final String[] SERVICES = {"WASH", "IRON", "WASH_IRON"};
    static final long[] RATES = {7000, 5000, 10000};
    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
    }
```

`Math.round` mengembalikan `long`; `base / 2` adalah pembagian bilangan bulat. Larik cocok untuk layanan karena jumlahnya sudah tetap ketika program ditulis; Python memakai dictionary. Buku: Subbab 5.9.1.

`parseLine`: Satu Aturan demi Satu Aturan

Kode 5.29: hitung *field*-nya **sebelum** memakai satu pun; aturan pertama yang gagal melempar `OrderFormatException` beserta nomor barisnya.

Kode 5.29 Sebagian `parseLine` di `OrderCsv.java`: cacah *field* lebih dahulu, dan berat yang dibungkus sebagai penyebab.

```
    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);
        }
        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");
        }
```

Di antara kedua bagian itu, field dirapikan dan id, nama, serta layanan diperiksa dengan cara yang sama. `!(kg > 0 && kg <= 50)` *juga menolak* `NaN`*, yang membuat setiap perbandingan bernilai salah. Buku: Subbab 5.9.3.*

`load`: `try` Bagian Dalam Adalah Rancangannya

Kode 5.30: sesudah pemeriksaan *header*, setiap baris mempunyai `try`-nya sendiri, yang `catch`-nya mencatat pesan lalu **terus melaju**.

Kode 5.30 Perulangan `load` di `OrderCsv.java`: satu baris buruk menjadi satu pesan.

```
            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
                }
            }
```

Berkasnya dibuka dengan `try`-with-resources, dan header yang salah melempar eksepsi sebelum perulangan ini. Letak `try` bagian dalam itulah keputusan rancangan alurnya, diwujudkan dalam kode. Id ganda dilemparkan di dalam `try` yang sama, sehingga setiap baris buruk dilaporkan dengan cara yang sama. Buku: Subbab 5.9.3.

Perulangan yang Sama di Python

Kode 5.31 membaca melalui `csv.reader`, yang atribut `line_num`-nya memberikan nomor baris yang baru saja dibaca.

Kode 5.31 Isi `load` di `order_csv.py`: pemuatan dan validasi yang sama di Python.

```
    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
```

Berkas dibuka dengan `newline=""`, sebagaimana diharuskan dokumentasi `csv`, dan dengan `encoding="utf-8"`. Buku: Subbab 5.9.3.

Contoh Terkerjakan 2: Hasilnya

`LaundryApp.java` memuat, melaporkan, menambahkan pesanan Maya, menghitung harga, menyimpan, memuat ulang, lalu mencoba berkas yang tidak ada dan berkas yang asing:

```
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).
```

Pendapatannya $24{,}500 + 75{,}000 + 10{,}000 + 126{,}000 + 22{,}500 = 258{,}000$. *Aplikasi Python mencetak baris-baris yang sama dan menulis* `orders_clean.csv` *yang identik sampai ke tingkat bita. Buku: Subbab 5.9.4.*

Profitina dalam Praktik

List yang ukurannya tidak diketahui: pemindaian situs tidak tahu berapa halaman, blok data terstruktur, atau sinyal yang akan ditemukannya; masing-masing adalah list yang dapat tumbuh.

CSV di tepian: ekspor tren skor mingguan dan ekspor data pribadi (UU PDP 27/2022) memakai UTF-8, baris *header*, dan tanda kutip yang benar.

Kegagalan dilaporkan, bukan disembunyikan: ketika model lokal (Qwen2.5 di Ollama) tidak tersedia, aplikasi *mengatakannya*; rilis baru melaporkan keberhasilan sesudah pemeriksaan kesehatannya lolos.

Validasi sebelum percaya: setiap tautan komunitas disaring; unggahan yang gagal terkirim dipulihkan agar dapat dicoba lagi.

Perangkat lunak memberi saran, penggunalah yang memutuskan — dan program hanya dapat memberi saran yang baik jika ia berkata jujur tentang apa yang salah. Buku: Subbab 5.10.

Kesalahan yang Sering Terjadi

`i <= a.length`, panjang yang ditulis mati, atau mencampuradukkan `length`, `length()`, dan `size()`.

Alias yang dikira salinan; `==` pada larik di Java; `remove(3)` pada list berisi `Integer`; mengharapkan -1 dari `index` di Python.

Sisa pindah baris sesudah `nextInt()`; mengulang `nextInt()` tanpa membuang token yang buruk.

`catch (Exception e)` **lebih dahulu atau di mana-mana**; `catch` yang kosong; menangkap eksepsi tak terperiksa alih-alih membetulkan *bug*-nya; `throws` pada `main` untuk membungkam kompilator.

Berkas yang tidak pernah ditutup; pengodean bawaan; `split(",")` yang kehilangan *field* di ujung; metode yang namanya berbohong; eksepsi untuk alur kendali biasa.

Buku mendaftar sembilan belas kesalahan, masing-masing dengan cara menghindarinya. Buku: Subbab 5.11.

Ringkasan: Lima Hal untuk Diingat

1. Larik adalah deretan slot tetap berindeks 0 sampai $n-1$, dicapai langsung menurut Persamaan (5.1); variabelnya menyimpan referensi, jadi penugasan membuat alias.

2. List tumbuh dengan sendirinya; pertumbuhan geometris membuat penambahan di akhir murah secara rata-rata (Persamaan (5.3)). List bilangan di Java memerlukan `Integer`.

3. Eksepsi dilemparkan, berpropagasi kembali melalui panggilan-panggilan, dan ditangkap oleh klausa pertama yang cocok; urutkan jaringnya dari yang spesifik ke yang umum; `finally` selalu dijalankan.

4. Eksepsi tak terperiksa adalah *bug* yang harus dibetulkan; eksepsi terperiksa harus ditangkap atau dideklarasikan. Python tidak mengenalnya, jadi dokumentasikanlah.

5. Tutuplah setiap berkas dengan `try`-with-resources atau `with`, nyatakan UTF-8, dan validasi setiap *field* CSV.

Profitina Laundry kini memuat berkas yang rusak, menjelaskan setiap baris buruk, terus melaju, dan menyimpan berkas yang dapat dimuat ulang dengan bersih. Buku: Subbab 5.12.

Pertemuan Berikutnya

Bab 6 — Dokumentasi, Pengujian, dan Antarmuka. Javadoc dan *docstring*, JUnit dan `unittest`, *overloading*, serta antarmuka sebagai kontrak.

Harga di Profitina Laundry menjadi sekeluarga aturan harga yang dapat saling dipertukarkan, masing-masing teruji.

Sebelum itu: kerjakan Soal Latihan 5.2, 5.5, 5.8, dan 5.10, di Java dan di Python.

Bagian `Raises:` pada docstring yang kita jumpai hari ini adalah cicipan pertama dokumentasi di Bab 6. Buku: Subbab 5.13 dan 5.14.

Kuliah 6 — Dokumentasi, Pengujian, dan Antarmuka

Pemrograman Berorientasi Objek (EF234302)

Buku: Irfan Subakti, *Pemrograman Berorientasi Objek dengan Java dan Python*, Bab 6

Dosen: Irfan Subakti — yifana@gmail.com

Hari ini kelas-kelas kita belajar membuat janji secara tertulis — untuk manusia, untuk program uji, dan untuk kompilator. Buku: Bab 6, Subbab 6.1–6.19.

Cakupan Pertemuan Ini

1. **Dokumentasi** — beberapa konstruktor, Javadoc dan *docstring*, `javadoc`, `pydoc`, dan `doctest`.
2. **Pengujian** — uji tanpa pustaka, *arrange–act–assert*, JUnit 6, `unittest` dan pytest, nilai batas, uji lebih dulu.
3. **Kesamaan** — identitas dan kesamaan, `equals` bersama `hashCode`, `toString`, `__repr__`, dan `__str__`.
4. ***Overloading*** — cara Java memilih sebuah versi, dan apa yang dilakukan Python sebagai gantinya.
5. **Antarmuka** — `Measurable`, menukar representasi, `Comparable`, pendengar, ABC, dan *protocol*.
6. **Profitina Laundry** — lima aturan harga di balik satu antarmuka, setiap aturan teruji.

Tiga gagasan besar menjawab tiga pertanyaan canggung, berurutan; perkakas kecil di antaranya adalah sandaran ketiganya. Buku: Subbab 6.1.

Titik Berangkat: Tiga Pertanyaan Canggung

Bab 5 memberi kita larik, list, masukan, eksepsi, dan berkas: cukup untuk menjalankan toko kecil selama satu pagi, tetapi belum dapat dipercaya.

Bagaimana rekan kerja tahu cara memakai sebuah kelas? → **dokumentasi**.

Bagaimana kita tahu kelas itu bekerja? → **pengujian**, setiap kali, dalam sedetik.

Bagaimana mengubahnya tanpa merusak pemakainya? → **antarmuka**: sisa program bergantung pada janji, tidak pernah pada rinciannya.

> **WAWASAN KUNCI.** Dokumentasi, uji, dan antarmuka adalah tiga cara menuliskan sebuah *janji*: dengan kata-kata untuk manusia, dalam kode yang memeriksanya, dan dalam sistem tipe.

Perangkat lunak yang baik menjaga ketiganya tetap selaras. Buku: Subbab 6.1.

Dua Cara Membuka Rekening

Dibuka dengan uang: *overdraft* 50. Dibuka kosong: *overdraft* 10. Java memberi kelas itu **dua konstruktor**, yang dibedakan oleh daftar parameternya (Kode 6.1).

Kode 6.1 Dua konstruktor `BankAccount.java`.

```
    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;
    }
```

```
BankAccount[balance=0.0, overdraft=10]
BankAccount[balance=25.0, overdraft=50]
```

`new BankAccount()` memilih konstruktor tanpa parameter; `new BankAccount(25)` memilih yang menerima `double`. Kelas tanpa konstruktor mendapat konstruktor kosong dari kompilator. Buku: Subbab 6.2 dan 6.2.1.

Python: Satu `__init__`, Satu Argumen *Default*

Python hanya mengizinkan satu `__init__` per kelas; yang kedua akan menggantikan yang pertama. Kode 6.2 memakai nilai *default* `None`, "tidak ada apa-apa di sini".

Kode 6.2 `__init__` di `bank_account.py`: satu metode dengan argumen *default* mengerjakan tugas dua konstruktor Java.

```
    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
```

Mengapa `None`, bukan `0`? `BankAccount(0)` adalah setoran yang kebetulan nol dan mendapat overdraft yang lebih besar. Di Java, satu konstruktor dapat memanggil yang lain dengan `this(...)` (Soal Latihan 6.3). Buku: Subbab 6.2.1.

Komentar untuk Orang yang Tak Pernah Membaca Kode Anda

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 butir yang digambarkannya. Di Python bentuknya adalah *docstring*: literal string yang ditulis sebagai pernyataan paling awal di dalam modul, kelas, atau fungsi yang digambarkannya.

Anda sudah memakai `String` dan `ArrayList` selama lima bab tanpa melihat kode sumbernya: halaman API memberi tahu semua yang dibutuhkan pemanggil.

Komentar biasa menjelaskan satu baris kepada pembaca kode; komentar dokumentasi menjelaskan kontraknya kepada pemakai. Buku: Subbab 6.3.

Komentar Javadoc, Tag demi Tag

Kalimat pertama adalah *ringkasan*; lalu satu `@param` per parameter, `@return`, dan `@throws` (Kode 6.3).

Kode 6.3 Metode `interestOn` di `BankAccount.java`, beserta komentar Javadoc-nya.

```java
/**
 * 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;
}
```

Tubuhnya menghitung bunga majemuk, $P(1+r)^n - P$: $123\,600$ untuk $1\,000\,000$ dengan 6% selama 2 tahun. "6% ditulis 0.06" termasuk komentar; "memakai `Math.pow`" tidak. Python: `Args:`, `Returns:`, `Raises:`. Buku: Subbab 6.3.1 dan 6.3.2.

Tag yang Akan Anda Pakai

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::`

Tabel 6.1: setiap tag Javadoc, dan tempat informasi yang sama diletakkan di dalam docstring Python. Buku: Subbab 6.3.1.

Jebakan: `/` alih-alih `*/`

Akhiri komentar `interestOn` dengan `/` saja, dan komentar itu berlanjut sampai `*/` **berikutnya** — menelan seluruh metode. `javac` diam saja; `javap` memperlihatkan metodenya lenyap:

```
Compiled from "Interest.java"
public class Interest {
  public Interest();
  public double twice(double);
}
```

Program yang memanggilnya lalu ditolak dengan `cannot find symbol`.

Penyunting dengan pewarnaan sintaksis menampilkan seluruh metode itu dengan warna komentar — satu alasan lagi untuk memakainya. Buku: Subbab 6.3.1.

Contoh yang Memeriksa Dirinya Sendiri

Docstring kelas di `bank_account.py` memuat baris-baris yang tampak seperti sesi Python:

Kode 6.4 Dari *docstring* kelas di `bank_account.py`: dua contoh *doctest*.

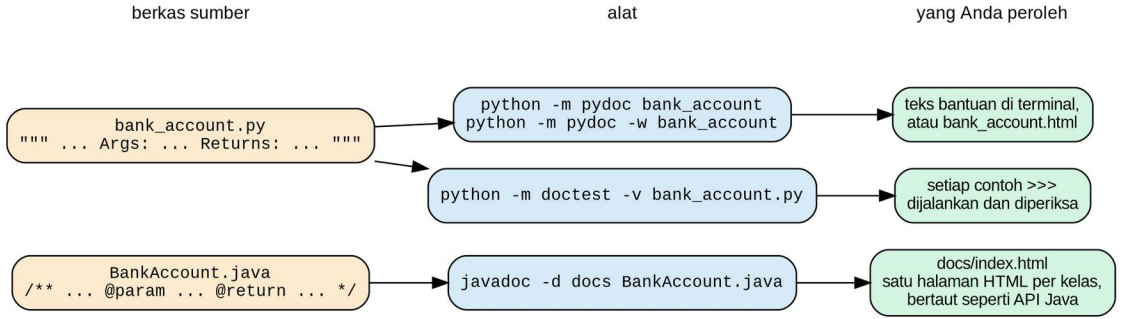
```
    >>> BankAccount().overdraft
    10
    >>> BankAccount(25).overdraft
    50
```

`python3 -m doctest -v bank_account.py` menjalankan setiap contoh lalu membandingkan; Kode 6.4 tidak mungkin usang tanpa ada yang menyadarinya.

```
3 tests in 7 items.
3 passed and 0 failed.
Test passed.
```

Doctest adalah sekilas pandang pertama kita pada pengujian: kode yang menjalankan kode lain dan membandingkan hasilnya dengan harapan. Buku: Subbab 6.3.3.

Dari Komentar Menjadi Halaman



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*.

Gambar 6.1: `javadoc -d docs BankAccount.java` membangun halaman seperti API Java dan melaporkan `@param` yang salah eja atau hilang; `python3 -m pydoc bank_account` mencetak bantuan. Buku: Subbab 6.4.1 dan 6.4.2.

Membaca API: `Random` dan *Seed*

Halaman API `java.util.Random` menyatakan bahwa `nextInt(10)` mengembalikan 0 (inklusif) sampai 10 (eksklusif), jadi mesin 1 sampai 4 adalah `nextInt(4) + 1` (Kode 6.5).

Kode 6.5 Dari `RandomDemo.java`: metode-metode utama `java.util.Random`, dengan *seed* tetap.

```
        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.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);
        }
```

```
Machines for 8 loads: 1 3 3 4 1 1 4 4
```

Seed `2026` membuat setiap eksekusi mencetak bilangan yang sama. Binatu sungguhan tidak memerlukan seed; uji memerlukannya, karena uji yang masukannya berubah setiap kali tidak membuktikan apa-apa. Python: `random.Random(2026)`; `randint(a, b)` mencakup b*. Buku: Subbab 6.4.3.*

Apa Itu Uji Unit

> **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.

Melihat itu tidak andal — manusia melewatkan `27000` untuk `28000`; kode tidak pernah.

Melihat itu tidak diulang — rangkaian uji berjalan lagi sesudah setiap perubahan.

Uji mengubah cara Anda merancang — kode yang sulit diuji hampir selalu terbagi dengan buruk.

Program benar jika ia melakukan apa yang dijanjikan dokumentasinya untuk setiap masukan yang diterimanya; kita tidak dapat mencoba semuanya, jadi kita memilih dengan cerdik. Buku: Subbab 6.5.

Kelas yang Diuji

`Dog` mempunyai nama dan umur; ulang tahun menambah satu tahun; umur negatif ditolak (Kode 6.6).

Kode 6.6 Konstruktor dan `incrementAge` di `Dog.java`, kelas yang akan kita uji.

```
    public Dog(String name, int age) {
        if (age < 0) {
            throw new IllegalArgumentException("age must not be negative: " + age);
        }
        this.name = name;
        this.age = age;
    }
    public void incrementAge() {
        age++;
    }
```

`Dog` tidak mempunyai `main`: ia kelas pustaka, dan kode pertama yang memakainya adalah uji-ujinya. `dog.py` di Python berbentuk sama, dengan property hanya-baca. Buku: Subbab 6.5.1.

Contoh Terkerjakan 1: Uji Tanpa Pustaka

Yang dibutuhkan sebuah uji hanyalah "saya mengharapkan ini, saya mendapat itu — cocokkah?" Kode 6.7 menuliskannya sebagai `check`.

Kode 6.7 Metode `check` di `DogCheck.java`: menguji tanpa pustaka uji apa pun.

```
    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);
        }
    }
```

Nilai yang diharapkan ditulis lebih dahulu, dan kegagalan menyebut apa yang diharapkan dan apa yang sebenarnya terjadi; sebagian besar nilai sebuah uji terletak pada pesan itu. Buku: Subbab 6.5.2.

Contoh Terkerjakan 1: Pemeriksaannya

Kode 6.8 membangun Rover berumur 5, bertindak, lalu memeriksa terhadap bilangan yang dihitung **sebelum** apa pun dijalankan.

Kode 6.8 Dari `main` di `DogCheck.java`: tiga pemeriksaan dan satu kasus tepi.

```
        // 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");
        }
```

*Pemeriksaan terakhir adalah **kasus tepi**: ia lulus hanya jika `new Dog("Rex", -1)` melempar eksepsi. `main` berakhir dengan `System.exit(1)` bila ada yang gagal, karena perkakas build membaca status keluar. Buku: Subbab 6.5.2.*

Contoh Terkerjakan 1: Rusakkan dengan Sengaja

Tanpa kerusakan, kedua bahasa mencetak empat baris `PASS` dan `4 passed, 0 failed`. Sekarang ganti `age++` menjadi `age += 2` lalu jalankan lagi:

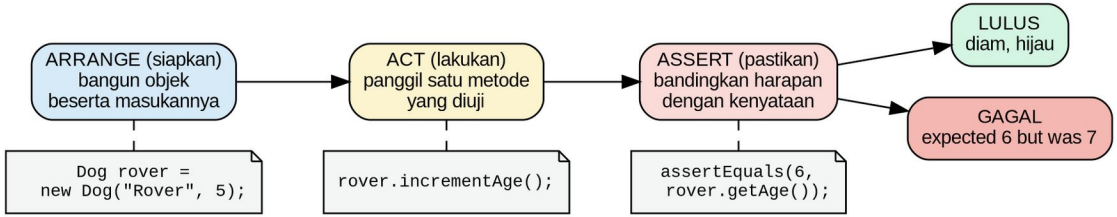
```
PASS  age is 5
FAIL  age is now 6: expected 6 but was 7
PASS  name is Rover
PASS  negative age is rejected
3 passed, 1 failed
```

Status keluarnya 1, dan pesannya menyebut janji yang dilanggar.

> **RENUNGKAN.** Jangan pernah memakai `assert` untuk memeriksa *data pengguna*: `python -O` menghapus setiap `assert`, dan `assert` di Java mati kecuali dijalankan dengan `java -ea`. `assert` untuk uji.

Batalkan perubahan itu sebelum melanjutkan. Buku: Subbab 6.5.2.

Arrange, Act, Assert



Gambar 6.2 *Arrange, act, assert*: bangun objeknya, panggil satu metode yang diuji, bandingkan nilai yang diharapkan dengan nilai sebenarnya.

Gambar 6.2: siapkan seminimal mungkin, lakukan sekali (satu uji, satu perilaku), pastikan terhadap literal — 6, jangan salinan rumus metode itu sendiri, yang akan menyetujui bug yang sama. Buku: Subbab 6.6.

Membandingkan Bilangan Pecahan

0.1 disimpan sebagai pecahan biner yang sedikit berbeda dari sepersepuluh, dan galatnya menumpuk (Kode 6.9).

Kode 6.9 Dari `DoubleEquality.java`: mengapa `==` adalah uji yang keliru untuk dua `double`.

```
    static boolean close(double expected, double actual, double epsilon) {
        return Math.abs(expected - actual) <= epsilon;
    }
        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));
```

```
0.1 + 0.2          = 0.30000000000000004
a == 0.3           : false
close(0.3, a, 1e-9): true
```

Terimalah jawaban yang cukup dekat: Persamaan (6.1), dengan $|0.3 - 0.30000000000000004| = 4 \times 10^{-17} \leq 10^{-9}$.

$$|e - a| \leq \varepsilon \tag{6.1}$$

JUnit `assertEquals(e, a, delta)`, `assertAlmostEqual(e, a, delta=…)`, pytest `a == pytest.approx(e)`. Lebih baik lagi untuk uang: rupiah bulat dalam `long` atau `int`. Buku: Subbab 6.6.1.

JUnit 6: Uji yang Sama, Berbingkai

Kode 6.10 menandai uji dengan `@Test`; `@BeforeEach` membangun Rover baru sebelum setiap uji.

Kode 6.10 Dari `DogTest.java`: uji-uji `Dog` yang ditulis untuk JUnit 6.

```
    @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");
    }
```

Tidak ada `main`: runner-lah yang memilikinya. Nilai yang diharapkan lebih dahulu. Uji ketiga, `assertThrows(…, () -> new Dog("Rex", -1))`, menyerahkan sebuah lambda untuk dijalankan JUnit di dalam `try`-nya sendiri. Buku: Subbab 6.7.1 dan 6.7.2.

`unittest`: Bentuk Java di Python

Pada Kode 6.11, metode yang namanya diawali `test` adalah uji; `setUp` memerankan `@BeforeEach`; `with self.assertRaises(...)` menggantikan *lambda*.

Kode 6.11 Dari `test_dog.py`: uji-uji `Dog` yang ditulis untuk `unittest`.

```
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)
```

Jalankan dengan `python3 -m unittest -v test_dog`; tanpa nama berkas ia menemukan setiap `test*.py`. Kegagalan mencetak `.F.` lebih dahulu: satu titik per uji lulus, `F` per uji gagal. Buku: Subbab 6.7.3.

JUnit 6, `unittest`, pytest

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`

Tabel 6.2: pytest membuang segala upacara — fungsi biasa, `assert` biasa — dan mencetak nilai setiap bagian ekspresi yang gagal. Buku: Subbab 6.7.4.

Kelas Ekuivalensi dan Nilai Batas

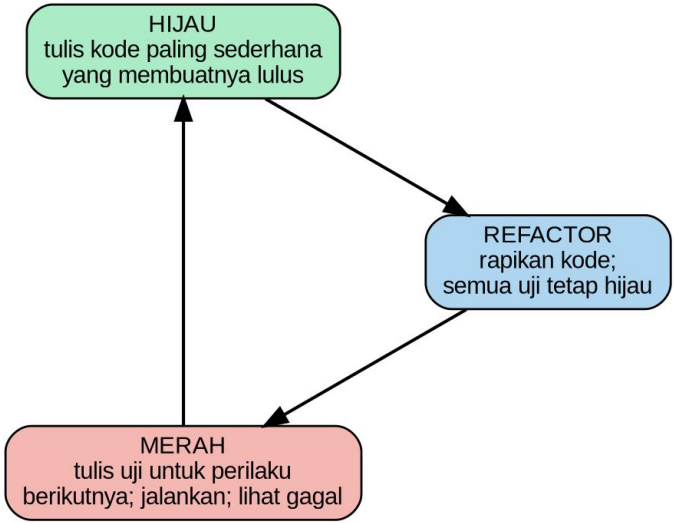
Cuci lipat: Rp 7 000 per kilogram yang **sudah dimulai**, tidak pernah kurang dari 3 kg. Tabel 6.3 memilih satu wakil per kelas dan nilai-nilai di setiap perbatasan.

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)

Di perbatasanlah pemrogram menulis `<` untuk `<=`. Untuk larik, ujilah yang pertama, yang terakhir, dan yang kosong; untuk setiap `if`, kedua cabangnya; untuk setiap eksepsi yang dijanjikan, uji yang memancingnya. Buku: Subbab 6.8.1.

Uji Lebih Dulu: Merah, Hijau, *Refactor*



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*).

Gambar 6.3: tulis satu uji yang gagal dan saksikan ia gagal; tulis kode paling sederhana yang meluluskannya; lalu rapikan dengan uji-uji sebagai jaring pengaman. Buku: Subbab 6.8.2.

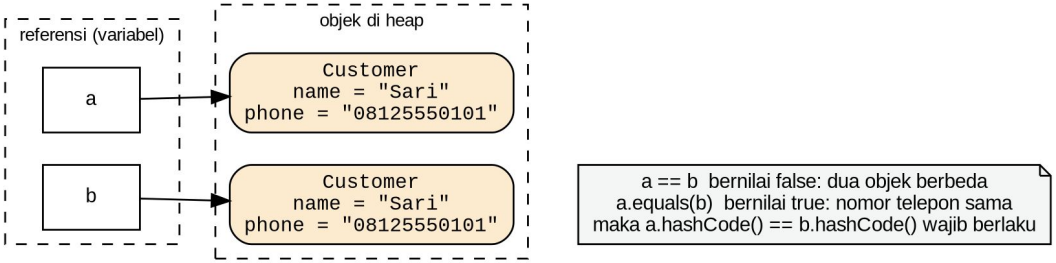
Kapan Dua Objek Dianggap Sama?

Senin: “Sari”, `0812-555-0101`. Kamis: ” Sari “, `0812 555 0101`. Pelanggan yang sama? Jika kelasnya diam saja, jawabannya **tidak**, dan poin loyalitasnya terbelah dua.

> **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)`).

`equals` yang diwarisi setiap kelas dari `Object` hanyalah `==`: sebuah objek hanya sama dengan dirinya sendiri. `__eq__` bawaan Python berperilaku sama. Buku: Subbab 6.9 dan 6.9.1.

Identik atau Sama?



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.

Gambar 6.4: dua variabel, dua objek terpisah, isi yang sama persis — tidak identik, tetapi sama jika kelasnya berkata demikian, dan kode hash-nya pun harus sama. Buku: Subbab 6.9.1.

`equals` Tanpa `hashCode`

`HalfCustomer` meng-*override* `equals` dan melupakan `hashCode`. Kode 6.12 memperlihatkan apa kata list dan *hash set* tentangnya:

Kode 6.12 Dari `IdentityDemo.java`: kelas dengan `equals` tetapi tanpa `hashCode`.

```
        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
```

```
  c.equals(d) : true
  list contains d: true
  set contains d : false
  set size       : 2
```

Hash set memilih ember menurut kode hash, lalu membandingkan dengan `equals` di dalamnya; `hashCode` warisan mengirim `c` dan `d` ke ember berbeda. Buku: Subbab 6.9.1.

Aturan yang Membuat *Hashing* Bekerja

Untuk dua objek a, b dan fungsi *hash* h mana pun, Persamaan (6.2) harus berlaku:

$$a \text{ sama dengan } b \Rightarrow h(a) = h(b) \tag{6.2}$$

Tidak berlaku terbalik: dua objek berbeda boleh berbagi kode (*collision*, tabrakan), yang hanya sedikit mengurangi kecepatan.

ATURAN. Setiap kali Anda meng-*override* `equals`, *override* juga `hashCode`, dengan *field* yang sama.

Python lebih ketat, dan itu lebih ramah: kelas dengan `__eq__` tanpa `__hash__` menjadi unhashable, dan memasukkannya ke himpunan langsung melempar `TypeError`. Buku: Subbab 6.9.1.

`Customer` yang Benar

Pelanggan dikenali dari **digit-digit** nomor teleponnya; Kode 6.13 menormalkannya di konstruktor dan tidak membandingkan hal lain.

Kode 6.13 `equals` dan `hashCode` di `Customer.java`: kesamaan menurut nomor telepon, dan *hash* dari *field* yang sama.

```
    @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);
    }
    @Override
    public int hashCode() {
        return phone.hashCode();
    }
```

Empat langkahnya: objek yang sama? kelas yang benar (dan bukan `null`)? cast; bandingkan field-nya dengan `equals`. Jangan menulis `equals(Customer other)`: itu overloading, bukan overriding, dan `@Override` menangkapnya. Buku: Subbab 6.9.2.

Kontrak `equals`

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 (6.2): objek-objek yang sama memiliki kode *hash* yang sama.

```
a == b      : false
a.equals(b) : true
same hash   : true
distinct customers: 2
visits by Sari: 3
```

Keluaran `main` milik `Customer`: sama tetapi tidak identik, hash yang sama, dua pelanggan berbeda di dalam himpunan, dan tiga kunjungan Sari terhitung bersama. Python mengembalikan `NotImplemented` untuk tipe asing. Buku: Subbab 6.9.2.

Cara Objek Menggambarkan Dirinya

Java memanggil `toString()` untuk `println(x)` dan `"…" + x`; bawaannya, `PlainCustomer@1b6d3586`, tidak berarti apa-apa bagi manusia. Konvensinya: `Customer[name=Sari, phone=08125550101]`.

Python membagi tugasnya: `__repr__`, tak ambigu dan untuk pengembang, `Customer(name='Sari', phone='08125550101')`; `__str__`, ramah dan untuk pengguna, `Sari (08125550101)`.

Record Java atau `@dataclass(frozen=True)` Python menuliskan semuanya untuk Anda — tetapi membandingkan *setiap field*.

> **JEBAKAN.** Jangan pernah memasukkan rahasia ke dalam `toString` atau `__repr__`: keduanya berakhir di berkas log dan laporan galat (UU PDP 27/2022).

List mencetak butir-butirnya dengan `__repr__`. Java hanya mempunyai `toString`, jadi bentuk untuk pelanggan mendapat metodenya sendiri, misalnya `receiptLine()`. Buku: Subbab 6.9.2 dan 6.10.

Satu Nama, Beberapa Daftar Parameter

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.

Kedua konstruktor `BankAccount` adalah *overload*; `println` mempunyai sepuluh. **Tipe kembalian bukan bagian dari *signature***: `javac` menolak dua metode yang hanya berbeda di situ.

Dua metode `average`, untuk rekening dan untuk negara, berbagi nama — dan tubuh yang nyaris sama. Overloading membuat kita menamainya sekali, bukan menulis-nya sekali. Buku: Subbab 6.11.

Contoh Terkerjakan 2: Versi Mana yang Berjalan?

Kode 6.14: tiga versi `describe`, enam pemanggilan. Tebaklah setiap baris sebelum membacanya.

Kode 6.14 Tiga *overload* di `OverloadDemo.java`: versi mana yang dipilih setiap pemanggilan?

```
    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";
    }
```

```
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
```

Kecocokan persis lebih dahulu; lalu ***pelebaran*** *(widening):* `byte` → `int`, `long` → `double` *(sehingga* `3L` *menjadi* `3.0 kg`*),* `char` → `int` *(sehingga* `'A'` *menjadi 65 potong); lalu boxing; lalu varargs. Buku: Subbab 6.11.1.*

Python Tidak Mengenal *Overloading*

`def average` yang kedua sekadar **menggantikan** yang pertama, seperti `x = …` yang kedua; rekening-rekening lalu gagal ketika program berjalan.

Perkakas Python, pada Kode 6.15: argumen *default*, `*args`, dan `functools.singledispatch`, yang memilih menurut tipe argumen pertama saat program berjalan.

Kode 6.15 Dari `overload_demo.py`: *overloading* menurut tipe dengan `functools.singledispatch`.

```
@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"
```

`describe(True)` memilih versi `int`, karena `bool` adalah subkelas `int` — jebakan yang sama dengan `char` di Java. Lakukan overload hanya bila versi-versinya mengerjakan hal yang sama. Buku: Subbab 6.11.2.

Antarmuka

Apa yang sebenarnya dibutuhkan `average` dari setiap objek? Bukan bahwa ia rekening atau negara: hanya **sebuah bilangan untuk dijumlahkan**.

> **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*.

Kode 6.16 `Measurable.java`, tanpa komentarnya: antarmuka dengan satu metode dan tanpa kode.

```java
public interface Measurable {
    /**
     * Returns the number that represents this object when averaging.
     *
     * @return the measure
     */
    double getMeasure();      // tanpa badan: setiap kelas pengimplementasi wajib menyediakannya
}
```

Kode 6.16: metode antarmuka otomatis `public` dan `abstract`. Antarmuka tinggal di berkasnya sendiri, seperti kelas publik. Buku: Subbab 6.12.

Satu `average` untuk Setiap `Measurable`

Pada Kode 6.17 tipe parameternya adalah antarmuka, sehingga perulangan hanya boleh memanggil `getMeasure()` — hanya itu yang dibutuhkannya.

Kode 6.17 Dari `MeasurableDemo.java`: janji `Account` dipenuhi, dan satu `average` untuk semua.

```
    @Override
    public double getMeasure() {
        return balance;
    }
    public static double average(Measurable[] objects) {
        if (objects.length == 0) {
            return 0;
        }
        double sum = 0;
        for (Measurable obj : objects) {     // obj bisa berupa Account atau Country
            sum = sum + obj.getMeasure();
        }
        return sum / objects.length;
    }
```

*Ketika `obj` sebuah `Account`, versi `Account`-lah yang berjalan; ketika sebuah `Country`, versi `Country`. Pilihan saat program berjalan itulah **pengiriman dinamis** (dynamic dispatch), jantung polimorfisme (Bab 7). Buku: Subbab 6.12.*

Rata-rata, dan Janji yang Dilanggar

Perulangan itu menghitung rata-rata biasa Persamaan (6.3); untuk tiga rekening, $(100+150+125)/3=125$.

$$\acute{x}=\frac{1}{n}\sum_{i=1}^{n} x_i \tag{6.3}$$

Hapus `getMeasure` dari `Country`, dan kompilator menangkapnya sebelum program sempat berjalan:

```
MeasurableDemo.java:20: error: Country is not abstract and does not override abstract method
getMeasure() in Measurable
class Country implements Measurable {
^
1 error
```

`new Measurable()` juga ditolak: antarmuka adalah kontrak, bukan benda. Variabel tetap boleh bertipe antarmuka: `Measurable y = new Account(…)`. Buku: Subbab 6.12.

Python: *Protocol*, Diperiksa Menurut Bentuk

Pada Kode 6.18, `Account` dan `Country` tidak pernah menyebut `Measurable`, namun `isinstance(accounts[0], Measurable)` bernilai `True`: keduanya mempunyai metodenya.

Kode 6.18 Dari `measurable_demo.py`: *protocol* `Measurable` dan satu `average`.

```
@runtime_checkable
class Measurable(Protocol):
    """Anything that can be reduced to one number for averaging."""

    def get_measure(self) -> float:
        ...
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)
```

`...` *berarti "tidak ada tubuh di sini". Python mengabaikan anotasi* `Sequence[Measurable]` *ketika berjalan; perkakas seperti mypy membacanya. Buku: Subbab 6.12.*

Menukar Representasi

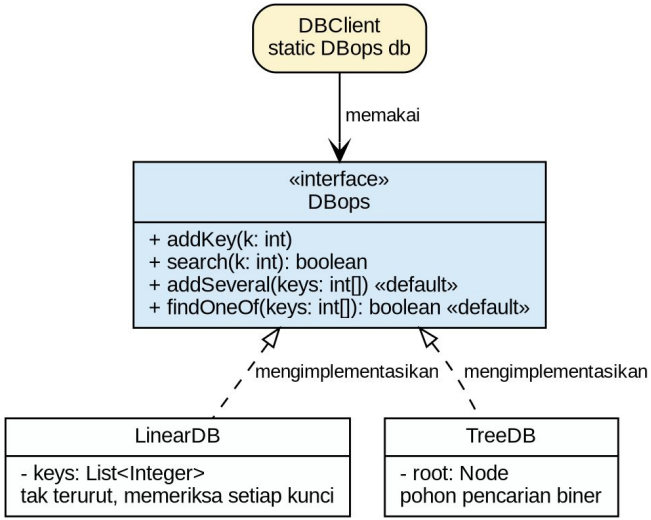
Tabel kecil berisi kunci bilangan bulat dapat tinggal di list tak terurut (`LinearDB`) atau pohon pencarian biner (`TreeDB`); klien memilih saat mulai.

Tanpa antarmuka: dua variabel, salah satunya selalu `null`, dan sebuah penanda yang diuji pada setiap operasi. Representasi ketiga berarti cabang ketiga di setiap tempat itu.

Dengan antarmuka: satu variabel bertipe `DBops`, dipilih sekali; setiap pemanggilan adalah pemanggilan biasa, dan pengiriman dinamis menjalankan kode yang tepat.

*`DBops` adalah **supertipe** kedua kelas; `db = new TreeDB()` dikonversi ke atas secara otomatis, karena `TreeDB` menawarkan semua yang dijanjikan `DBops`. Buku: Subbab 6.13.1–6.13.3.*

Klien Hanya Bergantung pada Antarmuka



Gambar 6.5 Klien hanya bergantung pada antarmuka `DBops`; `LinearDB` dan `TreeDB` mengimplementasikannya dan dapat dipertukarkan tanpa mengubah klien.

Gambar 6.5 dalam UML: «interface» di atas sebuah nama menandai antarmuka; garis putus-putus dengan segitiga kosong berarti "mengimplementasikan"; panah biasa berarti "memakai". Buku: Subbab 6.13.2.

Metode *Default*: Kode Bersama di Antarmuka

Kode 6.19: `addSeveral` dan `findOneOf` ditulis **sekali**, dari dua metode abstrak, dan diwarisi kedua kelas.

Kode 6.19 Dari antarmuka `DBops` di `dbiface/DBClient.java`: dua metode abstrak dan satu metode *default*.

```
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);
        }
    }
```

`findOneOf` adalah metode default kedua, sebuah perulangan atas `search`. Antarmuka tidak mempunyai field: ia menggambarkan perilaku, bukan keadaan. Python menulis antarmuka yang sama sebagai kelas dasar abstrak, `DBops(ABC)`, dan `DBops()` melempar `TypeError`. Buku: Subbab 6.13.4.

`Comparable`: Hanya Tandanya yang Dijanjikan

`Arrays.sort` bertanya "haruskah ini mendahului itu?" melalui `compareTo`, yang maknanya Persamaan (6.4):

$$\text{sign}(\texttt{a.compareTo(b)}) = \begin{cases} -1 & \text{jika } a \text{ datang sebelum } b \\ 0 & \text{jika urutan keduanya setara} \\ +1 & \text{jika } a \text{ datang sesudah } b \end{cases} \tag{6.4}$$

`"Ace".compareTo("Zonk")` bernilai -25, bukan -1. Karena itu `switch` yang hanya mempunyai kasus $-1, 0, 1$ keliru.

`List`, `Iterable`, dan `AutoCloseable` adalah antarmuka yang sudah Anda andalkan: `List<Order> orders = new ArrayList<>()`, perulangan for-each, `try`-with-resources. Buku: Subbab 6.14 dan 6.14.1.

Contoh Terkerjakan 3: Pengurutan yang Berbohong

Kode 6.20: `SwitchName` membiarkan `result` tetap `Integer.MIN_VALUE` untuk ±24, sehingga setiap Bogdanov "mendahului" Zonk *sekaligus* menyusulnya.

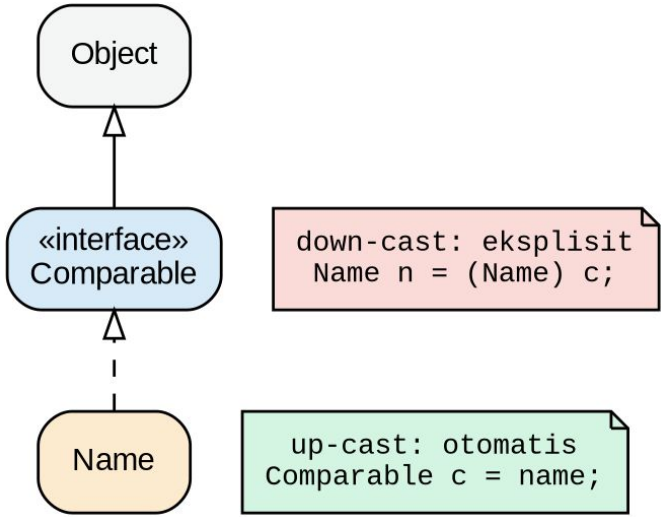
Kode 6.20 `switch` di dalam `SwitchName.compareTo` di `NameSortDemo.java`: memercayai `String.compareTo` hanya mengembalikan −1, 0, atau 1.

```
        switch (lastName.compareTo(o.lastName)) {
            case -1:
                result = -1;
                break;
            case 1:
                result = 1;
                break;
            case 0:
                result = firstName.compareTo(o.firstName);
                break;
        }
```

```
[Ace, Portgas D.]
[Bogdanov, Aleksandra]
[Zonk, Godot]
[Bogdanov, Alexei]
```

`Name` yang benar hanya melihat apakah hasil perbandingannya nol: jika nama belakangnya berbeda, hasil perbandingan itulah jawabannya, berapa pun nilainya. Dengan tiga nama bug itu bersembunyi; dengan empat ia tampak. Buku: Subbab 6.14.1.

Cast ke Atas dan ke Bawah



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.

Gambar 6.6: cast ke atas otomatis; cast ke bawah, `(SwitchName) other`, diperiksa saat berjalan dan dapat melempar `ClassCastException`. `Comparable<Name>` yang generik tidak memerlukan cast. Python mengurutkan dengan `__lt__` atau fungsi `key=`. Buku: Subbab 6.14.1.

Pendengar: Tombol yang Hanya Mengenal Antarmuka

Tombol menyimpan daftar pendengar dan, ketika diklik, memanggil masing-masing (Kode 6.21); `BouncingBall`, `Pendulum`, dan `RotatingSpiral` cukup menulis `implements StopListener`.

Kode 6.21 Dari `StopButtonDemo.java`: antarmuka pendengar dan tombol yang hanya mengenalnya.

```
/** Anything that wants to hear when the Stop button is pressed. */
interface StopListener {
    void stopRequested(String 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);
    }
```

Inilah gagasan di balik `ActionListener` dan setiap tombol Swing (Bab 10). Di Python, fungsi biasa pun dapat menjadi pendengar, dan itulah yang dilakukan `command=` di Tkinter. Buku: Subbab 6.14.2.

Empat Cara Menyatakan Kontrak

Tabel 6.4 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

Tabel 6.4: ABC gagal dengan lantang ketika objek yang tak lengkap dibuat; Protocol diperiksa oleh pemeriksa tipe seperti mypy sebelum program berjalan. Buku: Subbab 6.15.

Penentuan Harga di Profitina Laundry

Cuci lipat, cuci setrika: per kilogram yang sudah dimulai, minimum 3 kg. Cuci kering: per potong. Sepatu: satu harga tetap.

Ekspres: 50% lebih mahal, apa pun layanannya. Anggota: potongan 10%, apa pun layanannya.

> **WAWASAN KUNCI.** *Memprogramlah terhadap antarmuka, bukan terhadap implementasi.* Kode yang membutuhkan harga meminta "sesuatu yang dapat menghargai sebuah pesanan" — maka cara-cara baru menentukan harga adalah kelas-kelas baru, dan kode yang memakai harga tidak pernah berubah.

Bulan depan promosi musim hujan, tahun depan harga untuk hotel: dengan satu `if` per aturan, setiap aturan baru berarti menyunting kode nota. Buku: Subbab 6.16.

Kontraknya: `PriceRule`

Kode 6.22: dokumentasinya adalah bagian dari kontrak — rupiah bulat, tidak pernah negatif, penolakan dengan `IllegalArgumentException`.

Kode 6.22 Antarmuka `PriceRule.java`, tanpa metode statis pemformatnya.

```java
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();
```

`PriceRule.rupiah(28000)` menghasilkan `Rp 28.000`. Di Python `PriceRule` adalah kelas dasar abstrak, sehingga aturan yang tak lengkap gagal saat itu juga ketika dibuat. Buku: Subbab 6.16.2.

Aturan Per Kilogram

Tarif r, minimum m, berat w: Persamaan (6.5), dengan $\lceil 3.2 \rceil = 4$; Kode 6.23 adalah rumus itu baris demi baris.

$$P_{\text{kilo}} = r \cdot \max(m, \lceil w \rceil) \tag{6.5}$$

Kode 6.23 Metode `priceOf` di `PerKiloRule.java`: harga per kilogram yang sudah dimulai, dengan minimum.

```
    @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;
    }
```

$w = 2.0$: $\max(3, 2) = 3$, *jadi* $21\,000$; $w = 3.2$: $28\,000$ — *baris-baris tabel kasus uji.* `PerItemRule` *dan* `FlatRule` *nyaris remeh, dan itu justru keunggulan. Buku: Subbab 6.16.3.*

Aturan yang Membungkus Aturan Lain

Ekspres dan anggota memegang `PriceRule` lain, bertanya kepadanya, lalu menyesuaikan: pola **dekorator**. Ekspres adalah Persamaan (6.6) (Kode 6.24), anggota adalah Persamaan (6.7).

$$P_{\text{ekspres}} = P + \left\lfloor \frac{P}{2} \right\rfloor \tag{6.6}$$

$$P_{\text{anggota}} = P - \left\lfloor \frac{p \cdot P}{100} \right\rfloor \tag{6.7}$$

Kode 6.24 Metode `priceOf` di `ExpressRule.java`: dekorator yang menambahkan 50%, dibulatkan ke bawah.

```
    @Override
    public long priceOf(Order order) {
        long normal = base.priceOf(order);
        return normal + normal / 2;          // pembagian bilangan bulat: setengah rupiah dibuang
    }
```

P *ganjil* $= 21\,001$: $21\,001 + 10\,500 = 31\,501$. *Anggota: tulislah* `normal * percent / 100`; `normal * (percent / 100)` *menghitung* $10/100 = 0$ *dalam bilangan bulat dan tidak memberi potongan sama sekali. Buku: Subbab 6.16.4.*

Contoh Terkerjakan 4: Rantai Milik Budi

3,2 kg cuci lipat, ekspres, anggota: `new MemberDiscountRule(new ExpressRule(new PerKiloRule("wash & fold", 7_000, 3)), 10)`.

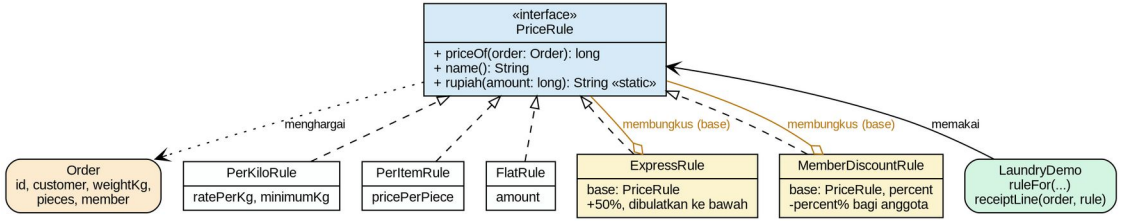
$$7\,000 \cdot \max(3, \lceil 3.2 \rceil) = 28\,000 \xrightarrow{\text{ekspres}} 42\,000 \xrightarrow{\text{anggota}} 37\,800 \tag{6.8}$$

Tabel 6.5 Kelima aturan harga Profitina Laundry.

Aturan	Harga	Menolak
`PerKiloRule(name, r, m)`	Persamaan (6.5)	pesanan tanpa berat
`PerItemRule(name, price)`	harga × jumlah potong	pesanan tanpa potongan
`FlatRule(name, amount)`	jumlah tetap	tidak ada
`ExpressRule(base)`	Persamaan (6.6) atas harga dasar	apa pun yang ditolak aturan dasarnya
`MemberDiscountRule(base, p)`	Persamaan (6.7) untuk anggota, harga dasar untuk yang lain	p di luar $0..100$ (ketika dibangun)

Dari dalam ke luar (Persamaan (6.8)): $7\,000 \cdot 4$*, lalu* $+\,14\,000$*, lalu* $-\,4\,200$*. Tabel 6.5 memberi harga setiap aturan dan apa yang ditolaknya. Buku: Subbab 6.16.4.*

Rancangan Penentuan Harga



Gambar 6.7 Rancangan penentuan harga: lima kelas mengimplementasikan `PriceRule`, dua di antaranya dengan membungkus `PriceRule` lain; program demo hanya bergantung pada antarmuka.

Gambar 6.7: lima kelas mengimplementasikan `PriceRule`, dua di antaranya dengan membungkus `PriceRule` lain; program demo hanya bergantung pada antarmuka. Buku: Subbab 6.16.4.

Nota Satu Pagi

`ruleFor` adalah *pabrik* (*factory*) kecil, satu-satunya kode yang menyebut kelas-kelas aturan; `receiptLine` hanya mengenal antarmuka (Kode 6.25).

Kode 6.25 Metode `receiptLine` di `LaundryDemo.java`: bekerja untuk setiap aturan yang ada dan setiap aturan yang ditulis besok.

```
    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)));
    }
```

```
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
```

A002: $40\,000 \to 60\,000 \to 54\,000$. *A006 (cuci kering tanpa potongan pakaian) ditolak, dan penolakan itu lewat tanpa berubah melalui pembungkus anggota. Totalnya* $263\,000$. *Buku: Subbab 6.16.5.*

Menguji Setiap Aturan

Setiap kelas pada tabel kasus uji, kedua sisi setiap batas, setiap penolakan, pembulatan dengan harga **ganjil**, dan tumpukan aturan (Kode 6.26).

Kode 6.26 Dari `test_price_rules.py`: uji batas yang berparameter dan uji dengan aturan pengganti.

```
@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_express_adds_half_rounding_down():
    odd = FlatRule("odd", 21_001)            # pengganti dengan harga ganjil
    assert ExpressRule(odd).price_of(by_pieces(1)) == 31_501
```

`@pytest.mark.parametrize` menjalankan satu tubuh sekali per baris. `FlatRule("odd", 21_001)` adalah stub: melalui antarmuka, `ExpressRule` tidak dapat membedakannya dari aturan sungguhan, sehingga ekspres teruji sendirian. Buku: Subbab 6.16.6.

Apa yang Kita Peroleh dari Antarmuka

Promosi musim hujan, potongan 5% di bulan Januari: satu kelas baru `SeasonalRule` yang membungkus aturan dasar, satu baris di `ruleFor`, dan beberapa uji.

`Order`, `PriceRule`, kelima aturan, `receiptLine`, dan setiap uji lama tidak tersentuh — dan uji-uji lama terus membuktikan harga-harga lama.

Dokumentasi menyatakan janji setiap aturan, *uji* memeriksa bahwa setiap aturan menepatinya, *antarmuka* membuat program dapat bersandar padanya.

Itulah seluruh bab dalam satu halaman. Soal Latihan 6.11 menambahkan potongan borongan semacam ini. Buku: Subbab 6.16.7.

Profitina dalam Praktik

Satu antarmuka, banyak sinyal: setiap sinyal kesiapan adalah kelas di balik "diberikan situs yang dipindai, hasilkan skor dan penjelasannya"; sinyal baru adalah kelas baru.

Pengganti untuk uji, tidak pernah untuk pengguna: kode yang memerlukan model lokal (Qwen2.5 di Ollama) memakai antarmuka yang sempit; uji memasang jawaban tetap, produksi terus terang ketika model tidak tersedia.

Uji sebagai gerbang rilis: jauh lebih dari seribu uji harus hijau, dan rilis baru terhitung selesai ketika versi baru lolos pemeriksaan kesehatannya.

Kesamaan: `https://Example.com/` dan `https://example.com` harus menjadi satu halaman — nomor telepon `Customer` yang dinormalkan dalam kostum lain.

FastAPI membangun halaman API dari anotasi dan docstring — gagasan `pydoc` yang sudah dewasa. Buku: Subbab 6.17.

Kesalahan yang Sering Terjadi

Dokumentasi: komentar yang diakhiri `/`; nama `@param` yang tak lagi cocok; menggambarkan implementasi, bukan kontraknya.

Uji: menghitung ulang jawaban dengan rumus yang sama; uji yang saling bergantung; hanya kasus yang menyenangkan; `==` pada `double`; nilai harapan dan nilai sebenarnya tertukar.

Kesamaan: `equals` tanpa `hashCode`; `equals(Customer other)`; `==` pada string; `toString` bawaan, atau rahasia di dalamnya.

***Overloading* dan antarmuka:** satu nama untuk makna berbeda; menganggap `compareTo` mengembalikan ± 1; mendeklarasikan `ArrayList` atau `TreeDB` padahal `List` atau `DBops` cukup; membulatkan uang di tempat yang salah.

Buku mendaftar empat belas kesalahan, masing-masing dengan cara menghindarinya. Buku: Subbab 6.18.

Ringkasan: Lima Hal untuk Diingat

1. Dokumentasikan setiap butir publik dengan Javadoc atau *docstring*; biarkan `javadoc`, `pydoc`, dan `doctest` membangun dan memeriksanya.
2. Uji unit adalah *arrange–act–assert* terhadap literal yang dihitung sebelumnya; pilih masukan menurut kelas dan batas; bandingkan `double` dengan toleransi (Persamaan (6.1)).
3. Identitas bukan kesamaan: *override* `equals` dan `hashCode` bersama-sama, dari *field* yang sama (Persamaan (6.2)).
4. *Overload* dipilih saat kompilasi: kecocokan persis, pelebaran, *boxing*; Python memakai argumen *default*, `*args`, dan `singledispatch`.
5. Antarmuka adalah janji tanpa tubuh; memprogramlah terhadapnya, dan perilaku baru menjadi kelas baru.

Harga di Profitina Laundry kini lima kelas di balik `PriceRule`, masing-masing teruji sendirian. Buku: Subbab 6.19.

Pertemuan Berikutnya

Bab 7 — Pewarisan, Polimorfisme, dan Kontrol Akses. Kelas yang memperluas kelas, *overriding*, pengiriman dinamis dijelaskan, kelas abstrak, dan siapa boleh melihat apa.

`Measurable` dan `DBops` hari ini menjadi hierarki kelas dengan keadaan bersama.

Sebelum itu: kerjakan Soal Latihan 6.1, 6.4, 6.7, dan 6.11, di Java dan di Python.

Pengiriman dinamis, yang hari ini membuat `average` bekerja untuk setiap `Measurable`, adalah jantung Bab 7. Buku: Subbab 6.20 dan 6.21.

Kuliah 7 — Pewarisan, Polimorfisme, dan Kontrol Akses

Pemrograman Berorientasi Objek (EF234302)

Buku: Irfan Subakti, *Pemrograman Berorientasi Objek dengan Java dan Python*, Bab 7

Dosen: Irfan Subakti — yifana@gmail.com

Hari ini satu baris, `s.price()`, belajar menghargai cuci, setrika, cuci kering, dan pesanan kilat tanpa menanyakan yang mana. Buku: Bab 7, Subbab 7.1–7.18.

Cakupan Pertemuan Ini

1. **Pewarisan** — masalah duplikasi, superkelas dan subkelas, *overriding*, konstruktor dan `super`.
2. **Polimorfisme** — satu panggilan, banyak perilaku; tipe statis dan dinamis; tabel metode; MRO Python.
3. **Merancang hierarki** — kelas abstrak, superkelas konkret, komposisi, prinsip substitusi.
4. **Pewarisan jamak** — antarmuka Java dan *default* yang bertentangan; berlian Python dan C3.
5. **Kontrol akses dan enkapsulasi** — empat tingkat akses Java, konvensi Python, invarian, *property*.
6. **Profitina Laundry** — hierarki layanan yang dihargai oleh satu perulangan polimorfis.

Urutannya sama dengan yang dialami pemrogram yang bekerja: sebuah masalah, obatnya, kejutan yang dibawa obat itu, dan disiplin yang menjaga hierarki tetap sehat. Buku: Subbab 7.1.

Di Mana Kita Berhenti: Kelas Penuh Kasus Khusus

`Service` di Bab 3 menjawab "berapa lama?" dengan `switch` pada kodenya sendiri, dan menghargai kilat dengan sebuah `if`. Bab 6 memberi kita antarmuka dan *overloading*.

Tambahkan cuci kering (per potong), sepatu (per pasang), promosi 20%: masing-masing berarti membuka lagi `Service`, satu `case` lagi, satu `if` lagi. Setahun kemudian, seribu baris yang tak berani disentuh siapa pun.

> **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*.

Segala isi bab ini adalah rincian dari salah satu ketiga kalimat itu. Buku: Subbab 7.1.

Dua Basis Data yang Saling Mengulang

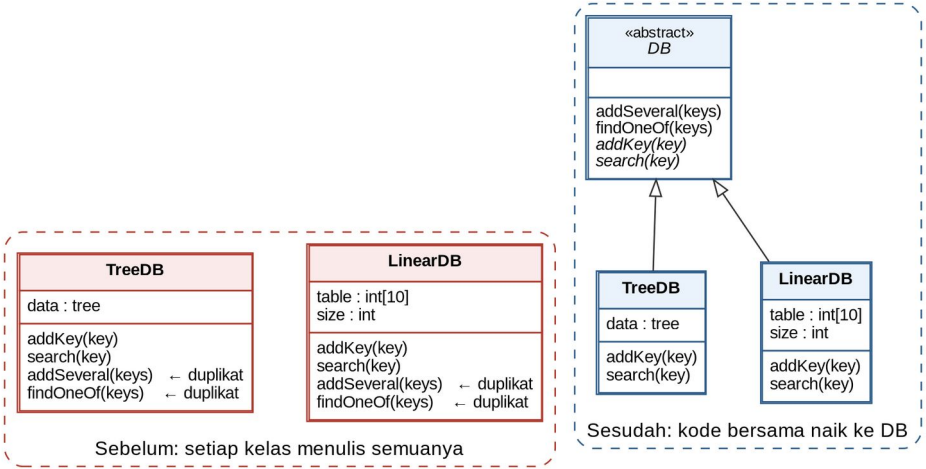
`LinearDB` menyimpan kunci bilangan bulat dalam larik; `TreeDB` dalam pohon pencarian biner. Keduanya memerlukan empat operasi yang sama:

```
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?
```

`addSeveral` memanggil `addKey` sekali per kunci; `findOneOf` memanggil `search` sampai ada yang berhasil. Keduanya tidak peduli **bagaimana** kunci disimpan.

Bila ditulis sebagai dua kelas yang tak berhubungan, kedua metode itu ada dua kali, huruf demi huruf — dan pada hari sebuah bug dibetulkan di satu salinan, kedua kelas itu diam-diam berselisih. Buku: Subbab 7.2.

Masalahnya dan Obatnya



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.

Gambar 7.1: metode-metode bersama naik ke `DB`, yang juga mendeklarasikan dua metode yang kodenya berbeda. Gagasan ini berasal dari Simula 67. Buku: Subbab 7.2.

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.

Ucapkan keras-keras: ***is-a***. Sebuah `LinearDB` *adalah* sebuah `DB`. Bila kalimatnya janggal — "sebuah `Customer` adalah sebuah `Service`" — pewarisannya keliru, sebanyak apa pun kode yang dihematnya.

Pewarisan bersifat **transitif**: bila `D extends C` dan `C extends B`, maka `D` memiliki segala milik `C` dan `B`.

Model mental yang berguna: subkelas berperilaku seakan-akan anggota superkelasnya disalin ke dalamnya. Sebenarnya tak ada kode yang disalin; Subbab 7.7 memperlihatkan yang terjadi. Buku: Subbab 7.2.1 dan 7.3.1.

Superkelas Abstrak `DB`

Kode 7.1: `addSeveral` mempunyai badan biasa yang memanggil `addKey` — yang hanya *dideklarasikan* oleh `DB`.

Kode 7.1 Dari `db/DBTest.java`: kelas abstrak `DB`, dengan satu metode lengkap dan dua metode abstrak.

```
abstract class DB {
    void addSeveral(int keys[]) {
        for (int i = 0; i < keys.length; i++) {
            addKey(keys[i]);
        }
    }
    abstract void addKey(int key);     // Tanpa kode
    abstract boolean search(int key);  // Tanpa kode
}
```

`abstract` berarti "belum lengkap: Anda tidak boleh membuat objeknya". `findOneOf` ditulis dengan cara yang sama, sebuah perulangan yang memanggil `search`. Buku: Subbab 7.3.

Sebuah Subkelas: `LinearDB extends DB`

Kode 7.2 menambahkan dua field dan menyediakan dua badan yang kurang; di sini keduanya hanya melaporkan bahwa mereka dijalankan.

Kode 7.2 Dari `db/DBTest.java`: subkelas `LinearDB`.

```
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;
    }
}
```

`DB ldb = new LinearDB();` — variabel bertipe `DB`, objek berkelas `LinearDB`. `ldb.addSeveral(new int[] {3, 4})` menjalankan perulangan milik `DB`, yang mencetak `LinearDB: addKey()` dua kali: kode di superkelas memanggil metode yang dipilih oleh objeknya. Buku: Subbab 7.3.

Proyek yang Sama dalam Python

Kode 7.3: `class LinearDB(DB):` — tanpa `extends`; `ABC` dan `@abstractmethod` memberi Python gagasan "kelas yang belum lengkap".

Kode 7.3 Dari `db_test.py`: `DB` mewarisi `ABC` dan menandai `add_key` serta `search` sebagai abstrak.

```
class DB(ABC):
    def add_several(self, keys):
        for key in keys:
            self.add_key(key)
    @abstractmethod
    def add_key(self, key):  # Tanpa kode
        ...

    @abstractmethod
    def search(self, key):  # Tanpa kode
        ...
```

```
DB(): Can't instantiate abstract class DB with abstract methods add_key, search
```

Java menolak `new DB()` saat kompilasi ("DB is abstract; cannot be instantiated"); Python menolak `DB()` saat program berjalan, dengan `TypeError` di atas. Field dibuat dengan menugasi `self.table` di `__init__`. Buku: Subbab 7.3.2.

Overriding: Mendefinisikan Ulang Metode Warisan

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`.

Kode 7.4: basis data berbasis larik dapat menambahkan banyak kunci dengan satu penyalinan sekaligus.

Kode 7.4 Dari `IntDBDemo.java`: `LinearIntDB` meng-*override* `addSeveral` warisannya; `TreeIntDB` mewarisinya tanpa perubahan.

```
    // 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)");
    }
```

*Selalu tulis `@Override`: ia mengubah nama yang salah eja atau tipe parameter yang keliru menjadi galat kompilasi. Di Python sebuah metode meng-override bila **namanya** sama; parameternya tidak berpengaruh. Buku: Subbab 7.4.*

Contoh Terkerjakan 1: Menghitung Perbandingan

Kedua basis data menerima $\langle 42, 7, 19, 3, 25 \rangle$ dan pencarian yang sama; keduanya menghitung perbandingan:

```
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
```

Dengan tangan, larik: `search(19)` memeriksa 42, 7, 19: 3. `search(8)`: 5. `findOneOf({8,9,25})`: 5+5+5=15. `findOneOf({1,2})`: 10. Totalnya 3+5+15+10=33.

`(bulk copy of 5 keys)` *menandakan override-nya berjalan. Pohon memberi jawaban yang sama dengan 22 perbandingan: setiap pencarian menyusuri satu jalur akar-ke-daun paling banyak empat simpul (Soal Latihan 7.2). Buku: Subbab 7.4.*

Aturan yang Diperiksa Java Saat *Override*

1. **Tipe kembalian:** sama, atau subtipenya (*kovarian*): `Garment make()` boleh menjadi `Shirt make()`.
2. **Akses:** tidak boleh lebih sempit — metode `public` tidak dapat di-*override* oleh metode *package-private* (“attempting to assign weaker access privileges”).
3. **`final`:** metode `final` tidak dapat di-*override*; kelas `final` (seperti `String`) tidak dapat diperluas.
4. **Metode `static` dan `private`** tidak di-*override*: metode subkelas *menyembunyikan* yang pertama dan tak berhubungan dengan yang kedua.
5. **Eksepsi terperiksa** boleh menyusut, tidak boleh bertambah.

Setiap aturan melindungi janji yang dibuat superkelas kepada pemanggilnya. Python tidak menegakkan satu pun; kesalahannya baru muncul ketika program berjalan. Buku: Subbab 7.4.1.

Konstruktor Berantai ke Atas

Bagian superkelas dibangun **lebih dahulu**, oleh konstruktor superkelas itu sendiri. Kode 7.5: `Shirt` adalah `Garment` adalah `LaundryItem`.

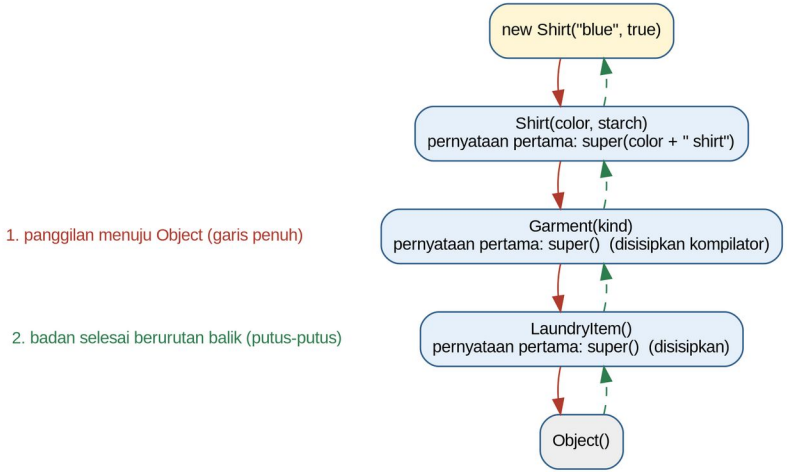
Kode 7.5 Kelas `Shirt` di `ConstructorChain.java`: `super(...)` lebih dahulu, dan `super.describe()` untuk memperpanjang teks induknya.

```
    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
```

Kita meminta `Shirt`, tetapi `LaundryItem` berbicara lebih dahulu. `super.describe()` berarti "kerjakan apa yang dikerjakan induk saya, lalu sedikit lagi": `item #1`, `, a blue shirt`, `, starched`. Buku: Subbab 7.5.

Rantai itu dalam Gambar



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.

Gambar 7.2: konstruktor `Garment` diawali `super();` tak terlihat yang disisipkan kompilator, demikian pula `LaundryItem`, sampai ke `Object`; lalu badan-badannya selesai dalam urutan terbalik. Buku: Subbab 7.5.

Tiga Aturan Konstruksi

1. **`super(...)` harus pertama**, bila ditulis: tak ada yang boleh terjadi pada `Shirt` sebelum bagian `Garment`-nya ada.
2. **Tanpa `super(...)`, kompilator menulis `super()`** — dan panggilan itu harus sah. Hapus `super(color + " shirt")` dan `javac` melaporkan bahwa `Garment` "cannot be applied to given types; required: String, found: no arguments".
3. **Konstruktor tidak diwariskan.** Setiap kelas menyatakan cara objeknya sendiri dibangun.

> **PYTHON TIDAK PERNAH MENYISIPKAN PANGGILAN ITU.** Pada Kode 7.6, `ForgetfulGarment` tidak pernah menjalankan `LaundryItem.__init__`, sehingga objeknya tidak mempunyai `tag`:

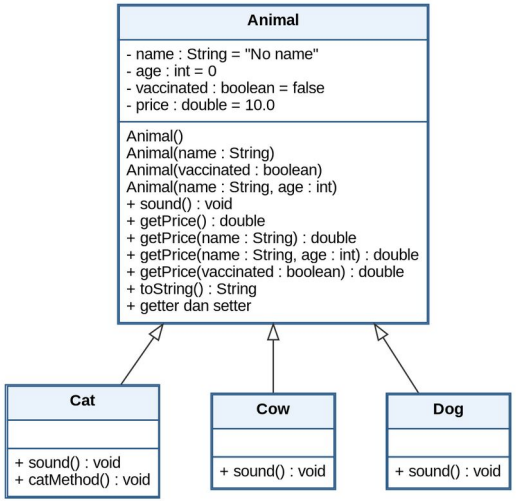
Kode 7.6 Dari `constructor_chain.py`: subkelas yang lupa memanggil `super().__init__()`.

```
class ForgetfulGarment(LaundryItem):
    def __init__(self, kind):
        self.kind = kind  # lupa super().__init__(): tag tidak pernah diisi
```

```
-- ForgetfulGarment("sock") --
  AttributeError: 'ForgetfulGarment' object has no attribute 'tag'
```

Galatnya muncul jauh dari penyebabnya. Kebiasaannya: baris pertama setiap `__init__` subkelas adalah `super().__init__(...)`. Buku: Subbab 7.5.

Polimorfisme: Program `Animal`



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.

Gambar 7.3: `Animal` mempunyai empat konstruktor dan empat metode `getPrice` yang di-overload; `Cat`, `Cow`, dan `Dog` meng-override `sound()`, dan `Cat` menambahkan `catMethod()`. Buku: Subbab 7.6.

Satu Panggilan, Banyak Perilaku

Pada Kode 7.7 keempat variabel dideklarasikan `Animal`. Keempat panggilannya tampak sama persis:

Kode 7.7 Dari `animal/AnimalTest.java`: empat panggilan yang sama, dan `Cat` yang meng-*override* `sound()`.

```
        // Polimorfisme
        animal.sound();
        cat.sound();
        cow.sound();
        dog.sound();
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");
    }
```

```
Animal makes a sound
Cat says: meow
Cow says: moo
Dog says: woof
```

*Kompilator tidak dapat memilih, karena ia hanya tahu bahwa setiap variabel adalah `Animal`; program yang sedang berjalanlah yang melihat objeknya. Itulah **polimorfisme waktu jalan**. Buku: Subbab 7.6.1 dan 7.6.2.*

Apa yang Diizinkan Kompilator

`Animal c1 = new Cat();` — objeknya mempunyai `catMethod()`, tetapi `c1.catMethod()` tidak dapat dikompilasi:

```
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
```

Kompilator bernalar dari tipe yang dideklarasikan. `((Cat) c1).catMethod()` berhasil — diperiksa saat program berjalan; `((Cat) a1)` pada `Animal` biasa melempar `ClassCastException`.

Python tidak punya apa pun untuk di-cast: `c1.cat_method()` langsung berhasil, dan `a1.cat_method()` baru gagal saat dipanggil, dengan `AttributeError` — duck typing. Buku: Subbab 7.6.2 dan 7.6.3.

Overloading versus *Overriding*

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

Tabel 7.1: kompilator memilih overload; objek yang berjalan memilih override. Python mengganti overloading dengan argumen default dan argumen kata kunci: `Animal(vaccinated=True)`*. Buku: Subbab 7.6.3.*

Dua Tipe untuk Setiap Variabel

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.

`Machine m = new Washer("W-01");` adalah *upcasting*, selalu aman. **Kompilator** memakai tipe statis untuk memutuskan apakah panggilan *diizinkan*; **program yang berjalan** memakai tipe dinamis untuk memutuskan *override* mana yang dijalankan.

*Pembagian kerja ini adalah **pengiriman dinamis** (dynamic dispatch, late binding). Ia hanya berlaku untuk metode instans yang dapat di-override. Buku: Subbab 7.7.1.*

Contoh Terkerjakan 2: Tiga Kejutan

Kode 7.8: tebaklah setiap baris sebelum membacanya.

Kode 7.8 Dari `DispatchDemo.java`: sebuah metode, sebuah field, dan sebuah metode statis, semuanya dicapai melalui `m`.

```
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());
```

```
W-01: fill water, spin at 1200 rpm
m.label  = machine
m.kind() = Machine.kind()
m.describe() = W-01 is a machine
```

1. `start()` dikirim secara dinamis: baris milik *washer*. **2.** `Washer` mendeklarasikan ulang `label`: ia *menyembunyikan* `Machine.label`, dan tipe statis memilih `machine`. **3.** `kind()` bersifat `static`: terikat pada `Machine`.

Jangan pernah mendeklarasikan ulang field superkelas; panggillah metode statis melalui nama kelasnya. `x instanceof Dryer d` *menguji dan meng-cast sekaligus;* `getClass()` *melaporkan tipe dinamis. Buku: Subbab 7.7.1.*

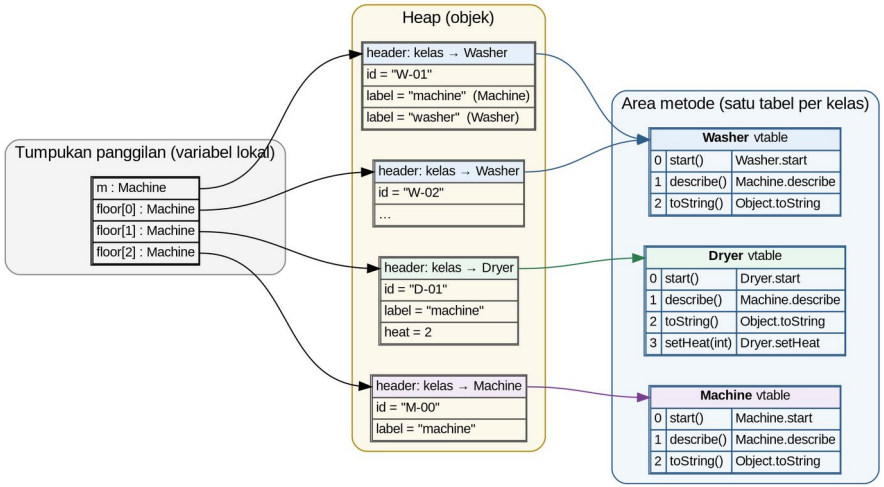
Apa yang Diputuskan Kapan

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 Kode 7.8
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

Tabel 7.2: hanya metode instans yang dapat di-override yang dikirim menurut tipe dinamis; field, metode `static` dan `private`, serta pilihan di antara overload semuanya ditetapkan kompilator. Buku: Subbab 7.7.1.

Gambaran Memori: Tabel Metode



Gambar 7.4 Memori selama Kode 7.8 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.

Gambar 7.4: referensi di stack, objek di heap yang kepalanya menunjuk kelasnya, dan satu vtable per kelas. `Washer` menyalin slot `Machine` lalu menimpa slot 0; `Dryer` juga menambahkan `setHeat`. Buku: Subbab 7.7.2.

Panggilan Adalah Tiga Pembacaan, Bukan Pencarian

Setiap metode yang dapat di-*override* mendapat **slot** tetap: `start()` 0, `describe()` 1, `toString()` 2. Panggilan `x.start()` menjadi Persamaan (7.1):

$$\text{kode yang dijalankan} = \text{vtable}\big(\text{kelas}(x)\big)[\text{slot}(\text{start})] \tag{7.1}$$

Ikuti `x` ke objeknya; kepala objek ke *vtable* kelasnya; lompat ke kode di slot yang ditetapkan kompilator. Dua kelas atau dua ratus: pembacaan yang sama sedikitnya.

Gambaran ini menjelaskan aturannya: field berada di posisi tetap yang dipilih kompilator; metode statis dan privat tidak pernah masuk *vtable*.

Spesifikasi JVM menetapkan perilaku `invokevirtual`, bukan struktur data ini; JVM sungguhan memakai vtable dan bahkan memanggil metode secara langsung bila sebuah tempat pemanggilan hanya pernah bertemu satu kelas. Buku: Subbab 7.7.2.

Python: Menyusuri MRO

Python memeriksa kamus milik objek itu sendiri, lalu setiap kelas pada `type(m).__mro__` secara berurutan (Kode 7.9).

Kode 7.9 Fungsi bantu `where_is` dari `dispatch_demo.py`: pencarian yang dilakukan Python, dalam enam baris.

```
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
```

```
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
```

> **WAWASAN KUNCI.** Java memutuskan “bolehkah saya memanggil ini?” dari tipe *statis* dan “versi mana?” dari tipe *dinamis*; Python memutuskan keduanya saat berjalan, dari objeknya. Bagaimanapun, **objeknyalah yang menentukan metode mana yang berjalan**.

Di Python `label` adalah atribut kelas yang ditemukan dengan penelusuran yang sama, jadi `m.label` mencetak `washer`: tidak ada jebakan field tersembunyi. Setiap kelas berakar pada `Object` / `object`, yang `toString`, `equals`, dan `hashCode`-nya memang untuk di-override. Buku: Subbab 7.7.3 dan 7.8.

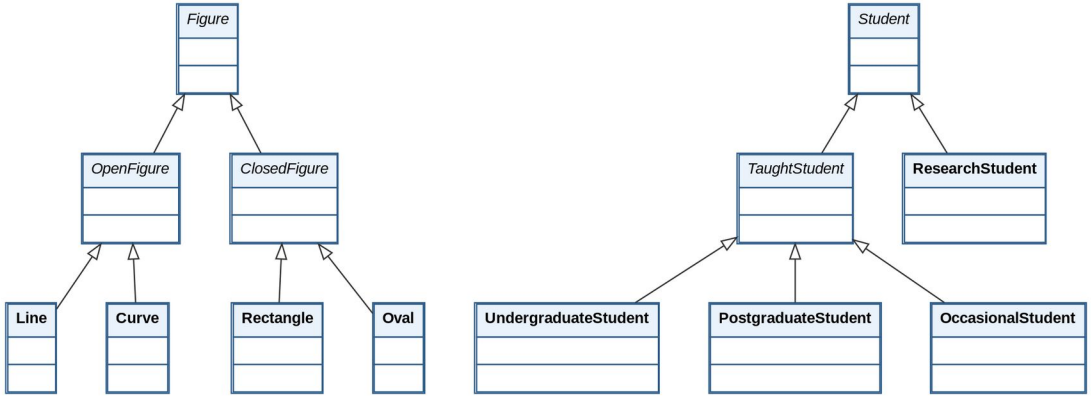
Kelas Abstrak

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 boleh memuat field, konstruktor, metode lengkap, dan metode abstrak sekaligus: **apa yang dimiliki bersama sebuah keluarga, termasuk kode**, dengan bagian yang berbeda diserahkan kepada subkelas.

Kelas abstrak berada di antara antarmuka, yang hanya memuat deklarasi metode, dan kelas konkret, yang hanya memuat metode lengkap. Buku: Subbab 7.9.

Hierarki dengan Tingkat Abstrak



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.

Gambar 7.5: bangun dibagi menjadi bangun terbuka dan tertutup; mahasiswa menjadi mahasiswa perkuliahan dan mahasiswa riset (Soal Latihan 7.4). Nama abstrak dicetak miring. Buku: Subbab 7.9.

Dua Rumus untuk Oval

Semisumbu a, b: luasnya Persamaan (7.2); kelilingnya tidak mempunyai rumus elementer, jadi kita memakai hampiran Ramanujan Persamaan (7.3).

$$A = \pi\, a\, b \tag{7.2}$$

$$P \approx \pi\left[3(a+b) - \sqrt{(3a+b)(a+3b)}\right] \tag{7.3}$$

Kode 7.10 Kedua metode `Oval` di `FigureDemo.java`.

```
    @Override
    double length() {  // aproksimasi keliling dari Ramanujan
        return Math.PI * (3 * (a + b) - Math.sqrt((3 * a + b) * (a + 3 * b)));
    }

    @Override
    double area() {
        return Math.PI * a * b;
    }
```

Kode 7.10 adalah kedua rumus itu baris demi baris. Persamaan (7.3) tepat untuk lingkaran: $a=b=r$ memberi $\pi\left[6r - \sqrt{16r^2}\right] = 2\pi r$. Buku: Subbab 7.9.

Contoh Terkerjakan 3: Periksa Bangun-bangunnya

`Figure.describe()` adalah **metode templat**: lengkap, tetapi memanggil `length()` yang abstrak. `ClosedFigure` menambahkan `area()` dan memperpanjang `describe()` dengan `super.describe()`.

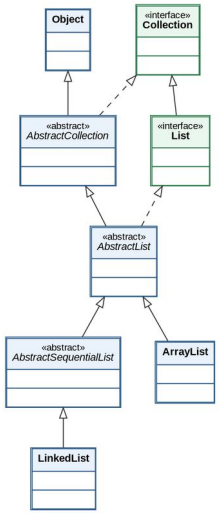
```
Line      length   5.000
Curve     length   2.828
Rectangle length  13.000   area  10.000
Oval      length  15.865   area  18.850
Total area of the closed figures: 28.850
```

Garis $(0,0)$–$(3,4)$: $\sqrt{3^2+4^2}=5$. Kurva: $2\sqrt{2}\approx 2.828$. Persegi panjang: $2(4+2.5)=13$, luas 10.

Oval 3×2: luas $6\pi\approx 18.850$; keliling $\pi\left(15-\sqrt{99}\right)\approx 15.865$. Luas bangun tertutup: $10+18.850=28.850$.

`Locale.US` menjaga titik desimal di setiap komputer; `new Figure("?")` tidak dapat dikompilasi. Buku: Subbab 7.9.

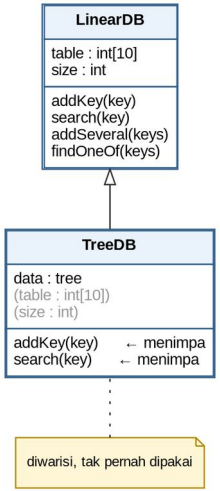
Pustaka Java Dibangun dengan Cara yang Sama



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.

Gambar 7.6: `AbstractList` mengimplementasikan `contains`, `indexOf`, `iterator`, dan lainnya dari `get(int)` dan `size()` saja. List jenis baru menulis dua metode dan mewarisi puluhan. Buku: Subbab 7.9.1.

Mewarisi Kelas Konkret



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`.

*Gambar 7.7: `TreeDB extends LinearDB` berjalan, tetapi membawa larik yang tak terpakai (**beban**), membuat `LinearDB x = new TreeDB()` lolos kompilasi (**janji palsu**), dan bergantung pada isi dalam `LinearDB` (**keterikatan**). Ekstraklah superkelas abstrak. Buku: Subbab 7.10.*

Komposisi: *Has-a* Alih-alih *Is-a*

> **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.

Utamakan komposisi objek daripada pewarisan kelas. Subkelas bergantung bukan hanya pada apa yang *dijanjikan* superkelasnya, tetapi juga pada *cara ia diimplementasikan*.

Sebuah `Order` mempunyai layanan; sebuah mobil mempunyai mesin. Pewarisan menciptakan keterikatan paling erat yang mungkin di antara dua kelas. Buku: Subbab 7.11.

Contoh Terkerjakan 4: Kelas Dasar yang Rapuh

Kode 7.11: himpunan label yang menghitung setiap label yang pernah ditambahkan, dengan memperluas `HashSet<String>`. Tiga label dimasukkan.

Kode 7.11 Dari `FragileBase.java`: kedua *override* milik `InheritedTagSet`.

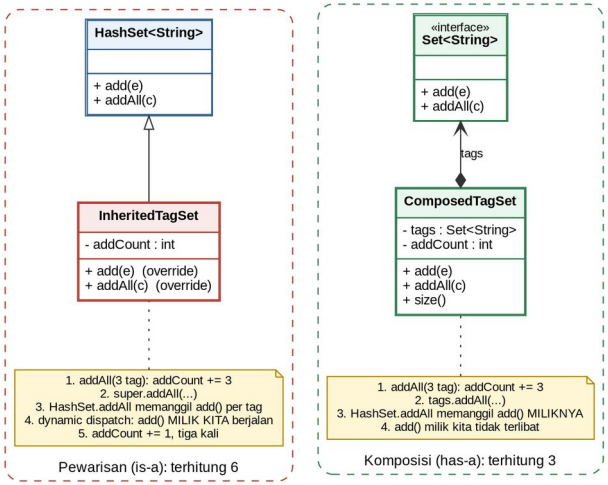
```
    @Override
    public boolean add(String tag) {
        addCount++;
        return super.add(tag);
    }

    @Override
    public boolean addAll(Collection<? extends String> tags) {
        addCount += tags.size();
        return super.addAll(tags);  // HashSet.addAll memanggil add() untuk setiap tag!
    }
```

```
inherited: size = 3, tags added = 6
composed:  size = 3, tags added = 3
```

`addAll` menambah 3, lalu `super.addAll` memanggil `add` tiga kali — dan pengiriman dinamis mengembalikan panggilan itu ke `add` milik kita: enam. Versi komposisi memegang sebuah `Set` dan menghitung 3. Buku: Subbab 7.11.1.

Mengapa Hitungannya Berlipat



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.

Gambar 7.8: apakah `addAll` memanggil `add` adalah rincian `HashSet` yang tidak didokumentasikan; subkelas yang benar dapat rusak ketika rincian itu berubah. `UserDict` di Python memasang jebakan yang sama. Buku: Subbab 7.11.1.

Kapan Pewarisan Tepat?

Prinsip substitusi Liskov, Persamaan (7.4): apa pun yang terbukti untuk setiap T harus berlaku untuk setiap subtipe S.

$$\forall x : T \,.\, \phi(x) \Rightarrow \forall y : S \,.\, \phi(y) \text{ bilamana } S \text{ adalah subtipe dari } T \tag{7.4}$$

1. Apakah kalimat *is-a* terdengar benar bagi orang awam? **2.** Apakah setiap pemanggil superkelas puas dengan objek subkelas? **3.** Inginkah Anda subkelas berubah ketika isi dalam superkelas berubah?

Binatu memakai keduanya: `ExpressService` *adalah* sebuah `Service` dan *mempunyai* sebuah `Service` — pola **dekorator**.

Bila pemanggil mengandalkan harga yang positif, tak satu subkelas pun boleh mengembalikan nol; bila mereka mengandalkan urutan penyisipan, `TreeDB` *tidak boleh menyamar sebagai basis data linear. Buku: Subbab 7.11.2.*

Java: Satu Superkelas, Banyak Antarmuka

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

Tabel 7.3: bahaya pewarisan jamak adalah mewarisi keadaan dan kode melalui dua jalur; antarmuka tidak membawa keadaan. Buku: Subbab 7.12.1.

Dua *Default* dengan Satu *Signature*

`Foldable` dan `Steamable` masing-masing mempunyai `finish()` *default*. `PremiumCare` mengimplementasikan keduanya dan **wajib** memilih (Kode 7.12):

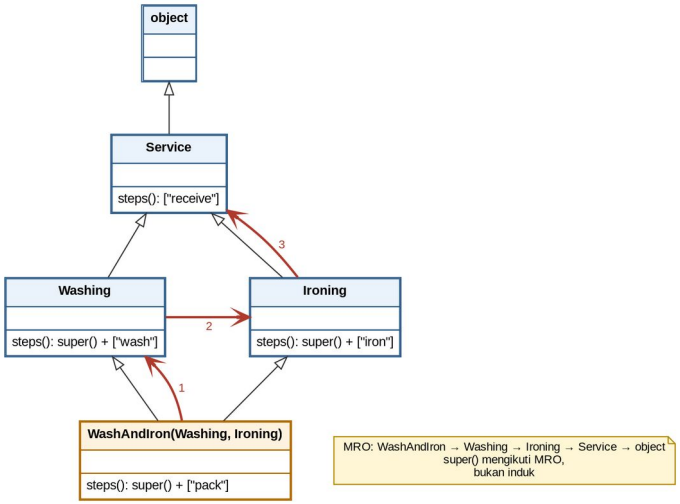
Kode 7.12 Dari `DiamondDemo.java`: menyelesaikan dua *default* yang bertentangan dengan `Interface.super.method()`.

```
// 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();
    }
}
```

```
PremiumCare: steamed, then folded
as Foldable:  steamed, then folded
as Steamable: steamed, then folded
HotelCare:   hung on a rack
```

Tanpa override itu, `javac` berhenti: "inherits unrelated defaults for finish()". Aturannya: metode kelas mengalahkan default; antarmuka yang lebih spesifik mengalahkan yang kurang spesifik; selain itu kelasnya wajib memilih. Buku: Subbab 7.12.1.

Berlian Python



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.

Gambar 7.9: `WashAndIron` mempunyai dua jalur ke `Service`; Python meluruskannya menjadi satu MRO, dan setiap `super()` berpindah ke kelas berikutnya di dalamnya, sehingga `Service.steps` berjalan sekali. Buku: Subbab 7.12.2.

`super()` Bukan "Induk Saya"

Kode 7.13: setiap `steps` memanggil `super().steps()` dan menambahkan satu kata.

Kode 7.13 Dari `diamond_mro.py`: berlian dengan panggilan `super()` yang kooperatif.

```
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"]
```

```
WashAndIron MRO: ['WashAndIron', 'Washing', 'Ironing', 'Service', 'object']
steps(): ['receive', 'iron', 'wash', 'pack']
```

*`super()` milik `Washing` pergi ke `Ironing` — bukan superkelasnya, melainkan kelas berikutnya dalam MRO `WashAndIron`. Itulah **pewarisan jamak kooperatif**. Buku: Subbab 7.12.2.*

Contoh Terkerjakan 5: C3 dengan Tangan

Linearisasi Persamaan (7.5): ambil *head* pertama yang tidak muncul di *tail* daftar lain mana pun.

$$L[C] = C + \text{merge}\big(L[B_1], \ldots, L[B_n], [B_1, \ldots, B_n]\big) \tag{7.5}$$

Gabungkan $[W, S, o]$, $[I, S, o]$, $[W, I]$: ambil W. *Head* S ada di sebuah *tail* — lewati; ambil I. Sekarang ambil S, lalu o.

$L[\text{WashAndIron}] = [\text{WashAndIron}, W, I, S, o]$, dengan W = `Washing`, I = `Ironing`, S = `Service`, o = `object`.

Leluhur bersama menunggu sampai semua keturunannya tercantum. `class Confused(Service, Washing)` *tidak mempunyai urutan yang konsisten, dan Python melempar* `TypeError`*. Buku: Subbab 7.12.2.*

Empat Tingkat Akses Java

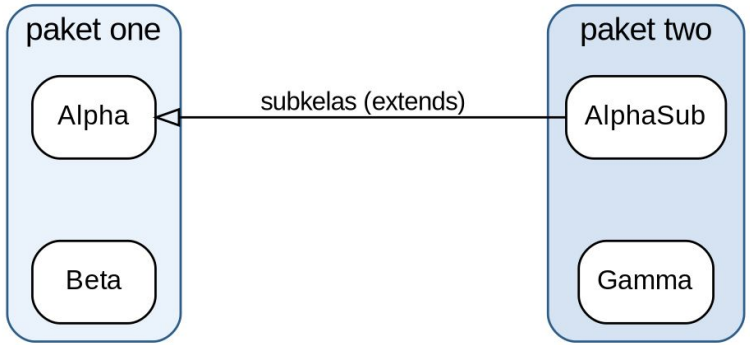
Tabel 7.4 Tingkat akses Java. *Kelas*: kode di kelas yang sama; *Paket*: kode di kelas mana pun dalam paket yang sama; *Subkelas*: kode di subkelas yang dideklarasikan di paket lain; *Dunia*: kode lain mana pun.

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

Kelas tingkat atas bersifat `public` atau *package-private*. Anggota mengambil satu dari empat tingkat: `public`, `protected`, tanpa pengubah (*package-private*), `private`.

Tabel 7.4 adalah tabel rujukan resmi: "Y" berarti kode di tempat itu dapat mengakses anggota yang dideklarasikan dengan pengubah tersebut. Buku: Subbab 7.13.1.

Empat Kelas, Dua Paket



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`.

Gambar 7.10: `Beta` sepaket dengan `Alpha`; `AlphaSub` berada di paket lain tetapi memperluas `Alpha`; `Gamma` tidak berhubungan dengannya. Buku: Subbab 7.13.1.

Contoh Terkerjakan 6: Biarkan Kompilator Membuktikan

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

`java -cp out two.Gamma` mencetak satu kolom Tabel 7.5 per kelas; menambahkan `a.protectedField`, `a.packageField`, `a.privateField` ke `Gamma` menghasilkan tiga galat yang berbeda.

Kehalusannya: `AlphaSub` boleh memakai `protectedField` milik objeknya **sendiri**, tetapi tidak `other.protectedField` milik sembarang `Alpha`.

Pakailah tingkat akses paling sempit yang masuk akal — `private` kecuali ada alasan kuat — dan tanpa field publik kecuali konstanta. `protected` adalah janji kepada setiap subkelas yang kelak ditulis siapa pun. Buku: Subbab 7.13.1 dan 7.13.2.

Python: Konvensi, Bukan Aturan

`name` publik; `_name` internal ("subkelas dipersilakan"); `__name` **diubah namanya** (*name mangling*) menjadi `_Alpha__name`, untuk menghindari bentrokan, bukan untuk kerahasiaan (Kode 7.14).

Kode 7.14 Dari `access_demo.py`: tiga konvensi penamaan Python.

```python
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}"
```

```
AlphaSub sees: public protected  (AttributeError: 'AlphaSub' object has no attribute
'_AlphaSub__private_field')
attribute names: ['public_field', '_protected_field', '_Alpha__private_field']
```

Di dalam `AlphaSub`, `self.__private_field` menjadi `_AlphaSub__private_field`, yang tidak ada. "Kita semua orang dewasa yang saling setuju": batasnya ditandai, bukan ditegakkan. Buku: Subbab 7.13.3.

Enkapsulasi Menjaga Invarian

Mesin cuci menjanjikan Persamaan (7.6): muatan ℓ di antara 0 dan kapasitas c. Hanya `addLoad` yang dapat menaikkan muatan (Kode 7.15).

$$0 \le \ell \le c \tag{7.6}$$

Kode 7.15 Metode `addLoad` di `EncapsulationDemo.java`: satu-satunya cara menaikkan muatan, dan ia menjaga invariannya.

```
    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;
    }
```

```
W-03: 5.5 / 8.0 kg
Refused: 3.0 kg does not fit, only 2.5 kg free
```

`m.loadKg = 99` tidak dapat dikompilasi: "loadKg has private access". Dan karena tak ada yang melihat `loadKg`, kelak ia dapat disimpan dalam gram tanpa mengubah kelas lain — information hiding ala Parnas. Buku: Subbab 7.14.

Property di Python

Property adalah metode yang dipanggil seperti field: Kode 7.16 menjalankan pemeriksaan *setter* bahkan untuk `m.load_kg = 99`.

Kode 7.16 Dari `encapsulation_demo.py`: `load_kg` sebagai *property* dengan *setter* yang memvalidasi.

```
    @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
```

```
Refused: 99.0 kg is outside 0..8.0 kg
AttributeError: property 'capacity_kg' of 'WashingMachine' object has no setter
```

`capacity_kg` tidak mempunyai setter, sehingga menugasinya melempar `AttributeError`. Pemrogram Python memulai dengan atribut biasa dan menambahkan property ketika muncul aturan; pemanggil tidak pernah menyadarinya. Buku: Subbab 7.14.1.

Profitina Laundry: Aturan Harganya

Wash & Fold Rp7,000/kg, min. 3 kg, 48 jam; **Iron Only** Rp5,000/kg, min. 2 kg, 24 jam; **Wash, Dry & Iron** Rp10,000/kg, min. 3 kg, 72 jam; **Dry Clean** per potong, 72 jam.

Berat terhitung Persamaan (7.7) dan harga menurut berat Persamaan (7.8); kilat Persamaan (7.9); total pesanan dan total anggota Persamaan (7.10).

$$\hat{w} = \frac{\lceil 2 \max(w, w_{\min}) \rceil}{2} \tag{7.7}$$

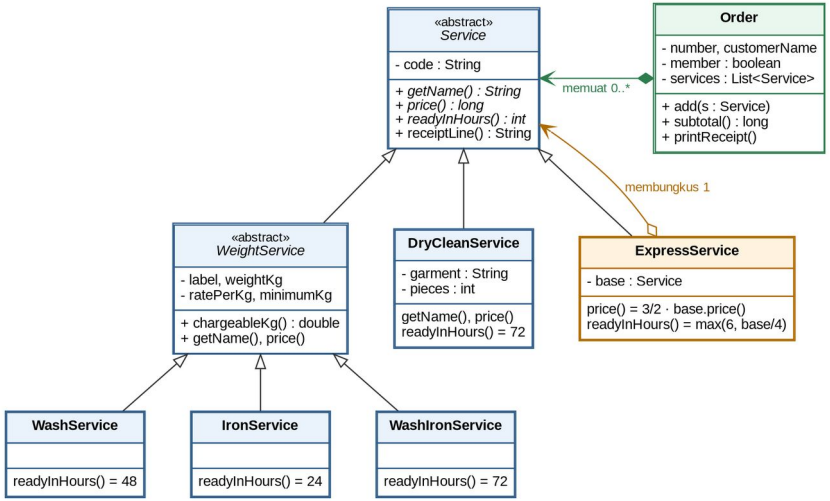
$$P_{\text{berat}} = \text{round}(r \cdot \hat{w}) \tag{7.8}$$

$$P_{\text{kilat}} = \lfloor \frac{3P}{2} \rfloor \tag{7.9}$$

$$T = \sum_{i=1}^{n} P(s_i), T_{\text{anggota}} = T - \lfloor \frac{T}{10} \rfloor \tag{7.10}$$

4,2 kg Wash & Fold: $\max(4.2, 3) = 4.2$*, dikali dua* 8.4*, dibulatkan ke atas* 9*, dibagi dua* 4.5 *kg,* $P = 31,500$*. Kilat selesai dalam seperempat waktu, tidak pernah kurang dari 6 jam. Buku: Subbab 7.15.*

Hierarki Layanan



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).

Gambar 7.11: `Service` dan `WeightService` abstrak; lima layanan konkret; `ExpressService` sekaligus adalah `Service` dan membungkus satu `Service`; sebuah `Order` memegang layanan sebanyak apa pun. Buku: Subbab 7.15.

`Service`: Tiga Janji dan Satu Templat

Kode 7.17: setiap layanan mempunyai kode; masing-masing wajib menyatakan nama, harga, dan waktu selesainya; `receiptLine` memformat semuanya.

Kode 7.17 Dari `LaundryPricing.java`: metode-metode abstrak `Service` dan metode templatnya.

```
    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());
    }
```

`receiptLine` tidak menyebut subkelas mana pun. `Order.subtotal()` adalah perulangan empat baris, `sum += s.price()`, yang juga tidak menyebut subkelas — pengiriman dinamis memilih `price()` yang tepat. Buku: Subbab 7.15.

Subkelas yang Nyaris Kosong

`WeightService` memegang aturan per kilogram sekali saja; Kode 7.18 memperlihatkan berat terhitung dan satu layanan konkret yang utuh.

Kode 7.18 Dari `LaundryPricing.java`: `chargeableKg` di `WeightService`, dan `WashService` selengkapnya.

```
    // Paling sedikit minimum, dibulatkan ke atas ke 0,5 kg berikutnya.
    double chargeableKg() {
        return Math.ceil(Math.max(weightKg, minimumKg) * 2) / 2;
    }
class WashService extends WeightService {
    WashService(double weightKg) {
        super("WF", "Wash & Fold", weightKg, 7000, 3.0);
    }

    @Override
    int readyInHours() {
        return 48;
    }
}
```

`switch` dari Bab 3 telah menjadi tiga override satu baris untuk `readyInHours()`. Hapus satu, dan `javac` menolak: "IronService is not abstract and does not override abstract method readyInHours()". Buku: Subbab 7.15.

Kilat: *Is-a* dan *Has-a* Sekaligus

Kode 7.19: `ExpressService` memperluas `Service` **dan** memegang sebuah `Service` bernama `base`.

Kode 7.19 Dari `LaundryPricing.java`: dekorator `ExpressService`.

```
    ExpressService(Service base) {
        super("EX-" + base.getCode());
        this.base = base;
    }
    @Override
    long price() {
        return base.price() * 3 / 2;  // biaya tambahan 50%
    }

    @Override
    int readyInHours() {
        return Math.max(6, base.readyInHours() / 4);
    }
```

Karena ia menerima `Service` apa pun, kilat berlaku untuk cuci kering, untuk layanan yang ditambahkan tahun depan, bahkan untuk kilat yang lain (Soal Latihan 7.9). Buku: Subbab 7.15.

Contoh Terkerjakan 7: Nota Siti

Satu pesanan, enam layanan, seorang anggota; satu perulangan menghargai semuanya:

```
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
```

Setrika 1,3 kg → minimum 2 kg, $10,000$*. WDI kilat:* $\lfloor 3 \cdot 30,000/2 \rfloor = 45,000$*, selesai dalam* $\max(6, 72/4) = 18$ *jam. Subtotal* $236,500$*; potongan* $23,650$*; total* $212,850$*. Python mencetak hal yang sama. Buku: Subbab 7.15.*

Profitina dalam Praktik

Pemindaian Profitina memeriksa banyak **sinyal** — ekstraksi yang bersih, data terstruktur schema.org, apakah `robots.txt` mengizinkan mesin penjawab masuk.

Satu abstraksi, *sebuah sinyal*, berjanji: ukur halaman, beri skor, jelaskan skornya. Banyak kelas sinyal konkret, masing-masing diuji sendirian. Pemindaiannya satu perulangan, seperti `Order.subtotal()`.

Perilaku bersama tinggal di objek pembantu yang dikomposisikan; data pribadi (UU PDP 27/2022) tetap berada di balik beberapa metode yang menegakkan aturannya, seperti `addLoad`.

Skor itu adalah saran: platform menjelaskan apa yang diukurnya; penggunalah yang memutuskan tindakannya. Buku: Subbab 7.16.

Kesalahan yang Sering Terjadi

Mewarisi padahal *is-a*-nya salah (`Order extends ArrayList<Service>`); ***overloading* padahal maksudnya *overriding*** — `equals(Customer)`; tulislah `@Override`.

Lupa `super().__init__()` di Python; **mendeklarasikan ulang field superkelas** di Java; mengharapkan anggota **statis** bersifat polimorfis.

***Downcasting* tanpa** `instanceof`; `equals` tanpa `hashCode`; **field** `protected` "supaya subkelas dapat memakainya".

Membaca `super()` sebagai "induk saya" pada berlian Python; **memanggil metode yang dapat di-*override* dari konstruktor**, sebelum field subkelas diisi.

Buku mendaftar sepuluh kesalahan, masing-masing dengan cara menghindarinya. Buku: Subbab 7.17.

Ringkasan: Lima Hal untuk Diingat

1. Pewarisan memindahkan kode bersama ke superkelas; ujilah dengan "C *adalah* B". Konstruktor berantai ke atas; Python memanggil `super().__init__()` hanya bila Anda memanggilnya.
2. *Overriding*: nama dan parameter sama, dipilih saat berjalan; *overloading*: parameter berbeda, dipilih saat kompilasi.
3. Tipe statis menentukan apa yang lolos kompilasi; tipe dinamis menentukan *override* mana yang berjalan (Persamaan (7.1)); Python menyusuri MRO.
4. Utamakan superkelas abstrak bersama, dan komposisi daripada pewarisan kecuali substitusi (Persamaan (7.4)) berlaku.
5. Pakailah akses tersempit yang berfungsi; enkapsulasi menjaga invarian; Python menandai batas dengan `_`, `__`, dan `@property`.

Di binatu, satu perulangan kini menghargai setiap jenis layanan, dan layanan baru adalah kelas baru, bukan label `case` baru. Buku: Subbab 7.18.

Pertemuan Berikutnya

Bab 8 — Evaluasi dan Pembahasan II. Soal-soal terkerjakan tentang Bab 5, 6, dan 7: pohon dan list rekursif, pengurai dengan eksepsi, larik yang tumbuh, dan soal-soal rancangan.

Setiap soal diselesaikan tuntas, di Java dan di Python, dan diperiksa oleh sebuah program.

Sebelum itu: kerjakan Soal Latihan 7.2, 7.4, 7.9, dan 7.10, di Java dan di Python.

Bawalah hierarki hari ini: pertemuan berikutnya kita merancang satu hierarki dari sebuah pernyataan soal. Buku: Subbab 7.19 dan 7.20.

Kuliah 8 — Evaluasi dan Pembahasan II: Struktur Rekursif, Kelas yang Tangguh, dan Hierarki Kelas

Pemrograman Berorientasi Objek (EF234302)

Buku: Irfan Subakti, *Pemrograman Berorientasi Objek dengan Java dan Python*, Bab 8

Dosen: Irfan Subakti — yifana@gmail.com

Tidak ada fitur bahasa baru hari ini: kita mengerjakan soal, dua puluh satu jumlahnya, masing-masing diselesaikan dari halaman kosong di Java dan di Python lalu diperiksa oleh sebuah program. Buku: Bab 8, Subbab 8.1–8.12.

Cakupan Pertemuan Ini

1. **Sebuah metode** — delapan langkah dari satu paragraf persyaratan menuju kelas, uji, dan kode; resep delegasi untuk rekursi.
2. **Dua struktur** — `Tree` yang bekerja melalui delegasi, dan *list* imutabel yang dibangun dari `nil` dan `cons`.
3. **Soal 8.1–8.7** — membangun, melipattigakan, menelusuri, menyalin, dan menyimpan pohon; pencarian dan penyisipan pada pohon pencarian biner.
4. **Soal 8.8–8.15** — *map*, pencarian, filter, keterurutan, penggabungan, dan duplikat pada *list* rekursif.
5. **Soal 8.16–8.21** — pengurai berkas yang tangguh, larik yang tumbuh sendiri, antarmuka dengan uji kontrak, hierarki yang ditelusuri dengan tangan, *refactoring*, dan perburuan *bug*.

Setiap soal menempuh jalur yang sama: pernyataannya, dengan bahasa kita sendiri, rancangan, Java, Python, baris demi baris, penyelesaian keliru yang menggoda, dan uji diri. Buku: Subbab 8.1.

Titik Tolak: Kotak Perkakas

Bab 5: larik dan *list*, masukan, eksepsi (`try`/`catch`/`finally`, terperiksa dan tak terperiksa, buatan sendiri), dan berkas yang ditutup dengan aman.

Bab 6: Javadoc dan *docstring*, JUnit 6 dan `unittest`, antarmuka, *overloading*.

Bab 7: pewarisan, kelas abstrak, *overriding*, polimorfisme, `super`, `final`, dan tingkat-tingkat akses.

> **CARA MEMAKAI PERTEMUAN INI.** Bacalah pernyataan soal, tutup sisanya, dan cobalah sendiri selama sepuluh menit: gambarlah rancangannya, tulislah kasus dasarnya, cobalah satu contoh dengan tangan. Lalu bandingkan.

Setiap program Java menguji dirinya dengan fungsi bantu kecil `check` yang mencetak `PASS` atau melempar `AssertionError`; program Python memakai `unittest`. Buku: Subbab 8.1.

Metode Delapan Langkah

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?

***Tabel 8.1**: Langkah 3, “siapa yang memiliki data yang dibutuhkan?”, mencegah sebagian besar rancangan buruk; Langkah 7, uji dengan kasus tepinya, menangkap sebagian besar penyelesaian keliru. Buku: Subbab 8.2.*

Tanggung Jawab dan Resep Delegasi

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.

RESEP DELEGASI. (1) *Kasus dasar*: jawaban untuk *list* kosong atau subpohon yang tidak ada. (2) *Delegasi*: apa yang saya minta dihitungkan oleh bagian-bagian saya yang lebih kecil? Anggaplah mereka mengerjakannya dengan benar. (3) *Tugas saya sendiri*: bagaimana menggabungkan jawaban mereka dengan data saya?

Bila ketiga jawaban itu benar, metodenya benar, menurut induksi atas ukuran strukturnya.

Utamakan hubungan has-a dan can-do; pakailah is-a hanya bila subkelas sungguh merupakan kasus khusus dan sungguh berbagi kode. Buku: Subbab 8.2.

Pohon yang Bekerja Melalui Delegasi

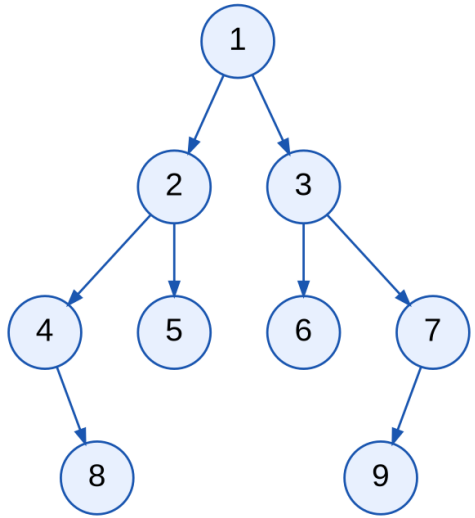
Kode 8.1: *field*-nya `private`; konstruktor daun berantai ke konstruktor cabang; `mirror()` meminta setiap subpohon mencerminkan dirinya, lalu menukar.

Kode 8.1 Dari `TreeStart.java`: konstruktor daun dan `mirror()` melalui delegasi.

```
        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;
        }
```

`toString()` menulis daun sebagai bilangannya dan cabang sebagai `root(left,right)`, `-` untuk subpohon yang tidak ada; setiap uji membandingkan string itu. Python memakai `_root` dan argumen default. Buku: Subbab 8.3.1.

Pohon pada Soal 8.1



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.

Gambar 8.1 *tercetak sebagai* `1(2(4(-,8),5),3(6,7(9,-)))`*: simpul 4 hanya mempunyai anak kanan dan simpul 7 hanya anak kiri. Buku: Subbab 8.3.1.*

List Imutabel dari `nil` dan `cons`

Sebuah *list* adalah kosong, `nil`, atau `cons` dari sebuah *head* dan *tail* yang lebih pendek, Persamaan (8.1):

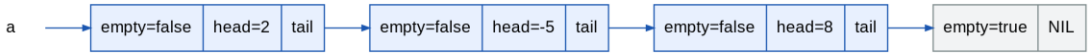
$$[2, -5, 8] = \text{cons}(2, \text{cons}(-5, \text{cons}(8, \text{nil}))) \tag{8.1}$$

Imutabel: *field* `private final`; setiap "perubahan" membangun *list* baru, sehingga berbagi *tail* aman. **Konstruktor privat, *factory* publik:** `nil()`, `cons()`, `of()`.

`ListException` untuk `head()` dari *list* kosong, tak terperiksa; `ListOps`, kelas utilitas, bukan superkelas: `NegateAll extends List` adalah *is-a* yang palsu.

Nama-nama itu berasal dari keluarga bahasa Lisp. Python menyimpan kelas dan fungsi-fungsi bantunya dalam satu modul, `cons_list.py`. Buku: Subbab 8.3.2.

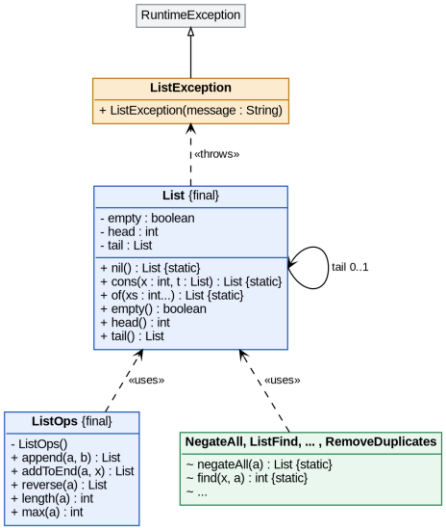
Objek-objek di Balik `[2, -5, 8]`



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.

***Gambar 8.2**: tiga sel, masing-masing sebuah head dan referensi ke tail-nya, serta satu list kosong bersama di ujungnya. Buku: Subbab 8.3.2.*

Kelas-kelas *List* dalam UML



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.

***Gambar 8.3**: `ListException` memperluas `RuntimeException`; kelas-kelas soal hanya memakai `List`, yang bersifat `final`. Buku: Subbab 8.3.2.*

Factory dan Kontrak yang Dilanggar

Kode 8.2: tak seorang pun di luar `List` dapat memanggil `new List(...)`; *list* kosong dibuat sekali dan dibagi bersama.

Kode 8.2 Dari `List.java`: dua *factory*, dan `head()` yang menolak *list* kosong.

```
    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);
    }
    public int head() {
        if (empty) {
            throw new ListException("head() of the empty list");
        }
        return head;
    }
```

`max` dari list kosong melempar `ListException`, dan sebuah `catch` di `main` mencetak pesannya; tanpa `try`, program akan berhenti dengan stack trace. Buku: Subbab 8.3.2.

Soal 8.1 — Membangun Pohon dengan Tangan

SOAL. Buatlah pohon pada gambar dengan sembilan pernyataan `new Tree`.

Cabang memerlukan subpohonnya sebagai argumen, jadi pohon dibangun **dari bawah ke atas** (**Kode 8.3**):

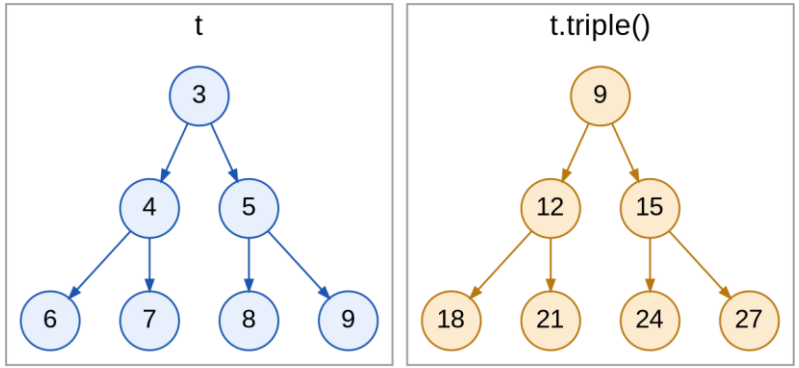
Kode 8.3 Metode `build()` di `BuildTree.java`: sembilan pemanggilan konstruktor, daun lebih dahulu.

```
    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
    }
```

```
t       = 1(2(4(-,8),5),3(6,7(9,-)))
wrong t = 1(2(4(8,-),5),3(6,7(9,-)))
```

Versi keliru menaruh setiap anak tunggal di kiri: `4(8,-)`. Simpulnya tetap sembilan, jadi hanya uji atas bentuk persisnya yang menangkapnya. Pada pohon, kiri dan kanan adalah bagian dari data. Buku: Subbab 8.4.1.

Soal 8.2 — Melipattigakan Setiap Simpul



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

***Gambar 8.4**: `void triple()` mengubah pohon di tempat; bentuknya tetap, setiap nilai dikalikan tiga. Buku: Subbab 8.4.2.*

Soal 8.2 — Dua Delegasi yang Dijaga

Kode 8.4: kasus dasar, penjaga `null`; delegasi, `left.triple()` dan `right.triple()`; tugas saya sendiri, `root = 3 * root`.

Kode 8.4 Metode `triple()` di `TripleTree.java`.

```
        void triple() {
            if (left != null) {
                left.triple();
            }
            if (right != null) {
                right.triple();
            }
            root = 3 * root;
        }
```

```
before: 3(4(6,7),5(8,9))
after : 9(12(18,21),15(24,27))
tripleWrong crashed: NullPointerException
half-done tree    : 9(12(18,7),5(8,9))
```

`tripleWrong()` membuang penjaganya, "setiap simpul mempunyai dua anak": daun tidak mempunyai satu pun. Macetnya meninggalkan pohon yang baru setengah dilipattigakan — periksalah sebelum mengubah. Buku: Subbab 8.4.2.

Soal 8.3 — Penelusuran *Depth-First*

SOAL. Tulislah `printDepthFirst()`, cabang kiri lebih dahulu: untuk pohon contoh ia mencetak `3 4 6 7 5 8 9`.

Tabel 8.2 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)

***Tabel 8.2**: setiap baris adalah satu pemanggilan; bacalah kolom tengah dari atas ke bawah dan Anda mendapat keluarannya. Inilah pre-order. Buku: Subbab 8.4.3.*

Soal 8.3 — Letak Pencetakannya

Kode 8.5: cetak akar saya, lalu minta kiri, lalu kanan.

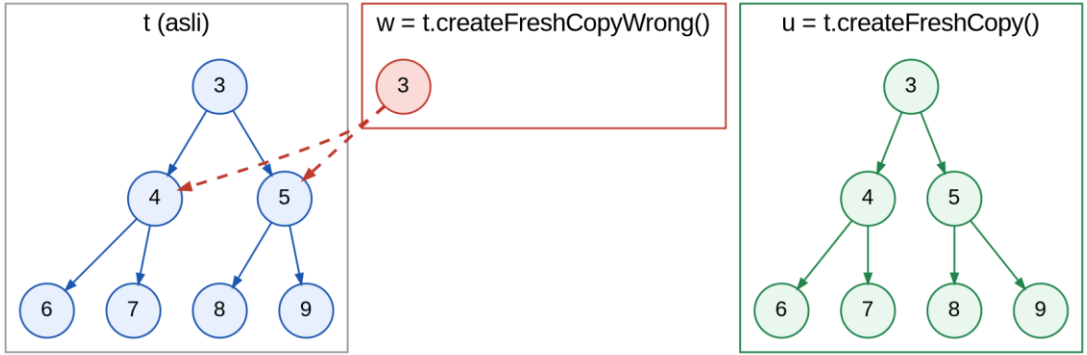
Kode 8.5 Metode `printDepthFirst()` di `DepthFirstPrint.java`.

```
        void printDepthFirst() {
            System.out.print(root + " ");
            if (left != null) {
                left.printDepthFirst();
            }
            if (right != null) {
                right.printDepthFirst();
            }
        }
```

```
3 4 6 7 5 8 9
6 4 7 3 8 5 9  <- wrong order
```

Memindahkan `print` ke antara kedua pemanggilan menghasilkan in-order, `6 4 7 3 8 5 9`. Ujinya menangkap `System.out` ke sebuah penyangga dan memulihkannya di `finally`; Python memakai `redirect_stdout`. Buku: Subbab 8.4.3.

Soal 8.4 — Salinan Dalam dan Salinan Dangkal



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`.

***Gambar 8.5**: sesudah `u = t.createFreshCopy(); t.mirror();`, `u` tidak boleh bergeser. Salinan dalam adalah tujuh objek baru; salinan keliru adalah satu akar baru yang menunjuk kembali ke dalam `t`. Buku: Subbab 8.4.4.*

Soal 8.4 — Simpan Hasil Pemanggilan

Kode 8.6: salinan segar sebuah simpul adalah simpul baru yang memegang salinan segar kedua subpohonnya.

Kode 8.6 Metode `createFreshCopy()` di `FreshCopy.java`.

```
        Tree createFreshCopy() {
            Tree leftCopy = (left == null) ? null : left.createFreshCopy();
            Tree rightCopy = (right == null) ? null : right.createFreshCopy();
            return new Tree(root, leftCopy, rightCopy);
        }
```

```
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))
```

*`createFreshCopyWrong()` melakukan pemanggilan rekursif tetapi **membuang hasilnya** lalu membangun `new Tree(root, left, right)`: salinan dangkal, setengah benar, yang hanya terbongkar oleh uji berbagi simpul. Buku: Subbab 8.4.4.*

Soal 8.5 — Menyimpan Penelusuran ke Larik

`saveDepthFirst(a, 4)` mengembalikan **posisi terakhir yang ditulis**, Persamaan (8.2):

$$end = s + n - 1, \tag{8.2}$$

Tabel 8.3 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

***Tabel 8.3**: subpohon kanan mulai sesudah sel terakhir subpohon kiri;* $4 + 7 - 1 = 10$*. Buku: Subbab 8.4.5.*

Soal 8.5 — Satu Konvensi, Dituliskan

Kode 8.7: `last` dimulai dari `whereToStart`; setiap subpohon mulai di `last + 1` dan jawabannya menggantikan `last`.

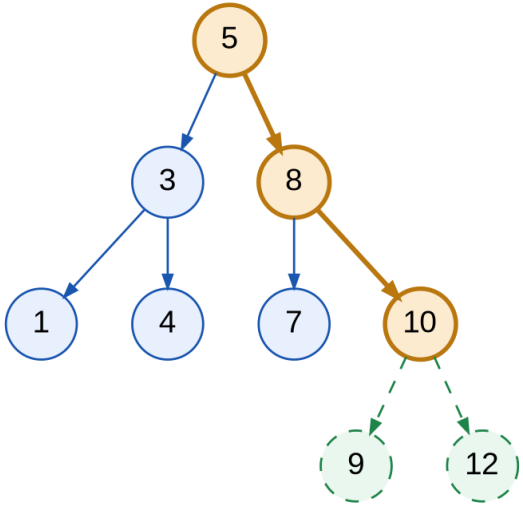
Kode 8.7 Metode `saveDepthFirst` di `SaveDepthFirst.java`.

```
        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;
        }
```

```
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
```

Versi keliru memulai subpohon kanan di `last`, bukan sesudahnya: tiga nilai tertimpa diam-diam. Mencampur "terakhir dipakai" dengan "bebas berikutnya" adalah galat selisih-satu yang klasik. Buku: Subbab 8.4.5.

Soal 8.6 — Pohon Pencarian Biner



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)`.

***Gambar 8.6**: nilai yang lebih kecil di kiri, yang lebih besar di kanan; `find(8)` menyusuri jalur tebal 5, 8 lalu berhenti. Buku: Subbab 8.4.6.*

Soal 8.6 — Satu Jalur ke Bawah

Kode 8.8: satu perbandingan menentukan paruh mana yang tidak mungkin memuat `n`; kunjungannya dibatasi oleh tinggi pohon, Persamaan (8.3).

$$\text{visits(find)} \le h, \qquad h \approx \log_2 n \text{ untuk pohon seimbang,} \tag{8.3}$$

Kode 8.8 Metode `find` di `BstFind.java`, dengan pencacah kunjungan.

```
        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);
            }
        }
```

```
find(8) = 8(7,10), nodes visited: 2
find(6) = null, nodes visited: 3
```

`findWrong` menghilangkan pemeriksaan `null` dan macet pada `find(6)`. Mencari di kedua subpohon "supaya aman" memang benar tetapi mengunjungi setiap simpul; hanya uji yang mencacah kunjungan yang menangkapnya. Buku: Subbab 8.4.6.

Soal 8.7 — Penyisipan Adalah `find` dengan Akhir Baru

Kode 8.9: turunlah seperti `find`; di tempat subpohon yang dibutuhkan tidak ada, buatlah daunnya **dan pasangkan**.

Kode 8.9 Dari metode `insert` di `BstInsert.java` (kasus sisi kanan simetris).

```
        Tree insert(int n) {
            if (n < root) {
                if (left == null) {
                    left = new Tree(n);          // pasang daun baru di sini
                } else {
                    left.insert(n);
                }
            }
            return this;                         // n == root: sudah ada
        }
```

```
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)))
```

Daftar in-order `1 3 4 5 7 8 9 10 12` membuktikan aturan pohon pencarian tetap berlaku. `insertWrong` membuat daun tetapi tak pernah menyimpannya — objek yang tidak dirujuk field mana pun adalah sampah. Buku: Subbab 8.4.7.

List Rekursif: Satu Kerangka

Setiap metode *list* di bawah ini mengisi kedua bagian kosong Persamaan (8.4):

$$f(a) = \begin{cases} \text{jawaban untuk list kosong} & \text{jika } a = \text{nil}, \\ \text{gabung}(\text{head}(a),\ f(\text{tail}(a))) & \text{selainnya.} \end{cases} \tag{8.4}$$

Hanya `empty()`, `head()`, `tail()`, `nil()`, dan `cons()` yang boleh dipakai: tanpa perulangan, tanpa *list* pustaka. *List* berisi n elemen menimbulkan $n + 1$ pemanggilan.

Setiap metode mengembalikan list baru; tidak ada list yang pernah diubah. Buku: Subbab 8.5.

Soal 8.8 — Menegasikan Sebuah *List*

Kode 8.10 adalah sebuah *map*: satu elemen keluaran per elemen masukan, `cons(-head, negateAll(tail))`.

Kode 8.10 Metode `negateAll` di `NegateAll.java`.

```
    static List negateAll(List a) {
        if (a.empty()) {
            return List.nil();                          // kasus dasar
        }
        return List.cons(-a.head(), negateAll(a.tail()));  // kasus rekursif
    }
```

```
a                = [2, -5, 8, 0]
negateAll(a)     = [-2, 5, -8, 0]
negateAllWrong(a) = [-2, -5, 8, 0]
```

Versi keliru lupa berekursi dan memasang tail aslinya. Menegasikan dua kali harus menghasilkan list semula — sebuah uji sifat. Di Java, $-(-2^{31})$ meluap kembali menjadi -2^{31}. Buku: Subbab 8.5.1.

Soal 8.9 — Sinyal yang Diteruskan

Kode 8.11: jawaban rekursifnya adalah posisi, digeser satu, atau sinyal -1, yang diteruskan **tanpa diubah**.

Kode 8.11 Metode `find` di `ListFind.java`.

```
    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;
    }
```

`findWrong` menambahkan 1 pada setiap jawaban: untuk nilai yang tidak ada di empat elemen, $-1 + 4 = 3$ — posisi yang sah, jenis jawaban keliru yang paling buruk. Buku: Subbab 8.5.2.

Soal 8.10 dan 8.11 — AND dan Filter

Kode 8.12: *list* kosong menjawab `true` (*kebenaran hampa*); **Kode 8.13**: filter *tail*-nya, lalu pertahankan atau buang *head*-nya.

Kode 8.12 Metode `allPositive` di `AllPositive.java`.

```
    static boolean allPositive(List a) {
        if (a.empty()) {
            return true;                    // tak ada elemen yang dapat melanggar aturan
        }
        return a.head() >= 0 && allPositive(a.tail());
    }
```

Kode 8.13 Metode `positives` di `Positives.java`.

```
    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;
    }
```

Kasus dasar kedua "list satu elemen pasti baik" melewatkan pemeriksaan: `[4, -1]` menjadi `true`. Berhenti pada elemen negatif pertama (`break` untuk `continue`) menghilangkan 8 sesudah −5. Buku: Subbab 8.5.3 dan 8.5.4.

Soal 8.12 — Apakah *List* Terurut?

Kode 8.14: langkah penggabungannya melihat **dua** elemen, jadi di sini dua kasus dasar memang tepat.

Kode 8.14 Metode `sorted` di `SortedCheck.java`.

```
    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());
    }
```

```
sorted([2, 3, -5, 8, -2]) = false
sorted([2, 3, 8])         = true
sorted([2, 5, 5, 8])      = true
sortedWrong([2, 5, 5, 8]) = false
```

"Naik dengan duplikat dibolehkan" berarti tidak turun: `<=`. `sortedWrong` memakai `<` dan menolak `[2, 5, 5, 8]` — bug satu karakter yang paling umum. Buku: Subbab 8.5.5.

Soal 8.13 — Menggabungkan Dua *List* Terurut

Kode 8.15: dua *list*, dua kasus dasar; setiap pemanggilan memindahkan *head* yang lebih kecil, sehingga kerjanya linear, Persamaan (8.5).

Kode 8.15 Metode `merge` di `MergeLists.java`.

```
    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()));
    }
```

$$\text{calls}(\text{merge}(a, b)) \leq |a| + |b|, \tag{8.5}$$

`merge([2,5,5,8], [9])` mengembalikan `b` utuh dan dibagi bersama begitu `a` habis. `mergeWrong` mengembalikan `nil()` di situ dan menghilangkan 9. `<=` mendahulukan elemen `a` yang sama: stabilitas. Buku: Subbab 8.5.6.

Soal 8.14 dan 8.15 — Lewati, Lalu Hapus Duplikat

Kode 8.16 adalah filter dengan uji "tidak sama dengan `x`"; **Kode 8.17** mempertahankan *head* dan **memakai ulang** `skip` untuk membuang salinan-salinannya yang kemudian.

Kode 8.16 Metode `skip` di `SkipElement.java`.

```
    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);
    }
```

Kode 8.17 Metode `removeDuplicates` di `RemoveDuplicates.java`.

```
    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));
    }
```

`[3, 1, 3, 2, 1]` menjadi `[3, 1, 2]`: masukannya tidak harus terurut. Versi yang hanya membandingkan tetangga benar untuk masukan terurut saja. Buku: Subbab 8.5.7 dan 8.5.8.

Harga Kesederhanaan

Untuk n elemen berbeda, pemanggilan `skip` menelusuri $n-1$, lalu $n-2$, … elemen, Persamaan (8.6):

$$(n-1)+(n-2)+\cdots+1+0=\frac{n(n-1)}{2}, \tag{8.6}$$

Untuk sejuta catatan pelanggan itu sekitar 5×10^{11} langkah; *hash set* (Bab 9) adalah alat yang tepat di sana.

Melempar eksepsi untuk *list* kosong juga keliru: *list* kosong tidak mempunyai duplikat, dan jawabannya adalah *list* kosong. Eksepsi untuk keadaan yang **tidak** mempunyai jawaban masuk akal.

Uji diri: `removeDuplicates([4, 4, 1, 4, 1, 7])` *adalah* `[4, 1, 7]`*, dan pemanggilan* `skip` *menelusuri 7 elemen, jauh di bawah 15. Buku: Subbab 8.5.8.*

Soal 8.16 — Pengurai Pesanan yang Tahan Eksepsi

> **SOAL.** Bacalah baris-baris `id;customer;service;kg`, pertahankan setiap pesanan yang sah, dan laporkan setiap baris buruk beserta nomor dan alasannya, *tanpa berhenti pada baris buruk pertama*. Berkas yang tidak ada menghasilkan pesan yang jelas.

```
# 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
```

Baris yang buruk dapat diduga: catatlah lalu lanjutkan. Berkas yang tidak ada menggagalkan seluruh pekerjaan: pemanggillah yang memutuskan. Buku: Subbab 8.6.1.

Soal 8.16 — Siapa Melapor, Siapa Menangani

Tabel 8.4 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`

***Tabel 8.4**: pilihan yang menentukan adalah letak setiap `try` — yang menangkap `OrderFormatException` berada di dalam perulangan, mengelilingi satu baris. Buku: Subbab 8.6.1.*

Soal 8.16 — Terjemahkan Galat Tingkat Rendah

Kode 8.18: `OrderFormatException` bersifat **terperiksa**, jadi setiap pemanggil wajib memikirkannya.

Kode 8.18 Akhir `parseLine` di `LaundryOrderParser.java`: `NumberFormatException` menjadi pesan yang dapat ditindaklanjuti orang.

```
        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);
```

`split(";", -1)` mempertahankan field kosong di ujung. Python menerjemahkan `ValueError` dengan `raise … from None`, karena pesan kita sudah mengatakan segalanya. Buku: Subbab 8.6.1.

Soal 8.16 — Satu Baris Buruk, Satu Baris Hilang

Kode 8.19: berkas dalam `try`-with-resources; `try` bagian dalam mencatat pesan lalu terus melaju.

Kode 8.19 Perulangan `readOrders` di `LaundryOrderParser.java`.

```
        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
                }
            }
        }
```

`Files.newBufferedReader` melempar `NoSuchFileException` untuk berkas yang tidak ada, dan `main` mengubahnya menjadi `Cannot find the file ...`. Buku: Subbab 8.6.1.

Soal 8.16 — Laporannya

```
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
```

Pembaca keliru memasang satu `try` di sekeliling seluruh perulangan: baris buruk pertama mengakhiri pembacaan dan `A107` hilang. Dan jangan pernah menulis `catch (Exception e) {}`.

Tiga pesanan sah, 10,5 kg, lima galat — satu dari setiap jenis dalam berkas; komentar dan baris kosong dilewati tanpa suara. Buku: Subbab 8.6.1.

Soal 8.17 — Larik yang Tumbuh Sendiri

`GrowableIntArray` menggandakan kapasitasnya bila penuh; `get` harus menolak indeks di luar `0` sampai `size() - 1`. Invarian kelasnya Persamaan (8.7):

$$0 \le size \le capacity = \texttt{data.length}, \quad \text{dan hanya } \texttt{data}[0..size-1] \text{ yang menyimpan nilai.} \tag{8.7}$$

Tabel 8.5 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

***Tabel 8.5**: prasyarat `get` diperiksa terhadap `size`, bukan terhadap panjang larik. Setiap metode publik ber-Javadoc; `javadoc -quiet` tidak mencetak apa pun. Buku: Subbab 8.6.2.*

Soal 8.17 — Uji pada `index == size`

Kode 8.20: `add` menyalin ke larik dua kali lebih panjang bila penuh; `get` memeriksa `index >= size`.

Kode 8.20 `add` dan `get` di `GrowableIntArray.java`.

```
    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++;
    }
    public int get(int index) {
        if (index < 0 || index >= size) {
            throw new IndexOutOfBoundsException("index " + index + ", size " + size);
        }
        return data[index];
    }
```

```
add(7) -> [3, 4, 5, 6, 7]  size 5, capacity 8
getWrong(5) = 0   <- a value that was never added
```

`getWrong` menguji `index > size`: `get(5)` membaca sel yang ada (kapasitas 8) tetapi tak pernah diisi — jawaban keliru yang diam. Penggandaan menjaga total penyalinan di bawah 2n. Buku: Subbab 8.6.2.

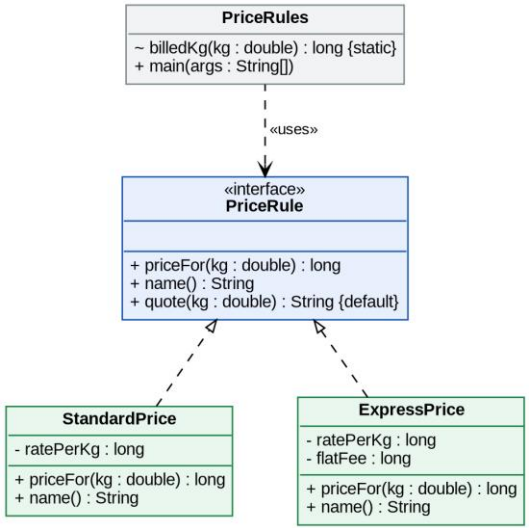
Soal 8.18 — Satu Antarmuka, Dua Aturan

> **SOAL.** Standar: Rp7,000 per kg. Kilat: Rp12,000 per kg ditambah Rp5,000. Setiap kilogram yang sudah dimulai ditagih; nol atau kurang adalah galat. Satu antarmuka `PriceRule`, sebuah metode `quote(kg)`, uji untuk keduanya.

Dengan tangan lebih dahulu: standar 2,3 kg, $\lceil 2.3 \rceil = 3$, 21 000; kilat, $3 \times 12\,000 + 5\,000 = 41\,000$; standar 0,1 kg, 7 000.

Sisa program memegang “suatu aturan harga” tanpa tahu yang mana; pembulatan dan validasinya ditulis sekali. Buku: Subbab 8.7.1.

Soal 8.18 — Rancangannya



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.

***Gambar 8.7**: dua metode abstrak, `priceFor` dan `name`, dan satu metode default, `quote`, yang ditulis di dalam antarmuka dengan memakai kedua metode lainnya. Buku: Subbab 8.7.1.*

Soal 8.18 — Metode *Default*

Kode 8.21: `quote` ditulis sekali untuk setiap aturan kini dan kelak.

Kode 8.21 Antarmuka `PriceRule` di `PriceRules.java`.

```
    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);
        }
    }
```

```
standard 2.3 kg = Rp21000
express 2.3 kg = Rp41000
wrong standard 2.3 kg = Rp16099
```

`(long) (7000 * 2.3)` menghasilkan 16099: dua bug, tanpa pembulatan ke atas dan 16099.999 … yang terpotong. Bulatkan ke atas dulu, lalu kalikan dengan tarif bilangan bulat. Python menjalankan satu kelas uji kontrak untuk kedua aturan; `@Test` dan `assertEquals` adalah bentuk JUnit 6 dari `check`. Buku: Subbab 8.7.1 dan 8.7.2.

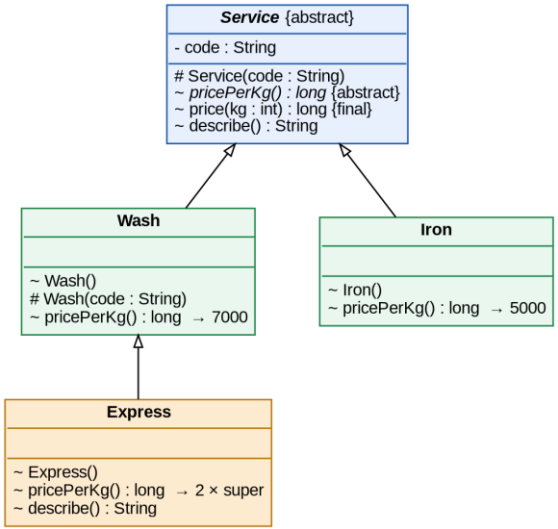
Soal 8.19 — Tebak Sebelum Menjalankan

> **SOAL.** `Wash` (Rp7,000/kg), `Iron` (Rp5,000/kg), dan `Express`, dua kali tarif cuci, “(same day)”. Sebuah `price(kg)` yang `final`, sebuah `describe()`, dan dua *overload* `label(Service)`, `label(Wash)`. Tebaklah setiap baris.

Overloading diselesaikan oleh kompilator, dari tipe yang **dideklarasikan**; *overriding* oleh program yang berjalan, dari **kelas objeknya**.

Mencampuradukkan keduanya adalah salah paham polimorfisme yang paling umum. Buku: Subbab 8.8.1.

Soal 8.19 — Hierarkinya



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`.

***Gambar 8.8**: `price` bersifat `final` di `Service` dan memanggil `pricePerKg` yang abstrak — pola template method. `Express` memperluas `Wash` dan mencapai versi-versi `Wash` dengan `super`. Buku: Subbab 8.8.1.*

Soal 8.19 — Kode yang Ditelusuri

Kode 8.22: `describe` di `Service` memanggil `pricePerKg()`, yang dikirim menurut objeknya — bahkan dari kode superkelas.

Kode 8.22 Dari `ServiceDispatch.java`: templat di `Service`, dan `Express` yang menggandakan tarif `Wash`.

```
        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";
        }
        @Override
        long pricePerKg() {
            return 2 * super.pricePerKg();      // dua kali tarif Wash
        }
```

Lupa `super.` pada `return 2 * super.pricePerKg();` membuat metode itu memanggil dirinya selamanya: `StackOverflowError`. Buku: Subbab 8.8.1.

Soal 8.19 — Penelusuran dengan Tangan

Tabel 8.6 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)`

***Tabel 8.6**: baris D adalah pelajarannya — `s` memegang `Express`, yang merupakan `Wash`, namun `label(Service)` yang dipilih, karena kompilator hanya melihat `Service s`. Buku: Subbab 8.8.1.*

Soal 8.19 — Jalankan dan Bandingkan

Kode 8.23 `main` di `ServiceDispatch.java`: baris A sampai F.

```
        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
```

```
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)
```

Kode 8.23 *cocok dengan penelusuran baris demi baris; cast mengubah tipe yang dideklarasikan dan karena itu mengubah overload-nya (baris F).* `singledispatch` *di Python memilih menurut kelas saat berjalan dan mencetak* `washing:` *untuk baris D. Buku: Subbab 8.8.1.*

Soal 8.20 — Dua Salinan Menjadi Satu Hierarki

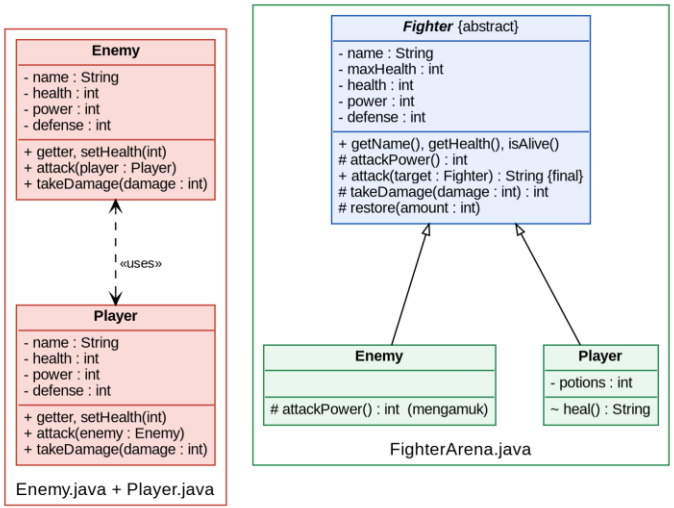
`Player` dan `Enemy` menduplikasi setiap *field* dan `takeDamage`. Dengan tangan, tiga pukulan: penjahat 80 → 60, pahlawan 100 → 90, penjahat 60 → 40 — dan program aslinya **tidak mencetak apa pun**.

Tabel 8.7 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

***Tabel 8.7**: tujuh masalah, masing-masing dengan perbaikannya; refactoring-nya adalah ekstrak superkelas. Buku: Subbab 8.8.2.*

Soal 8.20 — Sebelum dan Sesudah



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.

***Gambar 8.9**: satu `Fighter` abstrak; metode templat `final` bernama `attack`; hook (kait) `protected` `attackPower()` tempat subkelas boleh mengubah aturannya. Buku: Subbab 8.8.2.*

Soal 8.20 — Templat `final` dan Sebuah Kait

Kode 8.24: setiap serangan mengikuti langkah yang sama; musuh di bawah setengah nyawa menjadi murka.

Kode 8.24 Dari `FighterArena.java`: metode `final` `attack` dan kait yang di-*override* oleh `Enemy`.

```
    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)");
    }
    @Override
    protected int attackPower() {
        boolean enraged = getHealth() < getMaxHealth() / 2;
        return enraged ? getPower() + 10 : getPower();
    }
```

```
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)
```

Pada nyawa 20 penjahat itu murka: kekuatan 30, dan pahlawan kehilangan $30 - 10 = 20$*. Tidak ada setter untuk nyawa;* `Enemy extends Player` *adalah is-a yang palsu. Buku: Subbab 8.8.2.*

Soal 8.21 — Empat Jalan Masuk

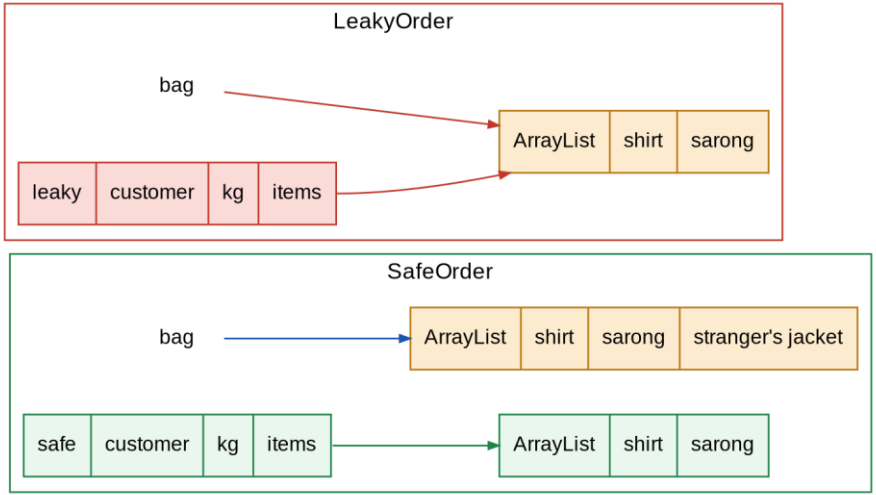
`LeakyOrder` lolos kompilasi dan berjalan, namun kode di luar dapat merusaknya dengan empat cara. Tanyakan pada setiap *field*: "siapa yang dapat mengubah ini?"

Tabel 8.8 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*

***Tabel 8.8**: bug 2 dan 4 sama-sama aliasing — dua referensi ke satu list yang dapat berubah, salah satunya dipegang orang asing. Buku: Subbab 8.8.3.*

Aliasing dalam Gambar



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.

***Gambar 8.10**: `SafeOrder` menyalin list itu, sehingga perubahan pemanggil kemudian tidak sampai kepadanya. Buku: Subbab 8.8.3.*

Soal 8.21 — Salin Masuk, Hanya-Baca Keluar

Kode 8.25: validasi di satu tempat, salin *list* ke dalam, serahkan tampilan hanya-baca.

Kode 8.25 Dari `OrderBugHunt.java`: konstruktor dan `getItems` milik `SafeOrder`.

```
        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
        }
        java.util.List<String> getItems() {
            return Collections.unmodifiableList(items); // tampilan baca-saja KELUAR
        }
```

```
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]
```

`final` membekukan referensi, bukan list-nya: `LeakyOrder.items` sudah `final` dan tetap dikosongkan dari luar. Python memakai property, setter yang memvalidasi, dan tuple. Buku: Subbab 8.8.3.

Perbandingan Soal: Pohon dan *List*

Tabel 8.9 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

***Tabel 8.9**, Soal 8.1–8.11: bacalah kolom terakhir ke bawah — setiap penyelesaian keliru gagal pada satu jenis masukan, jarang contoh di pernyataan soalnya. Buku: Subbab 8.9.*

Perbandingan Soal: Sisanya

Tabel 8.10 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
8.12	*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`

***Tabel 8.10**, Soal 8.12–8.21: argumen terkuat untuk Langkah 7 metode itu — tulislah uji kasus tepi lebih dahulu. Buku: Subbab 8.9.*

Profitina dalam Praktik

Satu antarmuka, banyak implementasi: pemindai memperlakukan setiap sinyal melalui satu antarmuka, seperti perulangan Soal 8.18 memperlakukan setiap `PriceRule`; sinyal baru adalah kelas baru dengan uji kontrak yang sama.

Masukan tak tepercaya tanpa macet: halaman tulisan orang asing dan tautan yang diunggah adalah Soal 8.16 dalam skala besar — satu butir buruk, satu butir hilang.

Enkapsulasi sebagai kewajiban hukum: persetujuan, ekspor, dan penghapusan menurut UU PDP 27/2022 menuntut data pribadi diubah di satu tempat terkendali, seperti `SafeOrder`; rilis menunggu lebih dari seribu uji.

Produknya memberi saran dan penggunanya yang memutuskan: skor adalah pengukuran beserta penjelasannya, bukan jaminan. Buku: Subbab 8.10.

Kesalahan yang Sering Terjadi

Rekursi: melupakan subpohon yang tidak ada; mengabaikan hasil pemanggilan; kasus dasar kedua yang melewatkan pemeriksaan; memperlakukan sinyal seperti -1 sebagai data.

Batas: `<` lawan `<=`, "terakhir dipakai" lawan "bebas berikutnya"; tuliskan konvensinya dan ujilah tepat di batasnya.

Eksepsi dan uang: satu `try` untuk segalanya; `catch (Exception e) {}`; uang dalam *floating-point*.

Kelas: mencampuradukkan *overloading* dan *overriding*; lupa `super`; pewarisan demi kemudahan; mengira `final` membuat objek imutabel; hanya menguji contoh di pernyataan soal.

Buku mendaftar dua belas kesalahan, masing-masing dengan soal yang menggambarkannya. Buku: Subbab 8.11.

Ringkasan: Lima Hal untuk Diingat

1. Selesaikan soal banyak kelas dalam delapan langkah: kata benda dan kata kerja, kelas, pemilik, hubungan, akses, gambar, uji dengan kasus tepi, satu metode setiap kali.

2. Tulislah rekursi dengan resep delegasi; untuk *list*, Persamaan (8.4) adalah templat setiap penyelesaian.

3. Tangkap catatan buruk di dalam perulangan, selalu tutup berkas, dan terjemahkan galat tingkat rendah menjadi pesan yang dapat ditindaklanjuti.

4. Jagalah invarian kelas dengan *field* privat, validasi, salinan defensif, dan tampilan hanya-baca.

5. *Overload* dipilih saat kompilasi, *override* saat berjalan; ekstrak superkelas bersama dengan templat `final` dan kait `protected`.

Setiap penyelesaian keliru yang menggoda hari ini lolos pada contoh biasa dan gagal pada satu kasus tepi. Buku: Subbab 8.12.

Pertemuan Berikutnya

Bab 9 — Koleksi dan Generik. *List*, himpunan, dan peta dari pustaka, serta parameter tipe yang membuatnya aman.

Hash set yang dinantikan Soal 8.15, dan `Set` yang membuat pemeriksaan duplikat cepat.

Sebelum itu: kerjakan Soal Latihan 8.1, 8.5, 8.7, dan 8.9, di Java dan di Python.

Penyelesaian soal latihan, beserta programnya, ada di buku. Buku: Subbab 8.13 dan 8.14.

Bab 1

Kuliah 9 — Koleksi dan Generik

Pemrograman Berorientasi Objek (EF234302)

Buku: Irfan Subakti, *Pemrograman Berorientasi Objek dengan Java dan Python*

Dosen: Irfan Subakti — yifana@gmail.com

Buku: Bab 9, Koleksi dan Generik. Setiap halaman handout ini diakhiri dengan penunjuk ke buku, yang menjadi rujukan utama; handout ini adalah bab tersebut dalam bentuk ringkas.

Pertemuan Hari Ini

1. Dua perkakas kecil: konstanta dalam antarmuka, dan rekursi dengan metode pembantu (Subbab 9.2).

2. Collections Framework dan koleksi Python: list, set, map, antrean, tumpukan, dan antrean prioritas — bagaimana masing-masing menyimpan elemennya dan berapa biaya setiap operasinya (Subbab 9.3–9.7).

3. Generik dari nol: parameter tipe, metode generik, batas, wildcard, dan *type erasure* — di Java, dan di Python dengan `typing` dan mypy (Subbab 9.8).

4. Urutan: `Comparable`, `Comparator`, dan fungsi `key=` Python (Subbab 9.9).

5. Memilih koleksi dengan menghitung kerjanya, dan buku pesanan Profitina Laundry dengan empat indeks atas objek-objek yang sama (Subbab 9.10–9.13).

Buku: Bab 9, Subbab 9.1 sampai 9.14.

Di Mana Kita Berhenti

Bab 5 memberi kita larik dan `ArrayList<String>` — sambil berjanji bahwa `<String>` akan dijelaskan di Bab 9. Inilah bab itu.

Bab 6: *antarmuka* adalah janji tanpa badan; `List` adalah antarmuka, `ArrayList` mengimplementasikannya; `Comparable` menjanjikan urutan alami.

Bab 7: hierarki, sehingga satu variabel bertipe `Service` memegang layanan cuci, setrika, atau kilat. **Bab 8:** struktur rekursif, list bilangan bulat imutabel dari `nil` dan `cons`.

> **PERTANYAAN HARI INI.** Hari Sabtu yang sibuk, beberapa ratus pesanan, empat pertanyaan dalam satu tarikan napas: *Di mana pesanan L-217? Apa saja yang dititipkan Bu Sari? Pesanan mana yang siap? Kantong mana yang masuk ke mesin berikutnya yang kosong?* Perulangan atas satu list besar menjawab keempatnya — dengan lambat. Sebuah **map**, sebuah **set** per status, dan sebuah **antrean prioritas** menjawab masing-masing seketika: pertanyaanlah yang memilih koleksi.

Buku: Subbab 9.1.

Konstanta dalam Antarmuka

Antarmuka boleh memegang konstanta selain kepala metode: `int LOSE = 0; int DRAW = 1; int WIN = 2;` di dalam `Measurable`.

Setiap field antarmuka **otomatis** `public static final` — public (janji publik), static (antarmuka tidak punya objek), final (tanpa keadaan yang berubah).

Kelas yang mengimplementasikan antarmuka mewarisi nama-namanya: di dalam `Team`, `r == WIN` tidak perlu awalan. **Kode 9.1** memperlihatkan kedua antarmuka.

Kode 9.1 `InterfaceConstants.java` (cuplikan): antarmuka `Measurable` dan `Units` beserta konstantanya.

```
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
}
```

Dua kemenangan dan satu seri: $3 + 1 + 3 + 0 = 7$ poin; sebelas pon adalah $11 \times 0.45359237 = 4.98951607$ kg, dicetak sebagai `4.990`.

Buku: Subbab 9.2.1 dan Kode 9.1.

Enum: Rumah yang Lebih Aman

Konstanta antarmuka hanyalah `int` biasa: tidak ada yang mencegah pemanggil mengirim `7`. **Enum** mendeklarasikan tipe yang nilainya persis yang didaftarkan, masing-masing objek dengan data dan metode.

`Measurable.WIN = 5;` tidak lolos kompilasi di Java (`cannot assign a value to static final variable WIN`). Di Python pernyataan itu berjalan — dan diam-diam menurunkan skor tim dari 7 menjadi 1, seperti diperlihatkan **Kode 9.2**.

Kode 9.2 `interface_constants.py` (cuplikan): atribut kelas bertanda `Final` dapat diikat ulang saat eksekusi; anggota `Enum` tidak.

```
    # 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)
```

```
after Measurable.WIN = 5: 1.0 points
AttributeError: cannot reassign member 'WIN'
```

> **mypy HANYA MEMERIKSA FUNGSI YANG BERANOTASI.** `Final` adalah janji yang ditegakkan pemeriksa tipe, bukan oleh Python. mypy melaporkan kedua penugasan — tetapi hanya karena `main` dideklarasikan `def main() -> None:`.

Buku: Subbab 9.2.1 dan Kode 9.2.

Rekursi: Dua Bagian

Metode rekursif memanggil dirinya sendiri. Bayangkan *bagi dan taklukkan*: masalah sejenis yang lebih kecil, ditambah sedikit kerja.

Kasus dasar: versi paling sederhana, dijawab langsung; dialah yang menghentikan pengulangan.

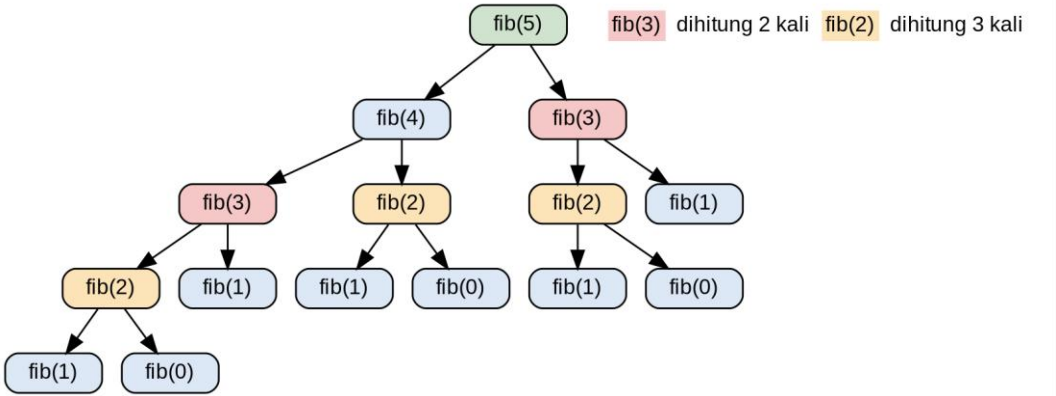
Panggilan rekursif: metode memanggil dirinya dengan argumen yang *lebih kecil*, bergerak menuju kasus dasar. Lupakan salah satunya, dan Java melempar `StackOverflowError`, Python `RecursionError`.

Contoh klasiknya, Persamaan (9.1), dapat diketik ke dalam Java hampir kata demi kata:

$$F(0) = 0, \qquad F(1) = 1, \qquad F(n) = F(n-1) + F(n-2) \text{ untuk } n \geq 2 \tag{9.1}$$

Buku: Subbab 9.2.2 dan Persamaan (9.1).

Mengapa Fibonacci Standar Lambat



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.

Gambar 9.1: `fib(4)` *juga* meminta `fib(3)`, dan tidak ada yang mengingat jawaban pertama.

Buku: Subbab 9.2.2 dan Gambar 9.1.

Menghitung Panggilannya

Misalkan $C(n)$ banyaknya panggilan untuk menghitung $F(n)$: satu untuk n sendiri, ditambah panggilan dari kedua panggilan rekursifnya.

$$C(0) = C(1) = 1, \qquad C(n) = 1 + C(n-1) + C(n-2) \quad \Rightarrow \quad C(n) = 2F(n+1) - 1 \tag{9.2}$$

Periksa dengan induksi: $C(n) = 1 + (2F(n) - 1) + (2F(n-1) - 1) = 2F(n+1) - 1$. Untuk $n = 5$: $2 \times 8 - 1 = 15$, kelima belas kotak itu.

Untuk $n = 30$, Persamaan (9.2) memberi $2 \times 1{,}346{,}269 - 1 = 2{,}692{,}537$ panggilan — demi bilangan yang dapat dihitung seorang anak dengan tiga puluh penjumlahan.

$F(n)$ tumbuh seperti $\varphi^n/\sqrt{5}$ dengan $\varphi \approx 1.618$: setiap langkah tambahan melipatkan kerjanya sekitar 1,6 kali.

Buku: Subbab 9.2.2 dan Persamaan (9.2).

Metode Pembantu Menyembunyikan Parameter Tambahan

Bawalah dua nilai terakhir sebagai parameter: satu panggilan rekursif per langkah, sekitar n panggilan — **rekursi ekor**.

Tetapi pemanggil yang menulis `fibonacci(10, 0, 0, 0)` mendapat jawaban salah. **Pembantu** `fib(n)` menyediakan parameter tambahannya (**Kode 9.3**).

Kode 9.3 `Fibonacci.java` (cuplikan): versi rekursi ekor dan metode pembantu yang dipakai pemanggil.

```java
// 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);
}
```

`count >= n - 1`, bukan `==`: dengan `==`, panggilan untuk $n = 1$ tidak pernah mencapai kasus dasarnya. *Kasus dasar hanyalah kasus dasar bila setiap rantai panggilan mencapainya.*

Buku: Subbab 9.2.3 dan Kode 9.3.

Banyaknya Panggilan, Diukur

Keluaran program lengkapnya (Kode 9.3 di buku) cocok persis dengan Persamaan (9.2): $2F(11) - 1 = 177$, $2F(21) - 1 = 21{,}891$, $2F(31) - 1 = 2{,}692{,}537$.

```
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
```

`@lru_cache` Python membuat definisi dua-panggilan itu *mengingat* setiap jawaban dalam sebuah kamus — kembali linear (**Kode 9.4**).

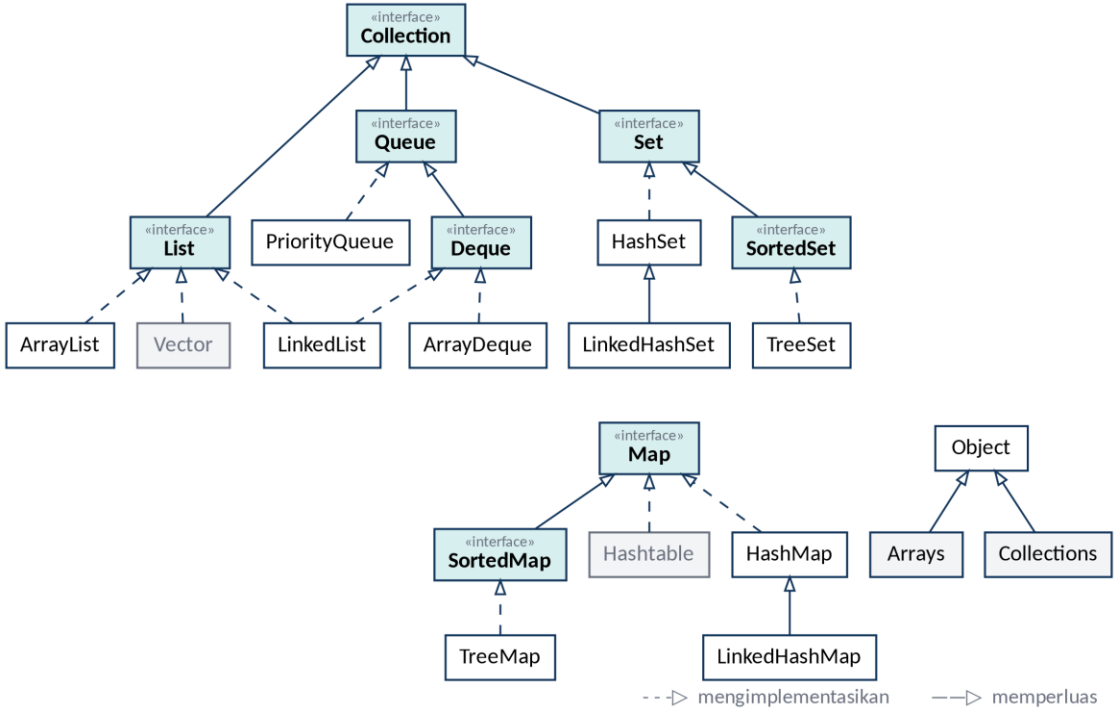
Kode 9.4 `fibonacci.py` (cuplikan): definisi standar yang dipercepat dengan `functools.lru_cache`.

```python
@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)
```

> **TIDAK ADA PENGHAPUSAN PANGGILAN EKOR.** Baik Java maupun Python tidak menghapus panggilan ekor; Python berhenti pada kedalaman sekitar 1000. Untuk sejuta langkah, tulislah perulangan.

Buku: Subbab 9.2.3 serta Kode 9.3 dan 9.4.

Collections Framework



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.

Gambar 9.2: `Collection` → `List`, `Queue` (→ `Deque`), `Set`; `Map` berdiri sendiri; `Arrays` dan `Collections` memuat metode pembantu statis.

Buku: Subbab 9.3 dan Gambar 9.2.

Tiga Keluarga: Set, List, Map

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`

Tabel 9.1: set menjawab *apakah ia ada di dalamku?*, list menyimpan semuanya di tempat Anda meletakkannya, map menjawab *apa pasangan kunci ini?*

Buku: Subbab 9.3.1 dan Tabel 9.1.

Memprogram terhadap Antarmuka

Tulislah `List<String> myList = new ArrayList<>();`, bukan `ArrayList<String> myList = ...`. Kode selebihnya hanya bergantung pada apa yang dapat dilakukan setiap `List`.

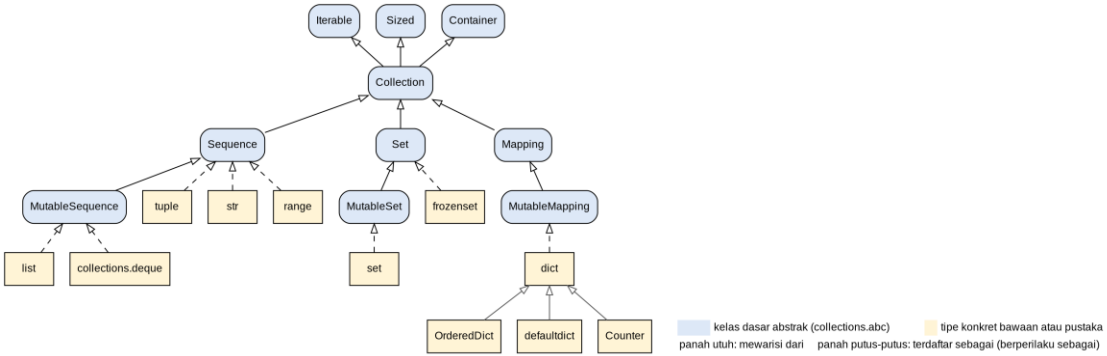
Bila kelak pengukuran lebih memihak `LinkedList`, ubahlah satu kata sesudah `new`; tidak ada bagian lain yang dapat rusak.

Begitu pula parameter: metode yang menerima `List<String>` menerima setiap list; metode yang menuntut `ArrayList<String>` menolak yang lain tanpa alasan.

`new ArrayList<>()` — **diamond** — membiarkan kompilator mengisi argumen tipenya (Java 7+).

Buku: Subbab 9.3.2.

Koleksi di Python



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

Gambar 9.3: `list`, `tuple`, `set`, `dict` punya sintaksis sendiri; `collections.abc` menamai jenis-jenisnya, dan tipe bawaan *didaftarkan* sebagai berperilaku seperti itu — *duck typing* yang diresmikan.

Buku: Subbab 9.3.3 dan Gambar 9.3.

List: Operasi Sehari-hari

`add` mengembalikan `true` bahkan untuk duplikat; operasi borongan menerima satu koleksi utuh (**Kode 9.5**).

Kode 9.5 `ListDemo.java` (cuplikan): operasi borongan dan posisi.

```
        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"));
```

`addAll` ≈ gabungan, `removeAll` ≈ selisih, `retainAll` ≈ irisan — tetapi list menghormati duplikat dan urutan: `[and, us, us]`.

`add(0, "we")` menyisipkan di depan, `set(1, "or")` mengganti, `indexOf("us")` menemukan kecocokan *pertama*: `[we, or, us, us] get(2) = us indexOf("us") = 2`.

Buku: Subbab 9.4.1 dan Kode 9.5.

List di Python

`append`, `insert`, pengindeksan, `index`, `len`, dan `in` mengerjakan tugas `add`, `add(i, x)`, `get`/`set`, `indexOf`, `size`, `contains`. Tidak ada `removeAll`/`retainAll`: cara Python adalah ***list comprehension*** (**Kode 9.6**).

Kode 9.6 `list_demo.py` (cuplikan): *comprehension* sebagai pengganti `removeAll` dan `retainAll`, serta posisi.

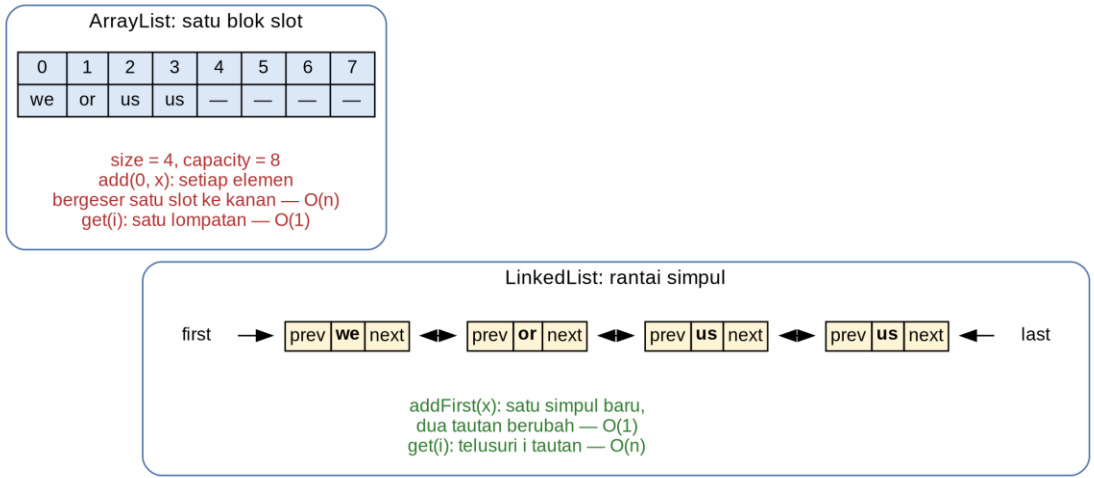
```
    my_list.extend(["love", "them"])
    print(my_list)
    my_list = [w for w in my_list if w not in {"love", "them", "dude"}]   # removeAll
    print(my_list)
    my_list = [w for w in my_list if w in {"and", "us", "love"}]          # retainAll
    print(my_list)

    # 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"))
```

`describe` melayani `list` dan `deque` tanpa antarmuka yang dideklarasikan — keduanya punya `pop` dan `len`: *duck typing*.

Buku: Subbab 9.4.1 dan Kode 9.6.

ArrayList versus LinkedList



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.

Gambar 9.4: `get(i)` bernilai $O(1)$ pada larik dan $O(n)$ pada rantai; menyisipkan di depan `ArrayList` menggeser semuanya. n sisipan di depan berbiaya Persamaan (9.3):

$$0 + 1 + 2 + \cdots + (n-1) = \frac{n(n-1)}{2} \tag{9.3}$$

Buku: Subbab 9.4.2, Gambar 9.4, dan Persamaan (9.3).

List Mana yang Menang?

`ArrayList`: alamat slot i dihitung dalam satu langkah, sehingga `get`/`set` bernilai $O(1)$; menambah di ujung rata-rata $O(1)$. Sisipan di depan: sekitar 5×10^9 pergeseran untuk $n = 100{,}000$.

`LinkedList`: menyisipkan di kedua ujung $O(1)$; tetapi `get(i)` menelusuri hingga $n/2$ simpul, sehingga perulangan berindeks atasnya berbiaya $O(n^2)$.

Dalam praktik `ArrayList` hampir selalu menang: slot yang bersebelahan dibaca cepat; setiap simpul tertaut adalah objek terpisah yang tersebar di memori.

Untuk kerja murah di *kedua* ujung, pakailah `ArrayDeque`. `list` Python adalah sebuah `ArrayList`; `deque`-nya melayani kedua ujung.

Buku: Subbab 9.4.2.

Menghapus Sambil Menelusuri: Jebakan Java

Tiga cara mengunjungi: lewat indeks, for-each (yang memakai `Iterator`), dan `Iterator` secara manual: `hasNext()`, `next()`.

Menghapus di dalam for-each: iterator tersembunyi melihat list berubah di belakangnya dan melempar eksepsi — **fail-fast**. Obatnya ada di **Kode 9.7**.

Kode 9.7 `IterationDemo.java` (cuplikan): menghapus melalui iterator itu sendiri, dan dengan `removeIf`.

```
        // 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);

        // 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);
```

```
for-each + remove: ConcurrentModificationException
words now: [you, us, and, us]
iterator.remove: [you, and]
removeIf: [you, and]
```

Buku: Subbab 9.4.3 dan Kode 9.7.

Jebakan Python Lebih Buruk

Di Python, menghapus di dalam perulangan `for` **tidak** memunculkan galat — dan meninggalkan jawaban yang salah (**Kode 9.8**).

Kode 9.8 `iteration_demo.py` (cuplikan): menghapus di dalam perulangan `for` diam-diam melewatkan elemen.

```
    # 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)
```

Telusuri `[you, us, us, and, us]`: pada indeks 1, `us` pertama dihapus dan `us` kedua bergeser ke indeks 1; perulangan maju ke indeks 2 dan tidak pernah melihatnya. Pada indeks 3, `remove("us")` menghapus `us` *pertama* — yang terlewat tadi. `us` terakhir selamat: `['you', 'and', 'us']`.

> **WAWASAN KUNCI.** Jangan pernah mengubah *struktur* koleksi selagi sebuah perulangan menelusurinya, kecuali melalui penelusurnya sendiri. Telusuri salinannya (`words[:]`), pakailah *comprehension*, `Iterator.remove`, atau `removeIf`. Mengganti elemen di tempat tidak berbahaya.

Buku: Subbab 9.4.3 dan Kode 9.8.

Set: Enam Kedatangan, Tiga Urutan

`add(3)` pada set yang sudah memuat 3 mengembalikan `false`. Aljabar himpunan: $\{1,2,3\} \cup \{1,\ldots,7\}$, $\{1,\ldots,7\} \setminus \{4,\ldots,10\} = \{1,2,3\}$, $\{1,2,3\} \cap \{2,3,4,5\} = \{2,3\}$.

Enam kedatangan yang sama dalam tiga jenis set, **Kode 9.9**:

Kode 9.9 `SetDemo.java` (cuplikan): nama-nama yang sama dalam `HashSet`, `LinkedHashSet`, dan `TreeSet`, serta kueri elemen terdekat.

```
        // 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"));
```

```
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]
```

Buku: Subbab 9.5.1 dan Kode 9.9.

Set di Python

Operator: `|` gabungan, `-` selisih, `&` irisan, `^` selisih simetris, masing-masing dengan bentuk di tempat seperti `|=`.

`set` berisi string beriterasi dalam urutan yang berubah *dari satu eksekusi ke eksekusi berikutnya* (hash diberi garam). Bila urutan penting, mintalah: `sorted(s)` ≈ `TreeSet`; `list(dict.fromkeys(items))` ≈ `LinkedHashSet` (**Kode 9.10**).

Kode 9.10 `set_demo.py` (cuplikan): urutan kemunculan pertama, urutan terurut, dan *ceiling* yang ditemukan dengan `bisect_left`.

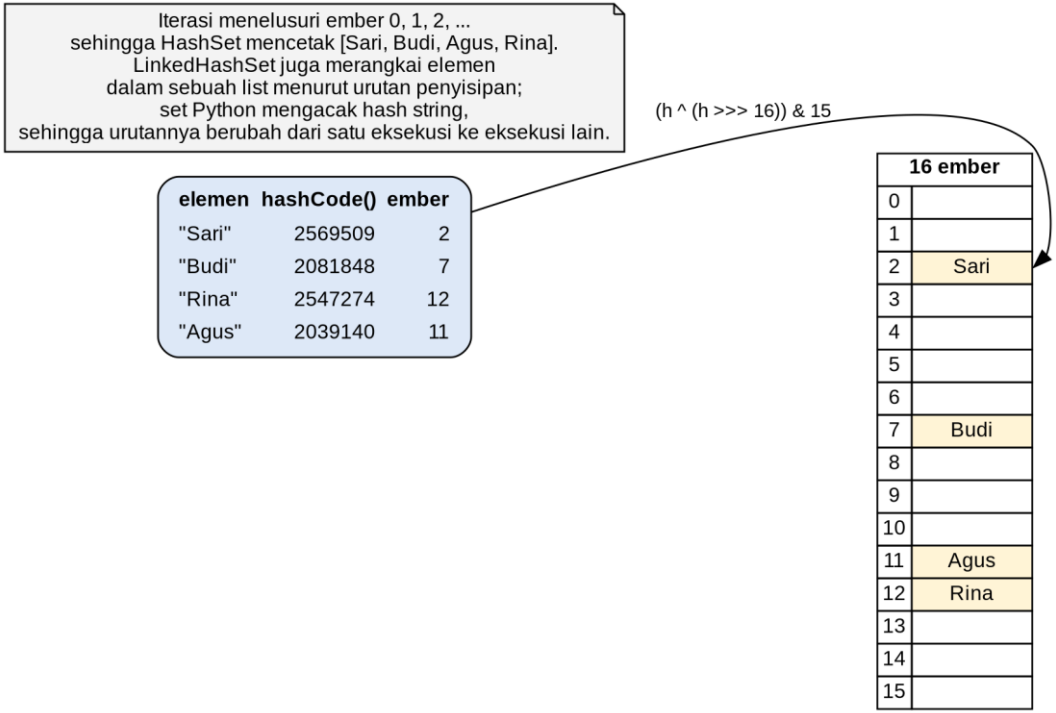
```
    arrivals = ["Sari", "Budi", "Rina", "Agus", "Budi", "Sari"]
    distinct = set(arrivals)                    # urutan iterasi dapat berubah dari eksekusi ke eksekusi
    print("set size   ", len(distinct))
    print("first-seen ", list(dict.fromkeys(arrivals)))   # seperti LinkedHashSet
    print("sorted     ", 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')]}")
```

`frozenset` imutabel dan dapat di-hash: ia dapat menjadi kunci `dict` atau elemen set lain.

Buku: Subbab 9.5.1 dan Kode 9.10.

Hashing: Menghitung, Bukan Mencari



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]`.

Gambar 9.5: `contains("Rina")` menghitung ember 12 dan hanya membandingkan di sana; iterasi menelusuri ember 2, 7, 11, 12.

Buku: Subbab 9.5.2 dan Gambar 9.5.

Contoh Tuntas: Ke Mana “Sari” Pergi?

Kode hash sebuah `String`: $s_0 \cdot 31^{k-1} + s_1 \cdot 31^{k-2} + \cdots + s_{k-1}$, dengan limpahan 32-bit.

`"Sari"`: $83 \cdot 31^3 + 97 \cdot 31^2 + 114 \cdot 31 + 105 = 2{,}472{,}653 + 93{,}217 + 3{,}534 + 105 = 2{,}569{,}509$.

Set baru punya $m = 16$ ember; ember $=$ `(h ^ (h >>> 16)) & 15` $= 2$ untuk `"Sari"`.

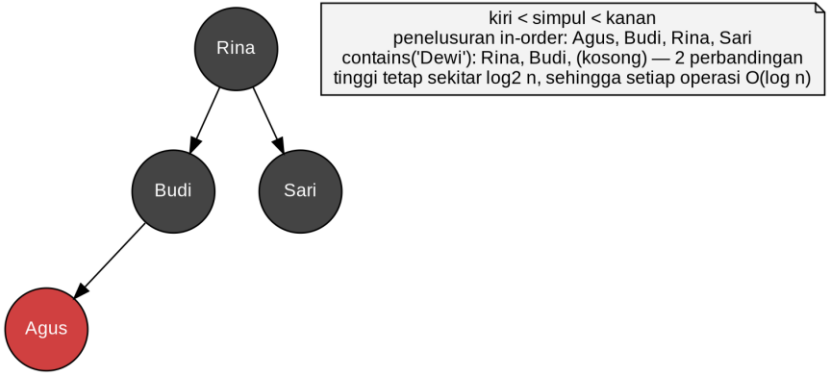
Panjang rantai rata-rata adalah **faktor muatan** Persamaan (9.4); Java menggandakan tabelnya bila $n > 0.75\,m$, sehingga berlaku Persamaan (9.5):

$$\alpha = \frac{n}{m} \tag{9.4}$$

$$T_{\text{hash}}(n) = O(1 + \alpha) = O(1) \tag{9.5}$$

Buku: Subbab 9.5.2 serta Persamaan (9.4) dan (9.5).

TreeSet Menjaga Urutan



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.

Gambar 9.6: untuk mencari `"Dewi"`, belok kiri di `Rina`, kanan di `Budi` — kosong, jadi tidak ada, sesudah dua perbandingan. Pohon merah-hitam menjamin Persamaan (9.6):

$$h \leq 2\log_2(n+1) \tag{9.6}$$

Buku: Subbab 9.5.3, Gambar 9.6, dan Persamaan (9.6).

Map: dari Kunci ke Nilai

`put` **mengembalikan nilai sebelumnya** (atau `null`); `get` mengembalikan `null` untuk kunci yang tidak ada; `getOrDefault` memberi nilai bawaan.

`containsKey` adalah pencarian hash; `containsValue` memeriksa setiap nilai, $O(n)$.

Menghitung dalam satu baris: `counts.merge(t, 1, Integer::sum)` (**Kode 9.11**).

Kode 9.11 `MapDemo.java` (cuplikan): menghitung dengan `merge` dalam `LinkedHashMap`, lalu hitungan yang sama dalam `TreeMap`.

```
        // 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));
```

```
LinkedHashMap {WASH=3, IRON=2, WASH_IRON=1}
TreeMap       {IRON=2, WASH=3, WASH_IRON=1}
```

Buku: Subbab 9.6.1 dan Kode 9.11.

dict Python

`d[key]` pada kunci yang tidak ada memunculkan **`KeyError`**; pakailah `d.get(key, default)`.

`dict` menjaga **urutan penyisipan** (Python 3.7+): sebuah `LinkedHashMap` secara bawaan.

Tiga cara menghitung, dan `defaultdict(list)` untuk mengelompokkan — padanan `computeIfAbsent` (**Kode 9.12**).

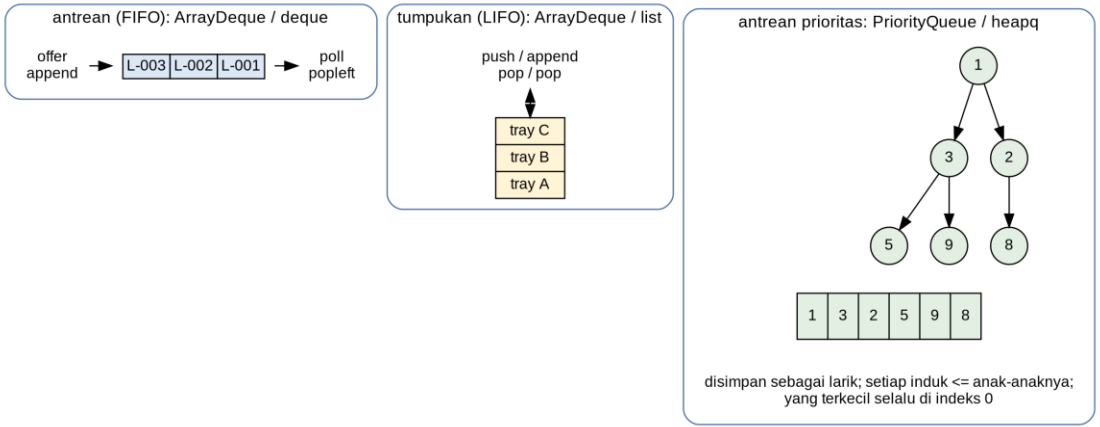
Kode 9.12 `map_demo.py` (cuplikan): menghitung dengan `get`, dengan `Counter`, dan mengelompokkan dengan `defaultdict`.

```
    # 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))
```

```
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)]
```

Buku: Subbab 9.6.1 dan Kode 9.12.

Antrean, Tumpukan, Antrean Prioritas



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.

Gambar 9.7: FIFO melepaskan yang tertua, LIFO yang terbaru, antrean prioritas yang terkecil — disimpan di indeks 0 larik heap.

Buku: Subbab 9.7 dan Gambar 9.7.

Contoh Tuntas: Binary Heap

`ArrayDeque` adalah larik melingkar: antrean *sekaligus* tumpukan, kedua ujung $O(1)$. Antrean prioritas adalah **binary heap** (**Kode 9.13**).

Kode 9.13 `QueueDemo.java` (cuplikan): `PriorityQueue` berisi enam bilangan, dicetak lalu diambil satu per satu sampai kosong.

```
        // 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());
```

```
heap array [1, 3, 2, 5, 9, 8]
poll order 1 2 3 5 8 9
```

`[1, 3, 2, 5, 9, 8]` *tidak* terurut: anak dari indeks i berada di $2i+1$ dan $2i+2$, dan setiap induk tidak lebih besar daripada anak-anaknya — maka indeks 0 adalah minimumnya. `offer`/`poll` $O(\log n)$, `peek` $O(1)$.

Buku: Subbab 9.7 dan Kode 9.15.

Antrean di Python

`collections.deque`: `append` dan `popleft`, keduanya $O(1)$. `list` biasa adalah tumpukan yang lazim.

`heapq` menerapkan aturan heap pada list biasa — larik yang sama, `[1, 3, 2, 5, 9, 8]` (**Kode 9.14**).

Kode 9.14 `queue_demo.py` (cuplikan): `heapq` sebagai antrean prioritas, dan bilangan yang dinegasikan untuk yang-terbesar-dulu.

```
    # 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))))

    # 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))
```

> **BUG ANTREAN PALING UMUM DI PYTHON.** `list.pop(0)` menggeser setiap elemen yang tersisa: $O(n)$ per panggilan. Pakailah `deque.popleft()`.

Buku: Subbab 9.7 dan Kode 9.16.

Generik: Definisinya

DEFINISI. Kelas generik ditulis dengan satu atau lebih **parameter tipe**, nama pengganti untuk tipe: `class GList<T>`. Pemakaiannya menyediakan **argumen tipe**, `GList<Integer>`, dan kompilator memeriksa setiap pemakaian seolah-olah `T` diganti dengan `Integer`. **Metode generik** mendeklarasikan parameter tipenya sendiri sebelum tipe kembaliannya: `static <T> T lastOf(List<T> list)`.

Satu kelas yang tidak peduli jenis objek yang dipegangnya — asalkan semuanya sejenis — dan kompilator yang mengikat setiap pemakai pada jenis yang dipilihnya.

Konvensi: `T` tipe, `E` elemen, `K` dan `V` kunci dan nilai.

Buku: Subbab 9.8.

Kehidupan sebelum Generik

Sebelum Java 5 (2004), `List` memegang `Object`. List mentah menerima `42` di antara nama-nama; galatnya baru muncul di *cast*, jauh dari kesalahannya (**Kode 9.15**).

Kode 9.15 `RawTypes.java` (cuplikan): `List` mentah menerima sebuah `Integer`, dan *cast*-nya gagal saat eksekusi.

```
    @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");
            }
        }
    }
```

Dengan `List<String>`, `names.add(42)` ditolak `javac`: *incompatible types: int cannot be converted to String*. Galat berpindah dari saat eksekusi, tempat pengguna menjumpainya, ke saat kompilasi, tempat pemrogram menjumpainya.

Buku: Subbab 9.8.1 dan Kode 9.17.

Kelas Generik: GList

List *bilangan bulat* Bab 8, ditulis sekali untuk sembarang tipe elemen (**Kode 9.16**).

Kode 9.16 `GListDemo.java` (cuplikan): kelas generik `GList<T>`, kedua konstruktornya, dan metode generik `nil()`.

```
// 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;
    }

    public static <T> GList<T> nil() {                  // metode generik
        return new GList<>();
    }
```

`toString` menyerahkan kerjanya kepada `toStringHelper(list, accumulator)` — pola rekursi-dengan-pembantu dari Subbab 9.2.3.

Buku: Subbab 9.8.2 dan Kode 9.19.

Satu Kelas, Dua Argumen Tipe

`GList<Integer>` memberi `[]`, `[10]`, `[50, 10]`; *kelas yang sama* memegang `String`, dan `towels.head()` mengembalikan `String` tanpa *cast*.

```
myGList: []
myGList2: [10]
myGList3: [50, 10]
towels: [white, blue], head white, tail [blue], size 2
IllegalStateException: Trying to access tail of an empty list
```

`nil()` bersifat statis, sehingga tidak dapat memakai `T` milik kelas: ia mendeklarasikan miliknya sendiri, `public static <T> GList<T> nil()`, dan kompilator menyimpulkan `T` dari konteks.

> **TIPE KEMBALIAN MENTAH MEMATIKAN PEMERIKSAAN.** `public static GList nil()` lolos kompilasi, dengan peringatan, dan diam-diam mematikan pemeriksaan bagi setiap pemanggil. Kompilasilah dengan `javac -Xlint:all` dan perlakukan setiap peringatan "unchecked" atau "rawtypes" sebagai bug.

Buku: Subbab 9.8.2 dan Kode 9.19.

Kelas yang Sama di Python

`T = TypeVar("T")` membuat variabel tipe; `class GList(Generic[T])` mendeklarasikan parameternya; `GList[int]`, `GList[str]` adalah tipe yang dipahami mypy (**Kode 9.17**).

Kode 9.17 `glist_demo.py` (cuplikan): `TypeVar`, `Generic[T]`, dan dua cara membangun list, `nil()` dan `cons()`.

```
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
```

Python 3.12 menambahkan `class GList[T]:`; kode ini memakai bentuk lama agar berjalan juga di Python 3.10 dan 3.11.

Buku: Subbab 9.8.2 dan Kode 9.20.

Metode Generik, dan Akhirnya sebuah Pair

`static <A, B> Pair<B, A> swap(Pair<A, B> p)`: "untuk sembarang tipe `A` dan `B` ...". Kompilator **menyimpulkan** `A` dan `B` dari argumennya (**Kode 9.18**).

Kode 9.18 `GenericMethods.java` (cuplikan): metode generik `swap` dan `lastOf`, serta `fibonacciPair` yang mengembalikan `Pair<Long, Long>`.

```
    // <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());
    }
```

`fibonacciPair(10)` kini tercetak `(55, 34)`, dengan bagian-bagian bernama `getFirst()` dan `getSecond()` alih-alih `[0]` dan `[1]`.

Buku: Subbab 9.8.3 dan Kode 9.21.

Parameter Tipe Terbatas

`a.compareTo(b)` di dalam `GList<T>` tidak lolos kompilasi: `T` bisa kelas *apa saja*. Katakanlah "sembarang `T` yang dapat dibandingkan dengan `T` lain": `<T extends Comparable<T>>` (**Kode 9.19**).

Kode 9.19 `GList2Demo.java` (cuplikan): metode generik terbatas `max`.

```
    // 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;
    }
```

`max(List.of(3, 9, 4))` bernilai 9, `max(List.of("wash", "dry", "iron"))` bernilai `wash`; `GList2<Object>` ditolak: *type argument Object is not within bounds of type-variable T*.

Python: `TypeVar("T", bound=SupportsLessThan)`, sebuah **protokol** dengan `__lt__` (Kode 9.24 di buku); mypy menolak list berisi `object` polos tepat di tempat list itu dibangun.

Buku: Subbab 9.8.4 serta Kode 9.23 dan 9.24.

Apakah List Sebuah List?

Tampaknya jelas. Ternyata salah. Seandainya `List<Animal> animals = cats` lolos kompilasi, `animals.add(new Dog(...))` akan menaruh seekor anjing ke dalam list kucing.

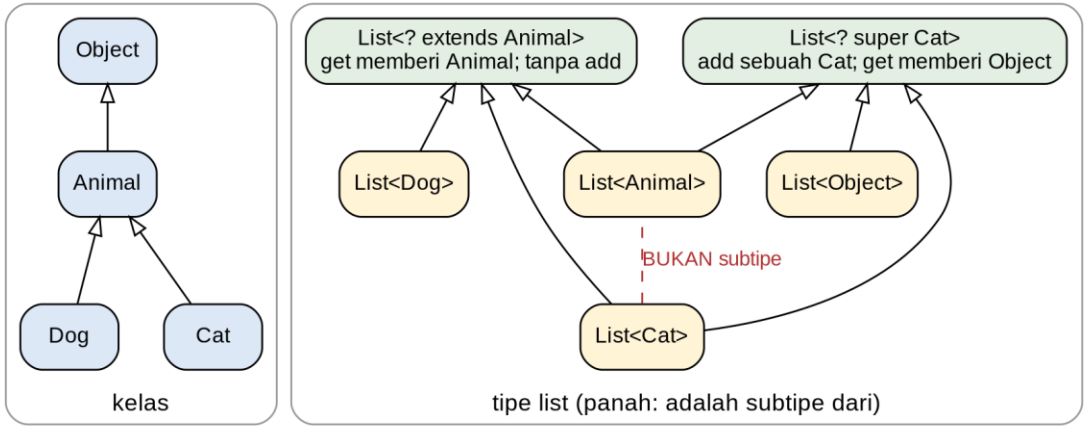
Maka tipe generik bersifat **invarian**, Persamaan (9.7) ($<:$ dibaca "adalah subtipe dari"):

$$Cat <: Animal \quad \not\Rightarrow \quad G\langle Cat\rangle <: G\langle Animal\rangle \tag{9.7}$$

Larik bersifat **kovarian** — `Cat[]` *adalah* `Animal[]` — sehingga `pets[0] = dog1` lolos kompilasi dan gagal saat eksekusi dengan `ArrayStoreException: Dog`.

Buku: Subbab 9.8.5 dan Persamaan (9.7).

Dua Jalan Keluar: 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>`.

Gambar 9.8: `? extends Animal` menerima list kucing dan anjing untuk **dibaca**; `? super Cat` menerima list kucing, hewan, dan objek untuk **ditulisi**.

Buku: Subbab 9.8.5 dan Gambar 9.8.

PECS: Producer extends, Consumer super

Tabel 9.2 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

Tabel 9.2: parameter yang hanya *diambil* isinya oleh metode adalah produsen, `? extends T`; parameter yang hanya *diisi* adalah konsumen, `? super T`.

Buku: Subbab 9.8.5 dan Tabel 9.3.

Wildcard Bekerja

`names` hanya membaca, sehingga menerima `List<? extends Animal>`; `addAll` membaca `src` dan menulisi `dest` (**Kode 9.20**).

Kode 9.20 `WildcardDemo.java` (cuplikan): parameter konsumen `? super T` dan parameter produsen `? extends Animal`.

```
    // 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();
    }
```

Dengan `T = Cat` atau `T = Dog`, `dest` boleh berupa `List<Animal>` atau `List<Object>`. `Collections.addAll` dan `Collections.copy` dideklarasikan dengan cara yang sama.

Python: `list[T]` invarian, `Sequence[T]` kovarian — `Sequence[Animal]` berperan sebagai `List<? extends Animal>`. Abaikan aturannya dan seekor anjing masuk ke antara kucing; mypy berkata *"list" is invariant — Consider using "Sequence"* (Kode 9.26 di buku).

Buku: Subbab 9.8.5 serta Kode 9.25 dan 9.26.

Type Erasure

Java memeriksa argumen tipe, menyisipkan *cast*-nya, lalu **menghapus** argumen itu: saat eksekusi `List<String>` hanyalah `ArrayList` (**Kode 9.21**).

Kode 9.21 `ErasureDemo.java` (cuplikan): satu kelas saat eksekusi, dan polusi heap melalui parameter mentah.

```
    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)");
        }
```

```
same class at run time: true
class name: java.util.ArrayList
size 2, contents [Siti, 42]
ClassCastException at get(1)
```

Tidak ada `new T()`, `new T[n]`, `instanceof List<String>`, atau `List<int>`. `Collections.checkedList` menolak elemen yang salah di pintu masuk. Python lebih jauh lagi: argumen tipe tidak pernah menjadi bagian program yang berjalan (Kode 9.28 di buku).

Buku: Subbab 9.8.6 serta Kode 9.27 dan 9.28.

Generik di Java dan Python

Tabel 9.3 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

Tabel 9.3: `javac` selalu menegakkan generik; mypy menegakkan petunjuk tipe Python ketika Anda menjalankannya.

Buku: Subbab 9.8.7 dan Tabel 9.4.

Comparable atau Comparator?

`Comparable<T>`: kelas menetapkan *urutan alaminya* dengan `compareTo`. `Comparator<T>`: objek terpisah menetapkan *sebuah* urutan dari luar — menurut umur, menurut nama, terbesar dulu.

Kontraknya: negatif, nol, atau positif; dan konsisten (bila a sebelum b dan b sebelum c, maka a sebelum c).

Kode 9.22 `MyCollection.java` (cuplikan): `AgeComparator`, comparator yang ditulis sebagai kelas.

```
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
        }
    }
}
```

Kode 9.22 untuk `s1` (umur 20) dan `s2` (umur 12): $20 - 12 = 8$, positif, jadi `s1` datang sesudah `s2`. Aman untuk umur — tidak secara umum.

Buku: Subbab 9.9.1 dan Kode 9.29.

Comparator untuk Pemakaian Sungguhan

Sejak Java 8, comparator *dibangun* dari field yang menjadi dasar pengurutan, lalu digabungkan (**Kode 9.23**).

Kode 9.23 `SortingDemo.java` (cuplikan): urutan alami, kunjungan menurun, serta kota lalu nama menurun.

```
        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()));
```

```
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)
```

Ketiga pelanggan dengan empat kunjungan tetap dalam urutan nama: pengurutan objek Java bersifat **stabil**.

Buku: Subbab 9.9.2 dan Kode 9.31.

Dua Jebakan Comparator

`TreeSet` dengan comparator memakai *comparator itu* untuk menentukan kesamaan: tiga pelanggan berkunjungan empat dianggap satu — `[Dewi(4), Agus(7), Budi(9)]`. Tambahkan pemecah seri sampai "dibandingkan sama" berarti "elemen yang sama".

Pengurangan melimpah. Untuk $a = -2{,}000{,}000{,}000$ dan $b = 2{,}000{,}000{,}000$, $a - b = -4 \times 10^9$ berputar menjadi $-4{,}000{,}000{,}000 + 2^{32} = 294{,}967{,}296$ — *positif*.

```
TreeSet by visits [Dewi(4), Agus(7), Budi(9)]
subtraction says low vs high = 294967296, Integer.compare says -1
```

Pakailah `Integer.compare(a, b)` atau `Comparator.comparingInt`, selalu.

Buku: Subbab 9.9.2 dan Kode 9.31.

Python Mengurutkan dengan Kunci

Pengurutan Python menerima **fungsi kunci** berargumen *satu*; `cmp_to_key` menyesuaikan fungsi perbandingan gaya Java (Kode 9.30 di buku).

Arah campuran pada *string*: tak ada bilangan untuk dinegasikan — manfaatkan kestabilan. Urutkan dulu menurut kunci *minor*, lalu menurut kunci *mayor* (**Kode 9.24**).

Kode 9.24 `sorting_demo.py` (cuplikan): kunjungan menurun dengan `attrgetter`, lalu dua pengurutan stabil untuk kota menaik dan nama menurun.

```
    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}")
```

Bilangan bulat Python tidak pernah melimpah: `low - high` hanyalah −4,000,000,000.

Buku: Subbab 9.9.1–9.9.2 serta Kode 9.30 dan 9.32.

Biaya Setiap Operasi di Java

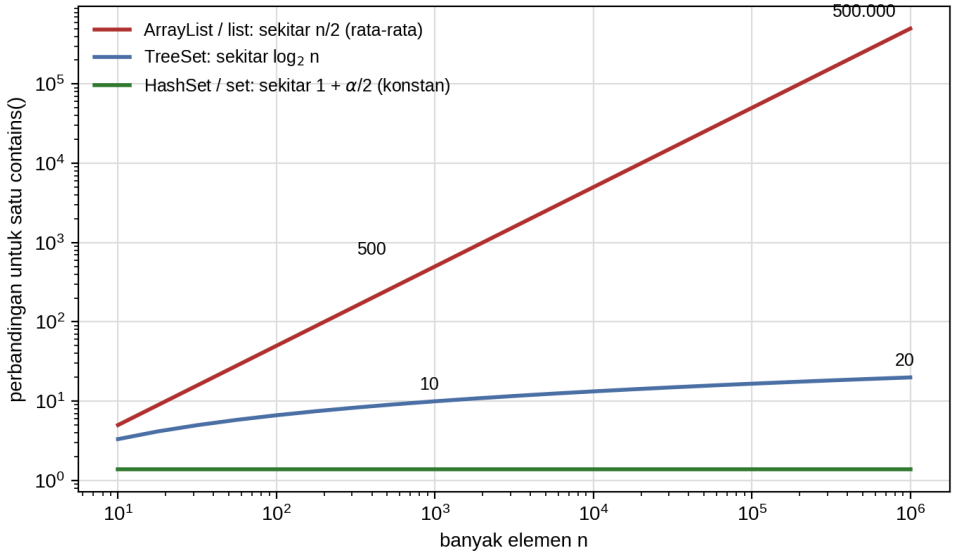
Tabel 9.4 Biaya operasi-operasi utama kelas koleksi Java untuk n elemen.

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 9.4: $O(1)$ terbatas, $O(\log n)$ bertambah satu langkah setiap kali n berlipat dua, $O(n)$ sebanding dengan n. `list`, `deque`, `set`, `dict`, dan `heapq` Python berbiaya sama dengan kembarannya di Java (Tabel 9.6 di buku).

Buku: Subbab 9.10 serta Tabel 9.5 dan 9.6.

Satu contains, Tiga Koleksi

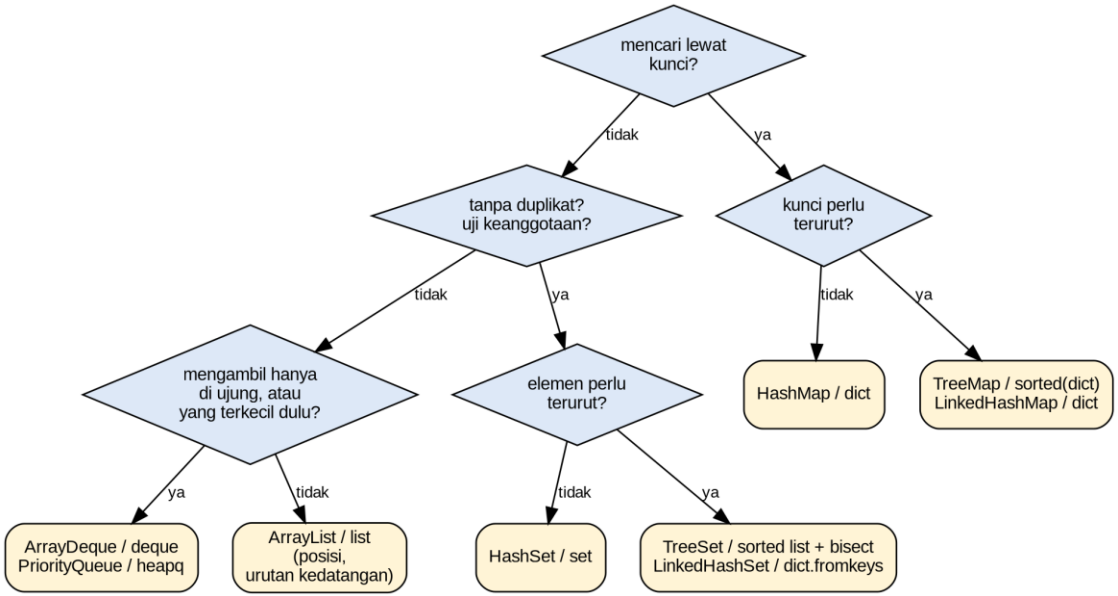


Gambar 9.9 Perbandingan yang dilakukan satu `contains` ketika n tumbuh (kedua sumbu logaritmik): sekitar $n/2$ untuk list, sekitar $\log_2 n$ untuk pohon seimbang, dan konstanta kecil untuk tabel hash dengan faktor muatan $\alpha \leq 0.75$.

Gambar 9.9: pada sejuta elemen, list memerlukan setengah juta perbandingan, pohon 20, tabel hash satu atau dua.

Buku: Subbab 9.10 dan Gambar 9.9.

Memilih dengan Empat Pertanyaan

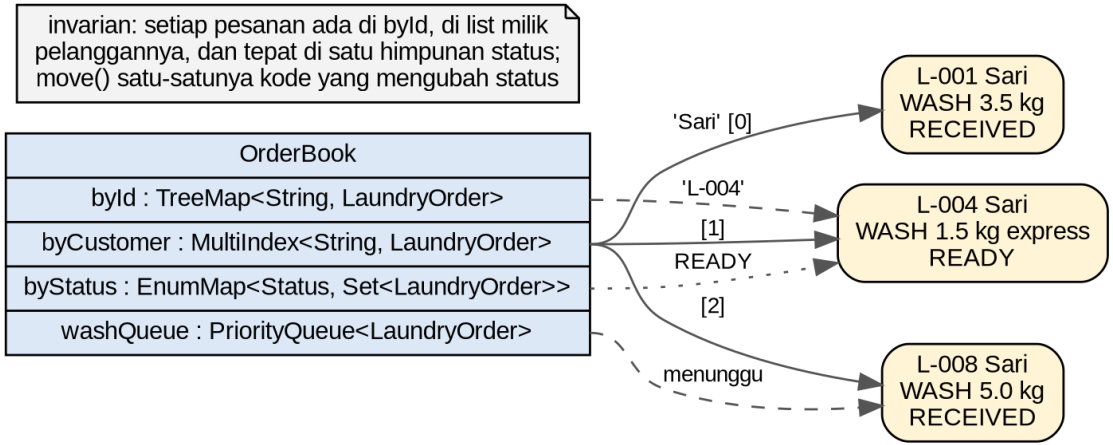


Gambar 9.10 Memilih koleksi dengan empat pertanyaan. Setiap daun menyebut kelas Java dan tipe Python yang menjawabnya.

Gambar 9.10: mencari menurut kunci? tanpa duplikat? hanya ujung-ujungnya, atau yang terkecil dulu? Bila semua jawabannya “tidak”, pakailah list. Ukurlah sebelum mengoptimalkan.

Buku: Subbab 9.10 dan Gambar 9.10.

Profitina Laundry: Empat Indeks



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.

Gambar 9.11: `TreeMap` menurut id, `MultiIndex` menurut pelanggan, `EnumMap` berisi `LinkedHashSet` menurut status, `PriorityQueue` untuk pencucian — semuanya atas objek-objek yang *sama*.

Buku: Subbab 9.11 dan Gambar 9.11.

Invarian yang Menjaga Kejujuran

Beberapa indeks tidak boleh saling berselisih: pesanan yang `READY` tetapi masih di himpunan `WASHING` memberi jawaban salah, secara diam-diam.

Kelas ini menjaga Persamaan (9.8), yang benar sesudah setiap metode publik:

$$\sum_{s \in Status} |byStatus[s]| = |byId| \quad \text{dan} \quad o \in byStatus[o.status] \text{ untuk setiap pesanan } o \tag{9.8}$$

Hanya satu metode privat, `move`, yang pernah mengubah status — dan ia memperbarui indeks pada saat yang sama (**Kode 9.25**).

Kode 9.25 `LaundryOrderBook.java` (cuplikan): `move`, satu-satunya tempat indeks status berubah.

```
    // 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);
    }
```

Penghapusan dilakukan *sebelum* field-nya berubah, selagi pesanan masih dapat ditemukan di himpunan lamanya.

Buku: Subbab 9.11, Persamaan (9.8), dan Kode 9.33.

Kilat Dulu, Lalu Siapa Cepat

Comparator antrean membandingkan `!o.isExpress()` (`false` terurut sebelum `true`), lalu nomor kedatangan (**Kode 9.26**).

Kode 9.26 `LaundryOrderBook.java` (cuplikan): keempat struktur `OrderBook`, semuanya `private final`, dan konstruktor yang memberi setiap status himpunan kosongnya sendiri.

```
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<>());
        }
    }
```

`receive` menolak id kembar *sebelum* menyentuh struktur mana pun, sehingga pesanan yang ditolak tidak meninggalkan jejak.

Buku: Subbab 9.11 dan Kode 9.33.

Antrean yang Sama di Python

`heapq` membandingkan dengan `<`, dan `LaundryOrder` tidak memilikinya. Maka setiap entri adalah tupel `(not order.express, order.arrival, order)`; nomor kedatangan yang unik membuat perbandingan tidak pernah sampai ke objek pesanan (**Kode 9.27**).

Kode 9.27 `laundry_order_book.py` (cuplikan): `receive` mendorong tupel prioritas; `start_next_wash` mengambilnya.

```
    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))

    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
```

Himpunan terurut: sebuah `dict` bernilai `None`; `MultiIndex(Generic[K, V])` mengembalikan *tupel*, potret yang imutabel (Kode 9.34 di buku).

Buku: Subbab 9.11 dan Kode 9.34.

Hasil Eksekusinya

```
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
```

Kilat `L-002, L-004, L-007` lebih dulu; `L-003` yang hanya disetrika tidak pernah masuk antrean. Hitungannya berjumlah $4 + 1 + 2 + 1 = 8$, sesuai tuntutan Persamaan (9.8).

Buku: Subbab 9.11 dan Kode 9.33.

Contoh Tuntas: Laporan Pendapatan dengan Tangan

Harga mengikuti Persamaan (9.9): WASH Rp7.000, IRON Rp5.000, WASH_IRON Rp10.000 per kg; kilat menambah separuhnya.

$$p = \mathrm{round}(w \cdot r) + e \cdot \left\lfloor \frac{\mathrm{round}(w \cdot r)}{2} \right\rfloor, \qquad e = \begin{cases} 1 & \text{kilat} \\ 0 & \text{lainnya} \end{cases} \tag{9.9}$$

Budi: round$(5.0 \times 10{,}000) = 50{,}000$, kilat 75,000; ditambah $4.0 \times 7{,}000 = 28{,}000$; jumlah 103,000.

Sari: $3.5 \times 7{,}000 = 24{,}500$; kilat 1.5 kg: $10{,}500 + 5{,}250 = 15{,}750$; $5.0 \times 7{,}000 = 35{,}000$; jumlah 75,250.

Agus 42,000, **Dewi** $25{,}000 + 12{,}500 = 37{,}500$, **Rina** $2.0 \times 5{,}000 = 10{,}000$. Total $267{,}750 = 10{,}000 + 145{,}250 + 112{,}500$ menurut layanan.

Buku: Subbab 9.11 dan Persamaan (9.9).

Profitina dalam Praktik

Profitina memindai situs web sebuah merek dan mengukur seberapa siap situs itu dikutip mesin penjawab AI (GEO). Produknya ditulis dalam Python.

Sebuah **himpunan** URL kanonis menjawab "sudah diambil?" dalam waktu harapan $O(1)$; halaman yang akan diambil menunggu dalam **antrean**, atau **antrean prioritas** bila sebagian lebih penting.

Map dari halaman ke hasil ukur, dari sinyal ke skor, dari minggu ke skor (dibaca terurut, diekspor sebagai CSV), dan penghitung "N baru" per kanal.

Petunjuk tipe seperti `dict[str, list[Signal]]` menyatakan apa yang dikembalikan fungsi; rangkaian lebih dari seribu uji adalah gerbang rilisnya. Hasil ukur memberi saran; penggunalah yang memutuskan.

Buku: Subbab 9.12.

Kesalahan yang Sering Terjadi

Mendeklarasikan kelas, bukan antarmuka; menghapus di dalam for-each; `get(i)` atas `LinkedList` atau `pop(0)` pada `list` Python; `contains` pada list untuk uji keanggotaan yang sering.

Mengandalkan urutan hash; mengubah field yang dipakai `hashCode` atau `compareTo` selagi objeknya berada di set, map, atau pohon.

Comparator dengan pengurangan; comparator yang menyebut objek-objek berbeda "sama" di dalam `TreeSet`; tipe mentah dan peringatan "unchecked" yang diabaikan.

Mengharapkan `List<Cat>` merupakan `List<Animal>`; mengira petunjuk tipe Python diperiksa saat eksekusi; `d[key]` atau `map.get(key).method()` pada kunci yang tidak ada; indeks yang saling menyimpang.

Buku: Subbab 9.13.

Ringkasan: Lima Hal untuk Diingat

1. Koleksi menjawab pertanyaan yang sering Anda ajukan; memprogramlah terhadap antarmuka, `List`, `Set`, `Map`, `Queue`, `Deque`.

2. Set dan map hash: waktu harapan $O(1)$ karena faktor muatannya dijaga di bawah 0,75; pohon: terurut, $O(\log n)$; list: posisi, pencarian $O(n)$; heap: yang terkecil dulu dalam $O(\log n)$.

3. Jangan pernah mengubah struktur koleksi selagi menelusurinya — kecuali melalui iteratornya.

4. Generik memindahkan galat ke saat kompilasi; tipe generik invarian, wildcard memulihkan keluwesan (PECS), dan Java menghapusnya sementara Python tidak pernah memilikinya saat eksekusi.

5. Urutan berasal dari `Comparable`, `Comparator`, atau `key=`; pengurutan bersifat stabil; satu kelas dan satu invarian menjaga beberapa indeks tetap jujur.

Buku: Subbab 9.14.

Pertemuan Berikutnya

Bab 10 — Antarmuka Grafis dan Penanganan Peristiwa. Program berhenti memegang kendali: sebuah jendela menunggu, dan penggunalah yang menentukan apa yang terjadi berikutnya.

Swing dan Tkinter berdampingan; *event loop* dijelaskan dari nol; listener, tata letak, penggambaran, kelas dalam, dan lambda — lambda yang hari ini kita jumpai pada `removeIf` dan comparator.

Sebelum pertemuan: kerjakan Soal Latihan 9.1, 9.4, 9.7, dan 9.8, lalu bacalah Subbab 10.1 sampai 10.4.

Buku: Subbab 9.15 dan Bab 10.

BAB 1

Kuliah 10 — Antarmuka Grafis dan Penanganan Peristiwa

Pemrograman Berorientasi Objek (EF234302)

Buku: Irfan Subakti, *Pemrograman Berorientasi Objek dengan Java dan Python*

Dosen: Irfan Subakti — yifana@gmail.com

Buku: Bab 10, Antarmuka Grafis dan Penanganan Peristiwa. Setiap halaman handout ini diakhiri dengan penunjuk ke buku, yang menjadi rujukan utama; handout ini adalah bab tersebut dalam bentuk ringkas.

Pertemuan Hari Ini

1. Siapa yang memegang kendali: dari konsol ke jendela; AWT, Swing, dan Tkinter; jendela pertama dan uji mandirinya (Subbab 10.2–10.5).

2. Gelung peristiwa dijelaskan dari nol, dan gelung mainan yang dapat Anda jalankan (Subbab 10.6).

3. Mendengarkan peristiwa, dan menata jendela dengan layout manager, `pack`, dan `grid` (Subbab 10.7–10.8).

4. Melukis, dan gaya-gaya listener: kelas dalam, kelas anonim, lambda, closure — dan apa yang diingat masing-masing (Subbab 10.9–10.10).

5. Aturan thread, dan jendela pencatatan pesanan Profitina Laundry dalam Swing dan Tkinter (Subbab 10.11–10.14).

Buku: Bab 10, Subbab 10.1 sampai 10.15.

Di Mana Kita Berhenti

Setiap program sejauh ini mulai dari puncak `main`, bekerja menurut urutan yang ditulis, mencetak, lalu berhenti. Bahkan dengan masukan (Bab 5), programlah yang memegang kendali.

Bab 7 memberi Profitina Laundry kelas-kelas layanan yang polimorfis; Bab 9 menyimpan banyak objek dalam koleksi yang aman-tipe.

Petugas di loket tidak memakai terminal. Mereka memakai **jendela**, dan mereka mengeklik di mana saja, dalam urutan apa saja.

> **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 gelung yang menunggu sesuatu terjadi.

Buku: Subbab 10.1.

Dari Konsol ke Jendela

Konsol mengajukan lima pertanyaan dalam urutan tetap; berubah pikiran soal *kilat* sesudah berat diketik, dan tidak ada jalan kembali.

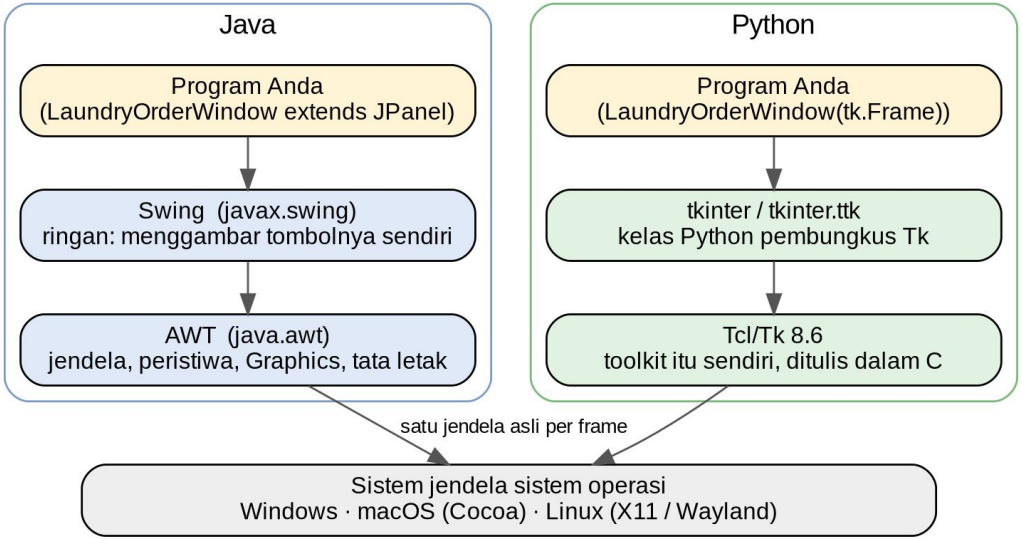
Jendela menampilkan kelima kotak sekaligus; programnya adalah sekumpulan penangan kecil: *bila berat berubah, hitung ulang harga; bila* Add order *diklik, validasi…*

> **DEFINISI. Peristiwa** (*event*) adalah apa saja yang mungkin ingin ditanggapi program: klik, tekanan tombol, gerakan tetikus, jendela yang tersingkap. **Penangan peristiwa** (*listener* di Java, *callback* di Python) adalah metode yang dipanggil toolkit ketika peristiwa itu terjadi. Program yang tersusun seperti ini bersifat **berbasis peristiwa**; pustakalah yang memanggil kode Anda — **pembalikan kendali**.

Akibatnya: **penangan harus singkat**; **keadaan tinggal di objek**, bukan di posisi dalam kode; jendela **dideskripsikan**, bukan dinarasikan.

Buku: Subbab 10.2.

Lapisan di Bawah Jendela



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.

Gambar 10.1: Swing bertumpu pada AWT, yang meminta jendela asli kepada sistem operasi; `tkinter` adalah lapisan Python yang tipis di atas Tcl/Tk.

Buku: Subbab 10.3 dan Gambar 10.1.

AWT, Swing, dan Tkinter

AWT (`java.awt`, JDK 1.0): *peer* asli yang *heavyweight*. Tetap menjadi fondasi: jendela, kelas peristiwa dan antarmuka listener (`java.awt.event`), `Graphics`, warna, layout manager.

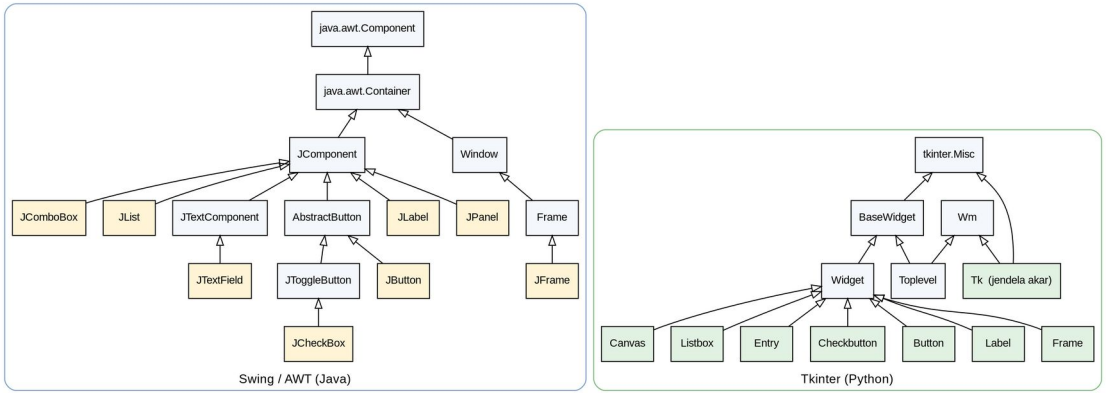
Swing (`javax.swing`, JDK 1.2): komponen *lightweight* yang menggambar dirinya sendiri; nama kelasnya diawali **J** — `JFrame`, `JButton`, `JLabel`. *Dalam program Swing, pakailah hanya kelas-kelas J.*

Tkinter: antarmuka Python ke Tk; satu akar `Tk` per program, setiap widget dibuat *di dalam* induk yang diberikan sebagai argumen pertama; `tkinter.ttk` menambahkan widget bertema.

> **TKINTER TIDAK ADA?** Di Debian/Ubuntu `sudo apt install python3-tk`; di Fedora `sudo dnf install python3-tkinter`. `ModuleNotFoundError: No module named 'tkinter'` berarti itu, bukan kode Anda.

Buku: Subbab 10.3.1–10.3.3.

Dua Hierarki Kelas



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.

Gambar 10.2: `JFrame` diturunkan dari `Window` milik AWT; setiap komponen Swing lainnya dari `JComponent`; setiap komponen juga sebuah `Container`. Rancangan Tkinter lebih datar.

Buku: Subbab 10.3.2 dan Gambar 10.2.

Jendela Pertama Anda di Swing

Kode 10.1 memuat enam langkah yang dilakukan hampir setiap program Swing.

Kode 10.1 `FirstWindow.java` (cuplikan): enam langkah `createAndShow`.

```
    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);
    }
```

`EXIT_ON_CLOSE`: tanpanya, menutup jendela hanya *menyembunyikan* frame dan program tampak macet. `pack()` lebih baik daripada `setSize`: ukuran yang diinginkan bergantung pada fon.

`main` menyerahkan `createAndShow` kepada `SwingUtilities.invokeLater` lalu selesai — namun program terus berjalan: **event dispatch thread** tetap hidup.

Buku: Subbab 10.4 dan Kode 10.1.

Uji Mandiri: Jendela Tanpa Layar

Dijalankan dengan `--selftest`, setiap program membangun komponennya persis seperti biasa, mengendalikannya dari kode, mencetak temuannya, dan tidak pernah membuka jendela.

`System.setProperty("java.awt.headless", "true")` didahulukan: komponen ringan tetap berfungsi, sedangkan `JFrame` akan melempar `HeadlessException`.

```
headless: true
components: 1
label text: Hello, Profitina Laundry!
on the EDT: true
self-test passed
```

Tk tidak punya mode *headless*: di server Linux, jalankan `xvfb-run -a python3 first_window.py --selftest`.

> **WAWASAN KUNCI.** Mode uji mandiri hanya berbiaya belasan baris dan berkali-kali membayar dirinya: ia memaksa jendela dibangun oleh sebuah metode, memeriksa perilaku dari kode, dan memungkinkan server (atau buku) menjalankan GUI Anda.

Buku: Subbab 10.4.1.

Jendela yang Sama di Tkinter

`tk.Tk()` adalah akarnya; `tk.Label(root, ...)` dibuat *di dalamnya*; `pack()` menempatkannya dan mengatur ukuran jendela (Kode 10.2).

Kode 10.2 `first_window.py` (cuplikan): membangun jendela di dalam akar.

```
def build(root):
    root.title("First window")
    label = tk.Label(root, text="Hello, Profitina Laundry!", padx=20, pady=10)
    label.pack()
    return label
```

Kode 10.3 `first_window.py` (cuplikan): `main`, yang menyerahkan kendali kepada gelung peristiwa.

```
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
```

Kode 10.3: `root.mainloop()` menyerahkan kendali kepada gelung peristiwa Tk; ia baru kembali ketika jendela ditutup. Uji mandirinya justru memanggil `root.withdraw()` dan tidak pernah menampilkan jendela.

Buku: Subbab 10.4 dan Kode 10.2.

Komponen, Kontainer, Panel

Jendela adalah **pohon**: frame di akar, kontainer sebagai simpul dalam, kontrol sederhana sebagai daun.

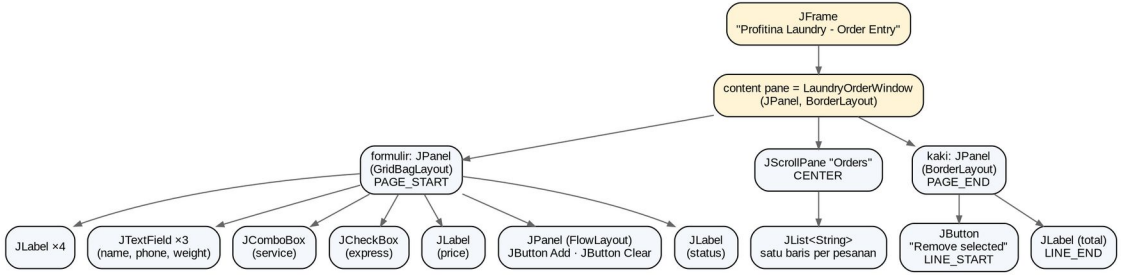
`JFrame` menampung beberapa *pane*; komponen Anda masuk ke **content pane** — atau bangun `JPanel` sendiri dan pasang dengan `frame.setContentPane(pane)`.

Praktik yang baik, dipakai untuk setiap jendela di bab ini: *setiap bagian penting jendela adalah kelas yang memperluas `JPanel`* (Tkinter: `tk.Frame`), dengan komponen-komponen di field dan dibangun di konstruktor.

Membaca jendela sebagai pohon adalah kunci menatanya: setiap kontainer menempatkan anak-anaknya sendiri, tidak lebih.

Buku: Subbab 10.5.1–10.5.3.

Pohon Komponen Jendela Sungguhan



Gambar 10.3 Pohon komponen jendela pencatatan pesanan Profitina Laundry. Setiap kontainer hanya menata anak-anaknya sendiri, dengan layout manager-nya sendiri.

Gambar 10.3: jendela pencatatan pesanan laundry dari Subbab 10.12 — sebuah formulir, sebuah list bergulir, dan sebuah kaki, masing-masing dengan tata letaknya sendiri.

Buku: Subbab 10.5.3 dan Gambar 10.3.

Gelung Peristiwa

Sistem operasi mengubah tekanan tombol tetikus, tombol papan ketik, atau jendela yang tersingkap menjadi **peristiwa** dan mengantrekannya untuk program pemilik jendela.

Di dalam setiap toolkit berjalan Pseudokode 10.1:

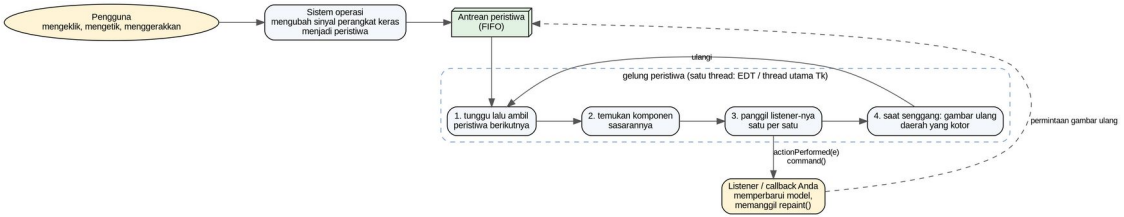
Pseudokode 10.1 Gelung peristiwa yang dijalankan setiap toolkit GUI: tunggu, ambil peristiwa berikutnya, panggil listener yang terdaftar untuknya, dan gambar ulang saat senggang.

```
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
```

Baris 3: menunggu, bahkan tanpa memakai prosesor. Baris 4–7: **dispatch**. Baris 7 adalah *satu-satunya* tempat kode Anda berjalan sesudah `main` — dan selama ia berjalan, tidak ada hal lain yang terjadi.

Buku: Subbab 10.6.1–10.6.2 dan Pseudokode 10.1.

Riwayat Sebuah Peristiwa



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.

Gambar 10.4: listener yang mengubah apa yang harus ditampilkan hanya *meminta* penggambaran ulang; gelung menggambar ulang saat senggang (baris 8–9).

Buku: Subbab 10.6.2 dan Gambar 10.4.

Gelung Peristiwa yang Dapat Anda Baca

Widget tiruan dengan daftar listener, sebuah antrean, dan gelung Pseudokode 10.1 hampir baris demi baris (Kode 10.4).

Kode 10.4 `toy_event_loop.py` (cuplikan): gelung peristiwa buatan tangan — tunggu, ambil, salurkan.

```
    # 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}")
```

Pendaftaran tidak menjalankan apa pun: `button.add_listener(on_click)` hanya menyimpan fungsinya, untuk dipanggil nanti.

Listener berbagi keadaan melalui objek: `typed` adalah list satu elemen (Java: larik satu elemen) — Subbab 10.10 menjelaskan alasannya.

Buku: Subbab 10.6.3 dan Kode 10.4.

Contoh Tuntas: Menelusuri Gelung Mainan

```
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
```

$4 \times 7000 = 28{,}000$: listener klik tidak mencetak apa pun; ia *mengantrekan penggambaran ulang*. Putaran berikutnya menemukan peristiwa itu sudah di antrean dan menyalurkannya sebelum meminta apa pun dari pengguna. Lalu $3 \times 7000 = 21{,}000$, total $49{,}000$; `close` memutus gelung.

Buku: Subbab 10.6.3 serta Kode 10.3 dan 10.4.

Mendengarkan Peristiwa

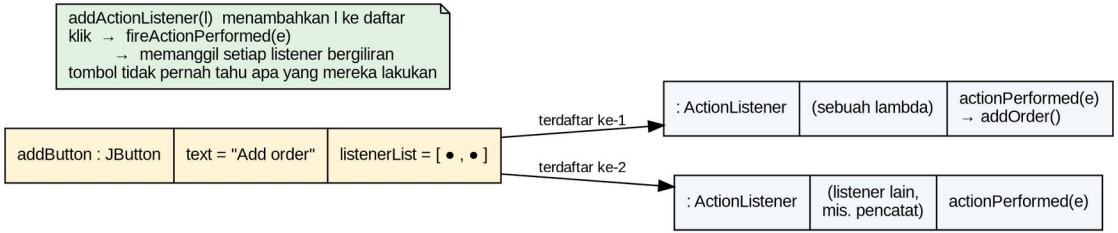
Di Java, tiga langkah: **deklarasikan** (`implements ActionListener`), **daftarkan** (`addActionListener`), **definisikan** (`actionPerformed(ActionEvent e)`).

Tombol tidak peduli apa kelas Anda — hanya bahwa Anda berjanji memiliki `actionPerformed`. Antarmuka Bab 6 sedang bekerja.

Objek peristiwa menyatakan apa yang baru terjadi: `e.getSource()` (komponen mana), `e.getPoint()`, `e.getX()` untuk tetikus, `e.getKeyCode()` untuk papan ketik.

Buku: Subbab 10.7 dan 10.7.2.

Sebuah Tombol dan Para Listener-nya



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.

Gambar 10.5: subjek yang memberi tahu sekumpulan pengamat tanpa tahu apa yang mereka lakukan — pola **Observer**, tulang punggung setiap toolkit.

Buku: Subbab 10.7 dan Gambar 10.5.

Membaca Field Teks

Komentar bernomor pada Kode 10.5 menandai perubahan dibandingkan panel yang hanya menampilkan; perubahan (1) adalah impor `ActionEvent` dan `ActionListener` dari `java.awt.event`.

Kode 10.5 `EchoPanel.java` (cuplikan): deklarasikan, daftarkan, definisikan.

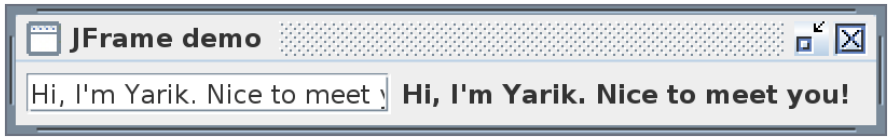
```
public class EchoPanel extends JPanel
        implements ActionListener {         // (2) deklarasikan: panel ini mendengarkan
    private final JTextField t = new JTextField(15);
    private final JLabel l = new JLabel("Write something on text-field & enter");

    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());
    }
```

Buku: Subbab 10.7.1 dan Kode 10.5.

Jendela Gema



Gambar 10.6 Jendela gema setelah pengguna mengetik sapaan di field teks dan menekan Enter: listener telah menyalin teksnya ke label.

Gambar 10.6: sesudah Enter. Uji mandirinya memanggil `t.postActionEvent()` — persis yang dilakukan Enter — dan menemukan satu listener pada field itu.

Buku: Subbab 10.7.1 dan Gambar 10.6.

Tkinter: bind dan Objek Peristiwa

Tanpa antarmuka listener: `widget.bind("<Return>", callback)` menempelkan objek apa pun yang dapat dipanggil pada sebuah **urutan peristiwa** — `"<Button-1>"`, `"<B1-Motion>"`, `"<KeyPress-a>"` (Kode 10.6).

Kode 10.6 `echo_panel.py` (cuplikan): subkelas `tk.Frame` yang mengikat Enter ke sebuah *bound method*.

```
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())
```

Tk memanggil callback dengan sebuah **objek peristiwa**: `event.widget`, `event.x`, `event.y`, `event.keysym`.

Jendela yang disembunyikan tidak menerima peristiwa papan ketik, sehingga uji mandiri memanggil `panel.on_enter(None)` sendiri; tombol menyediakan `invoke()`.

Buku: Subbab 10.7.1 dan Kode 10.6.

Peristiwa Item: Kotak Centang

`implements ItemListener`, `cb.addItemListener(this)`, `itemStateChanged(ItemEvent e)` — satu listener untuk kedua kotak (Kode 10.7).

Kode 10.7 `FruitChoice.java` (cuplikan): satu listener item yang bertanya kotak mana yang berubah.

```
    @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");
        }
    }
```

```
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
```

`doClick()` melakukan klik melalui jalur yang persis sama dengan tetikus, termasuk memanggil listener. Peristiwanya menyatakan apa yang baru terjadi; `getSource()` menyatakan komponen mana.

Buku: Subbab 10.7.2–10.7.3 dan Kode 10.7.

Jendela Kotak Centang



Gambar 10.7 Jendela kotak centang setelah *Mango* dicentang: listener item telah menulis ulang label penjelasan.

Gambar 10.7: sesudah *Mango* dicentang. Di Tkinter setiap `Checkbutton` menyimpan keadaannya dalam `BooleanVar` dan mendapat lambda `command`-nya sendiri — lambda itulah jawaban atas “kotak yang mana?” (Kode 10.8 di buku).

Buku: Subbab 10.7.3 dan Gambar 10.7.

Empat Jenis Peristiwa

Tabel 10.1 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)`

Tabel 10.1: sebuah panel boleh mendengarkan beberapa jenis sekaligus — `ShadeBox` menangani peristiwa aksi dan tetikus.

Buku: Subbab 10.7.4 dan Tabel 10.2.

Tata Letak dengan Aturan, Bukan Piksel

"Pada piksel (192, 82), lebar 104 piksel" salah di komputer berikutnya: fon lain, bahasa lain, jendela yang lebih lebar.

Java: setiap kontainer punya **layout manager**, yang memakai ukuran *diinginkan*, *minimum*, dan *maksimum* setiap anaknya. Tkinter: **geometry manager** — `pack`, `grid`, `place`.

FlowLayout, bawaan `JPanel`: kiri ke kanan seperti kata, baris baru bila penuh, ukuran yang diinginkan dipertahankan (Gambar 10.8).



Gambar 10.8 Lima tombol dalam `FlowLayout`: masing-masing mempertahankan ukuran yang disukainya, dan barisnya mengalir dari kiri ke kanan.

Buku: Subbab 10.8 dan 10.8.1, Gambar 10.8.

BorderLayout: Lima Wilayah

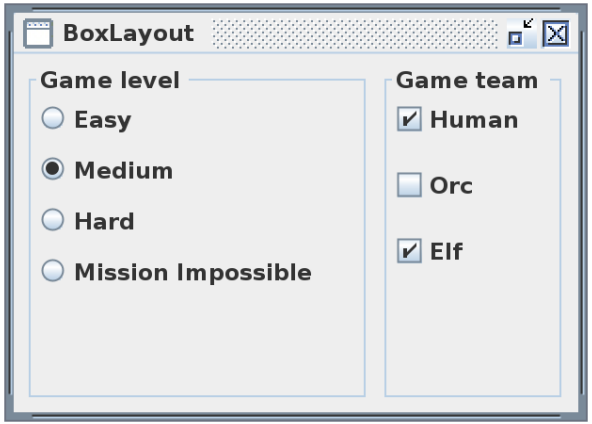


Gambar 10.9 Lima wilayah `BorderLayout`. Wilayah tengah mengambil seluruh ruang yang disisakan keempat tepi.

Gambar 10.9: bawaan *content pane*. Atas dan bawah mengambil lebar penuh, kiri dan kanan sisa tinggi, pusat semua sisanya.

Buku: Subbab 10.8.2 dan Gambar 10.9.

BoxLayout: Satu Baris atau Satu Kolom

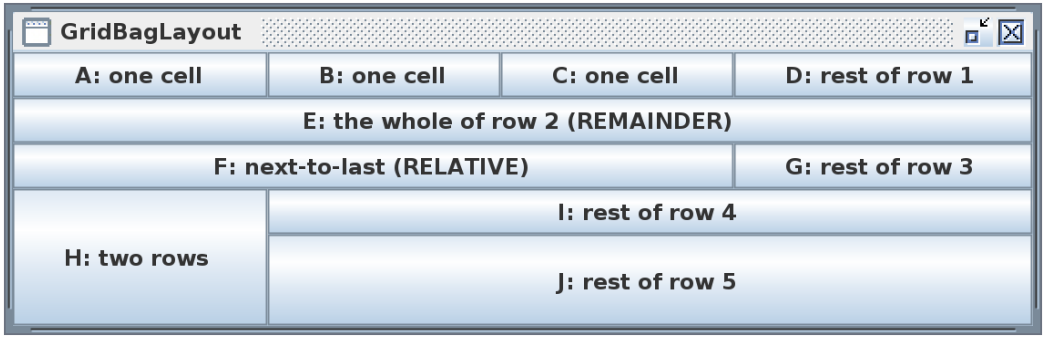


Gambar 10.10 Dua kotak tegak di dalam sebuah kotak mendatar. Tombol radio di kiri tergabung dalam satu `ButtonGroup`, sehingga hanya satu yang dapat dipilih; kotak centang di kanan saling bebas.

Gambar 10.10: `Box.createVerticalBox()`, *strut* untuk celah tetap, *glue* sebagai pegas.

Buku: Subbab 10.8.3 dan Gambar 10.10.

GridBagLayout: yang Paling Luwes



Gambar 10.11 Sebuah `GridBagLayout` tempat tombol-tombol merentang satu sel, sisa satu baris, atau dua baris, sesuai `gridwidth`, `gridheight`, dan bobotnya.

Gambar 10.11: kisi sel dengan `GridBagConstraints` — `gridx`, `gridwidth`, `weightx`, `fill`, `anchor`. Lupa menyetel ulang batasan yang dipakai ulang adalah bug klasiknya; sembunyikan di dalam metode pembantu.

Buku: Subbab 10.8.4 dan Gambar 10.11.

Tkinter: pack, grid, dan place

`pack(side=..., fill=..., expand=...)` menumpuk widget pada sisi-sisi induknya, menurut urutan pemanggilan `pack`.

`grid(row=..., column=..., sticky="ew")` ditambah `columnconfigure(1, weight=1)`: `GridBagLayout` tanpa objek batasan.

`place(x=..., y=...)`: koordinat mutlak — jarang merupakan gagasan yang baik.

> **JANGAN PERNAH MENCAMPUR pack DAN grid PADA SATU INDUK.** Tk 8.6 menolak: `TclError: cannot use geometry manager grid inside . which already has slaves managed by pack`. Satu pengelola per induk; sarangkan frame bila Anda memerlukan keduanya.

Buku: Subbab 10.8.5.

Contoh Tuntas: Mengamati BorderLayout Bekerja

Uji mandiri `LayoutTour.java` memberi ukuran pada setiap panel, memanggil `doLayout()`, dan mencetak batas-batasnya (Kode 10.9 di buku; cuplikan keluarannya):

```
  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
```

Atas dan bawah: lebar penuh 460 dengan tinggi 25. Kiri dan kanan mempertahankan lebar yang diinginkan, 187 dan 120, dan meregang setinggi sisa $150 - 25 - 25 = 100$. Pusat mendapat $460 - 187 - 120 = 153$.

Di Tkinter, `pack(side=LEFT)` **tidak** membungkus: pada lebar 230 piksel, *Long-Named Button 4* dan *5* sama sekali tidak mendapat ruang.

Buku: Subbab 10.8.6 dan Kode 10.9.

Memilih Tata Letak

Tabel 10.2 Layout manager Swing dan geometry manager Tkinter untuk kebutuhan umum.

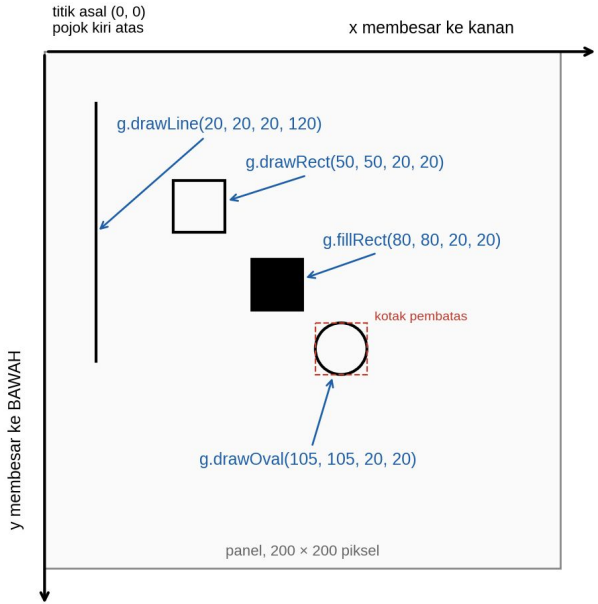
Kebutuhan	Swing	Tkinter
Baris yang membungkus, seperti kata	`FlowLayout` (bawaan untuk `JPanel`)	tidak ada padanan langsung; `grid` dengan baris yang dihitung sendiri
Satu baris atau satu kolom	`BoxLayout` / `Box`	`pack(side=LEFT)` / `pack(side=TOP)`
Tengah ditambah empat tepi	`BorderLayout` (bawaan 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)`

Tabel 10.2: jendela sungguhan dibangun dengan **penyarangan** — `BorderLayout` untuk bentuknya, formulir `grid` di satu wilayah, sebaris tombol di dalam formulir.

Buku: Subbab 10.8.6 dan Tabel 10.3.

Melukis: Sistem Koordinat

Override `protected void paintComponent(Graphics g)`: `g` adalah pena yang sudah diarahkan ke panel. Titik asalnya di **kiri atas**, dan y tumbuh **ke bawah** (Gambar 10.12).



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.

Buku: Subbab 10.9.1 dan Gambar 10.12.

Warna, dan Siapa yang Memanggil paintComponent

RGB: merah, hijau, biru, masing-masing 0–255; jumlah yang sama menghasilkan abu-abu. Persamaan (10.1) menghitung banyaknya warna:

$$256 \times 256 \times 256 = 256^3 = 16,777,216 \tag{10.1}$$

Anda tidak pernah memanggil `paintComponent`. Swing memanggilnya ketika jendela muncul, dipulihkan, tersingkap, atau diubah ukurannya; ia harus melukis semuanya dari field panel.

Sesudah mengubah keadaan, panggil `repaint()`: ia hanya menandai panel "kotor"; gelunglah yang melukis nanti (baris 9 Pseudokode 10.1). Baris pertama setiap `paintComponent`: `super.paintComponent(g)`.

Buku: Subbab 10.9.2–10.9.3 dan Persamaan (10.1).

Kotak yang Menggelap dan Menerang

`ShadeBox` mendengarkan peristiwa aksi *dan* tetikus; dua tombol dan klik pada kotak itu sendiri (Kode 10.8).

Kode 10.8 `ShadeBox.java` (cuplikan): melukis dari keadaan, dan `shade` dengan `clamp`.

```
    @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));
    }

    private void shade(int delta) {
        level = clamp(level + delta);
        darken.setEnabled(level > 0);
        lighten.setEnabled(level < 255);
        repaint();  // minta agar paintComponent segera dijalankan
    }
```

`actionPerformed` bertanya `e.getSource() == darken` — kesamaan referensi: *objek yang sama persis*. `shade` mengubah keadaan dan memanggil `repaint()`; ia tidak pernah menggambar.

Buku: Subbab 10.9.4 dan Kode 10.11.

Contoh Tuntas: Berapa Klik sampai Hitam?

`clamp` menghitung Persamaan (10.2), dari dalam ke luar:

$$v' = \min\left(255, \max\left(0, v + \delta\right)\right) \tag{10.2}$$

Setiap klik mengurangi 10 dan klik terakhir dijepit, sehingga dari putih ke hitam diperlukan Persamaan (10.3):

$$\left\lceil \frac{255}{10} \right\rceil = \lceil 25.5 \rceil = 26 \tag{10.3}$$

Sesudah 25 klik levelnya $255 - 250 = 5$; klik ke-26 memberi -5, dijepit menjadi 0, dan *Darken* dinonaktifkan. Piksel yang dilukis `#0a0a0a`: $0\text{a}_{16} = 10$ sesudah satu klik di dalam kotak.

```
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
```

Buku: Subbab 10.9.4 serta Persamaan (10.2) dan (10.3).

Jendela Peneduh



Gambar 10.13 Jendela penggelap setelah cukup banyak klik pada *Darken*: kotaknya hitam, dan tombol *Darken* nonaktif karena tingkatnya tidak bisa lebih rendah lagi.

Gambar 10.13: kotaknya hitam dan *Darken* nonaktif. `MouseListener` memaksakan lima metode pada kelas, empat di antaranya kosong — Subbab 10.10 menghilangkan gangguan itu.

Buku: Subbab 10.9.4 dan Gambar 10.13.

Label yang Menjelaskan



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.

Gambar 10.14: satu `setText` di batas-batasnya membuat pengguna tidak perlu bertanya-tanya mengapa sebuah tombol berhenti bekerja.

Buku: Subbab 10.9.4 dan Gambar 10.14.

Canvas Tkinter: Mode Tertahan

`create_rectangle(...)` membuat sebuah **item** yang diingat dan digambar ulang oleh Tk; ubahlah dengan `itemconfig` — tanpa `paintComponent`, tanpa `repaint()` (Kode 10.9).

Kode 10.9 `shade_box.py` (cuplikan): item canvas, klik yang diikat pada item itu sendiri, dan `shade` yang mengubah item.

```
        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)
```

WAWASAN KUNCI. Swing (mode langsung): *keadaan di field, piksel di* `paintComponent`, `repaint()` *untuk menghubungkan keduanya*. Canvas Tk (mode tertahan): *buat item sekali, lalu ubah item-itemnya*.

Buku: Subbab 10.9.5 dan Kode 10.12.

Kelas Dalam, Kelas Anonim, Lambda

DEFINISI. Kelas dalam dideklarasikan di dalam kelas lain tanpa `static`. Setiap objek kelas dalam dibuat di dalam sebuah objek luar dan menyimpan referensi tersembunyi kepadanya, sehingga ia dapat memakai field dan metode objek luar — bahkan yang `private`; referensi itu ditulis `Outer.this`.

Kelas anonim mendeklarasikan dan menginstansiasi dalam satu ekspresi: `new ActionListener() { ... }`. Memperluas sebuah **adaptor** (`MouseAdapter`) cukup meng-*override* metode yang diperlukan.

Antarmuka bermetode satu adalah **antarmuka fungsional**; **lambda** mengimplementasikannya: `darken.addActionListener(e -> shade(-10));`. **Referensi metode**: `this::resetCount`.

Buku: Subbab 10.10.1–10.10.2.

Lima Gaya Berdampingan

Kode 10.10: panel itu sendiri, kelas dalam bernama, kelas anonim, lambda, dan referensi metode — ditambah lambda yang dibuat di dalam perulangan.

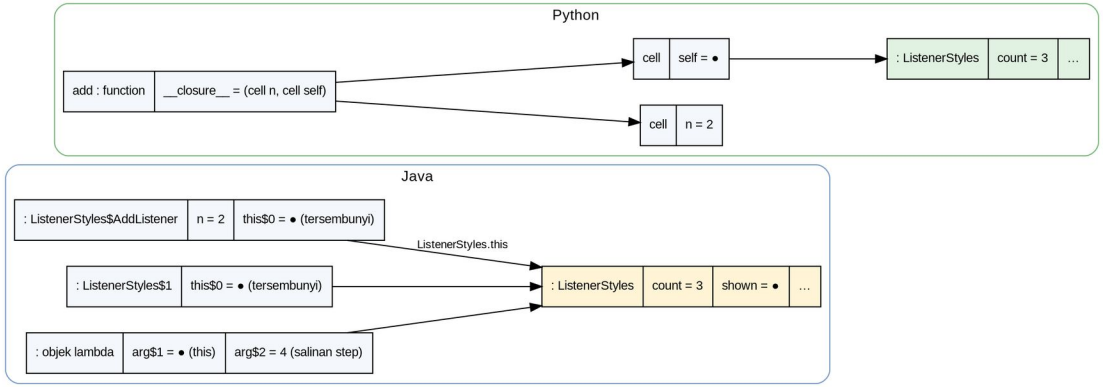
Kode 10.10 `ListenerStyles.java` (cuplikan): lima cara menulis action listener, dan lambda yang dibuat di dalam perulangan.

```
        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);
        }
```

Hitungannya: 1, 3, 6, 10, lalu +10, +20, +30 memberi 20, 40, 70; *Reset* kembali ke 0. Kelas dalam dikompilasi menjadi `ListenerStyles$AddListener.class`, yang anonim menjadi `ListenerStyles$1.class`; lambda mendapat kelas tersembunyi saat eksekusi.

Buku: Subbab 10.10.3 dan Kode 10.13.

Apa yang Diingat Listener



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.)

Gambar 10.15: objek dalam dan anonim menyimpan `this$0`; lambda menyimpan *salinan*; closure Python menyimpan *sel*.

Buku: Subbab 10.10.3 dan Gambar 10.15.

Effectively Final: Mengapa Java Menyalin

Lambda hanya boleh memakai variabel lokal yang **effectively final**: lambda bisa berjalan lama sesudah metodenya selesai, sehingga Java menyimpan *salinan* — dan melarang variabel itu berubah sesudahnya.

Perulangan yang wajar `for (int n = 10; n <= 30; n += 10)` tidak lolos kompilasi:

```
LoopCapture.java:9: error: local variables referenced from a lambda expression must be final or
effectively final
            b.addActionListener(e -> count += n);
                                              ^
1 error
```

`for (int n : new int[] {10, 20, 30})` tidak masalah: `n` yang *baru* pada setiap putaran. Field bebas: lambda mencapai `count` melalui referensi panel yang ditangkapnya.

Buku: Subbab 10.10.4.

Python: Lima Callback dan Satu Jebakan

Bound method, closure (`make_adder`), lambda, `functools.partial`, objek yang dapat dipanggil dengan `__call__` (Kode 10.11).

Kode 10.11 `listener_styles.py` (cuplikan): jebakan *late binding* dalam perulangan, dan obatnya dengan argumen bawaan.

```
        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)))
```

Ketiga tombol “late” semuanya menambah 30: closure menyimpan *variabel* `n`, bukan nilainya, dan membacanya ketika **dipanggil** — ***late binding***. `lambda n=n:` mengevaluasi nilai bawaannya *sekarang*.

```
late  +10 -> count = 30
late  +20 -> count = 60
late  +30 -> count = 90
fixed +10 -> count = 10
fixed +20 -> count = 30
fixed +30 -> count = 60
```

Buku: Subbab 10.10.5 dan Kode 10.14.

Menggambar dengan Tetikus

Satu `MouseAdapter` anonim menjadi listener tetikus *sekaligus* listener gerak tetikus; gambarnya tinggal di `strokes`, sebuah `List<List<Point>>` (Kode 10.12).

Kode 10.12 `Doodle.java` (cuplikan): `MouseAdapter` anonim yang memulai goresan, memperpanjangnya, dan melaporkan posisi penunjuk.

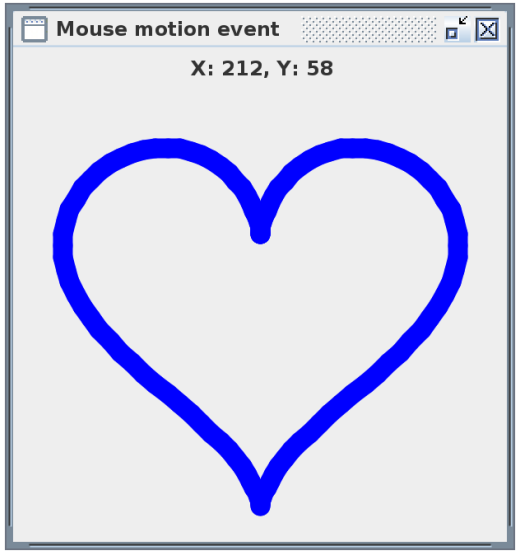
```
        MouseAdapter pen = new MouseAdapter() {  // sebuah kelas dalam anonim
            @Override
            public void mousePressed(MouseEvent e) {
                strokes.add(new ArrayList<>(List.of(e.getPoint())));  // goresan baru
                repaint();
            }

            @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
```

Buku: Subbab 10.10.6 dan Kode 10.15.

Mengapa Titik-titiknya Disimpan?



Gambar 10.16 Gambar hati yang dibuat dengan menyeret tetikus, dengan koordinat penunjuk ditampilkan di label.

Gambar 10.16 dapat digambar dengan `getGraphics()` di dalam `mouseDragged` — sampai jendela diminimalkan atau tertutup: lalu gambarnya hilang. Hanya yang dilukis `paintComponent` dari field yang bertahan. Di Tkinter setiap segmen adalah item garis, dan Tk menyimpannya (Kode 10.16 di buku).

Buku: Subbab 10.10.6 dan Gambar 10.16.

Aturan Thread

> **DEFINISI. Aturan thread.** Komponen Swing dan widget Tkinter hanya boleh dibuat, dibaca, dan diubah di thread GUI — *event dispatch thread* (EDT) di Swing, thread yang menjalankan `mainloop()` di Tkinter. Sebaliknya, thread GUI tidak boleh disibukkan lama.

Semua urusan GUI di thread GUI: listener sudah berjalan di sana; kode lain memakai `SwingUtilities.invokeLater(r)` atau `invokeAndWait(r)` — itulah sebabnya `main` membangun jendela dengan `invokeLater`.

Tidak ada yang lambat di thread GUI: sekitar 0,1 detik terasa seketika; sesudah sekitar 1 detik alur pikiran pengguna terputus.

Buku: Subbab 10.11.1.

Pekerjaan Lambat, Salah dan Benar

Sebulan pesanan, 20 ms per hari, sekitar 600 ms seluruhnya; total yang harus ditampilkan kedua tombol adalah Persamaan (10.4):

$$\sum_{d=1}^{30} \left(100{,}000 + 5{,}000 \left(d \bmod 7\right)\right) = 30 \times 100{,}000 + 5{,}000 \times 87 = 3{,}435{,}000 \tag{10.4}$$

Sisa $d \bmod 7$ untuk $d = 1, \ldots, 30$: empat minggu masing-masing 21, ditambah $1+2$ — jadi $4 \times 21 + 3 = 87$.

Tombol yang benar menyerahkan penghitungan kepada `SwingWorker` (Kode 10.13):

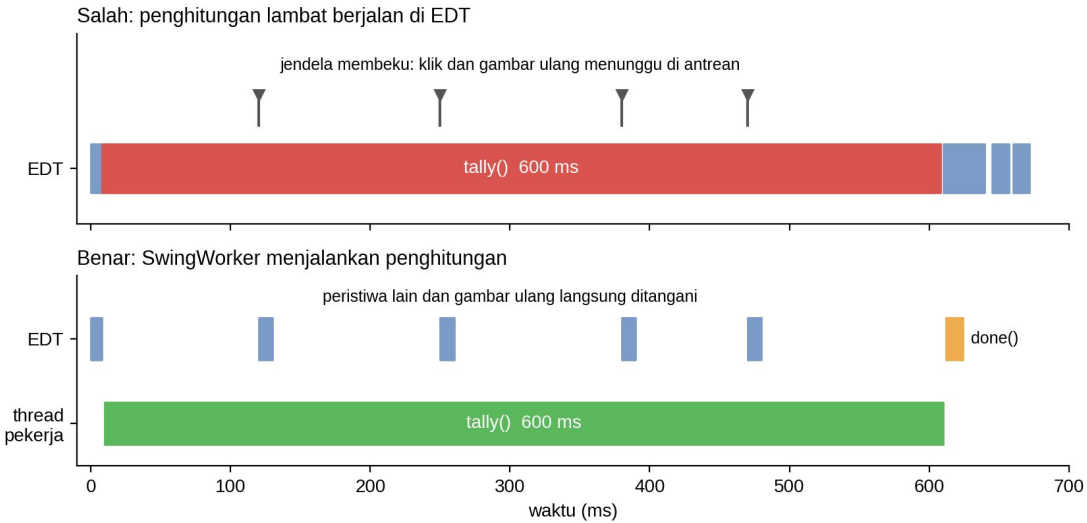
Kode 10.13 `SlowWork.java` (cuplikan): `SwingWorker` — `doInBackground` di thread pekerja, `done` kembali di EDT.

```
        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();
            }
        }.execute();
```

Buku: Subbab 10.11.2, Persamaan (10.4), dan Kode 10.17.

Aturan Thread 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.

Gambar 10.17: log menyebut nama thread-nya — `AWT-EventQueue-0` tertahan 600 ms untuk tombol yang salah; `SwingWorker-pool-1-thread-1` untuk yang benar, dan `done()` kembali di EDT.

Buku: Subbab 10.11.2 dan Gambar 10.17.

Resep yang Sama di Tkinter

Mulai thread untuk pekerjaannya; biarkan ia menaruh hasilnya di antrean yang aman-thread; biarkan thread GUI memeriksa antrean itu dengan pewaktu (Kode 10.14).

Kode 10.14 `slow_work.py` (cuplikan): thread pekerja *daemon*, sebuah `queue.Queue`, dan `after` yang memeriksa antrean di thread GUI.

```
    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())

    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)
```

`work` tidak menyentuh widget apa pun. `poll` berjalan di `MainThread` setiap 50 ms sampai hasilnya tiba.

Buku: Subbab 10.11.2 dan Kode 10.18.

Profitina Laundry: Jendela Pencatatan Pesanan

Formulir: nama, telepon, layanan (Wash & Fold, Iron Only, Wash, Dry & Iron), berat, *Express*.

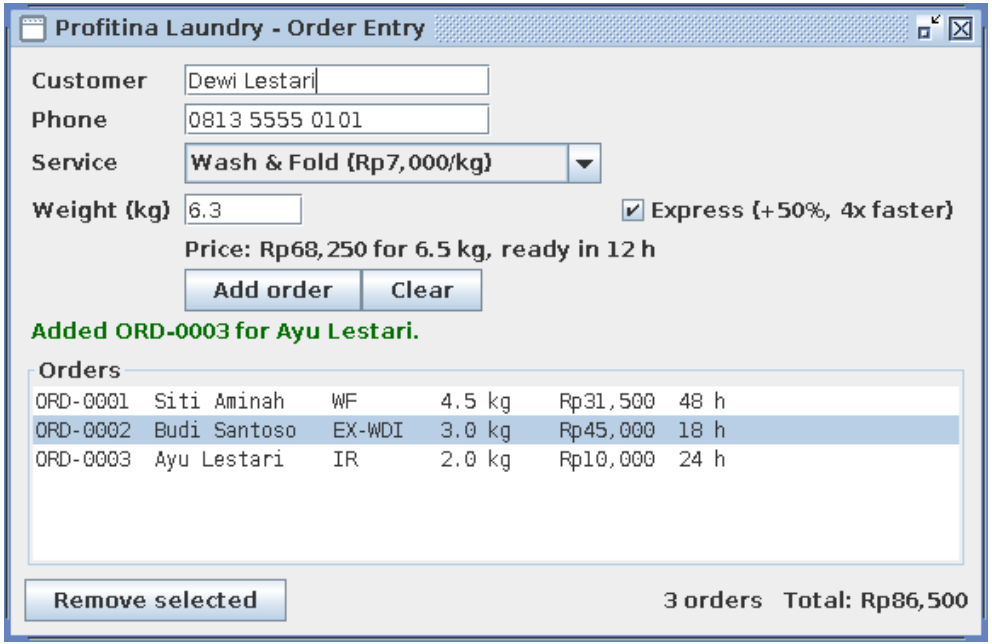
Harga langsung begitu berat, layanan, atau kilat berubah — sebelum apa pun ditambahkan.

Validasi: nama paling banyak 40 karakter; telepon 10–13 digit diawali 0; berat di atas 0 dan paling banyak 50 kg, dan `4,2` diterima sama seperti `4.2`. Galat ditampilkan merah, kursor di field yang harus diperbaiki.

List, hapus, dan total: ORD-0001, ORD-0002, …; *Remove selected* aktif hanya selagi sebuah baris dipilih; kaki jendela menampilkan jumlah dan total.

Buku: Subbab 10.12.

Jendela Swing

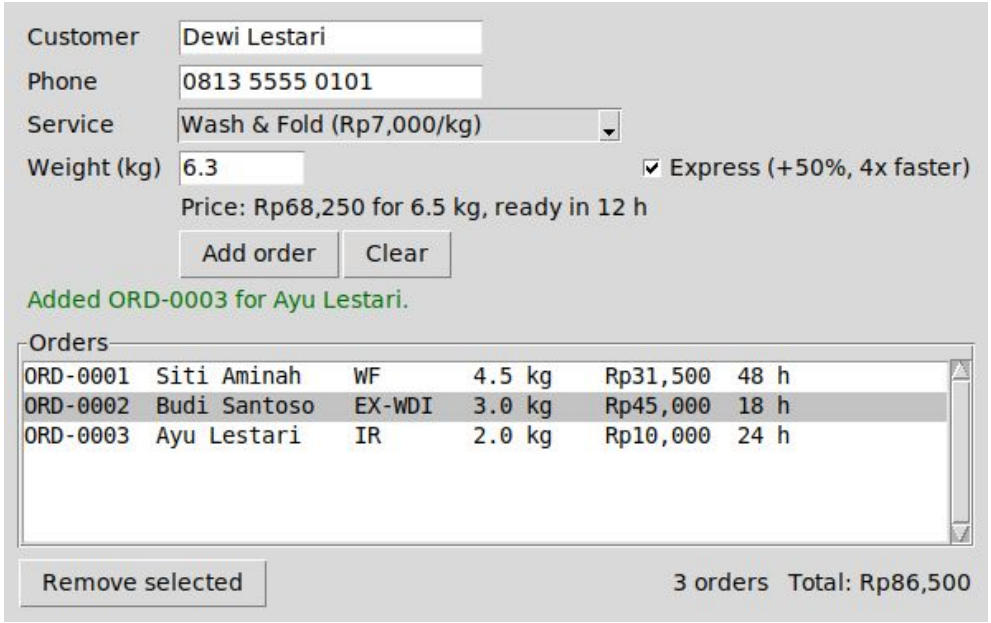


Gambar 10.18 Jendela pencatatan pesanan Swing dengan tiga pesanan, pesanan keempat sedang diketik (harga langsungnya sudah menunjukkan Rp68,250), dan baris kedua terpilih.

Gambar 10.18: tiga pesanan, pesanan keempat sedang diketik (harga langsung sudah menunjukkan Rp68.250), baris kedua dipilih.

Buku: Subbab 10.12.1 dan Gambar 10.18.

Jendela Tkinter



Gambar 10.19 Jendela pencatatan pesanan Tkinter dalam keadaan yang sama. (Tangkapan layar memperlihatkan isi jendela tanpa bilah judul yang ditambahkan sistem operasi.)

Gambar 10.19: keadaan yang sama — dan uji mandiri kedua versi mencetak baris-baris yang *identik*.

Buku: Subbab 10.12.1 dan Gambar 10.19.

Contoh Tuntas: Harga, secara Cermat

Berat yang ditagihkan Persamaan (10.5), harga dan waktu selesai Persamaan (10.6), aturan kilat Persamaan (10.7):

$$\hat{w} = \frac{\lceil 2 \max(w, w_{\min}) \rceil}{2} \tag{10.5}$$

$$P = r \cdot \hat{w}, H = h \tag{10.6}$$

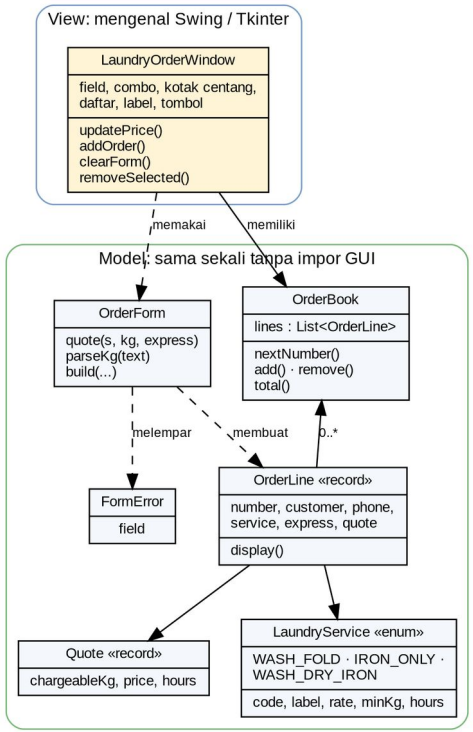
$$P_{\text{ex}} = \lfloor \frac{3P}{2} \rfloor, H_{\text{ex}} = \max\left(6, \lfloor \frac{h}{4} \rfloor\right) \tag{10.7}$$

Siti, 4,2 kg Wash & Fold: $\lceil 8.4 \rceil = 9$, $\hat{w} = 4.5$, $P = 31{,}500$, 48 jam. **Budi**, 2,6 kg WDI kilat: $\hat{w} = 3.0$, $P = 30{,}000$, $P_{\text{ex}} = 45{,}000$, $H_{\text{ex}} = \max(6, 18) = 18$ jam. **Ayu**, 1,3 kg setrika: $5000 \times 2 = 10{,}000$. Total $86{,}500$.

Dewi, 6,3 kg Wash & Fold kilat: $\lceil 12.6 \rceil = 13$, $\hat{w} = 6.5$, $P = 45{,}500$, $P_{\text{ex}} = 68{,}250$, 12 jam — harga langsung pada Gambar 10.18.

Buku: Subbab 10.12.2 dan Persamaan (10.5)–(10.7).

Model yang Tidak Tahu Apa-apa tentang Jendela



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.

Gambar 10.20: `LaundryService`, `Quote`, `OrderLine`, `FormError`, `OrderForm`, `OrderBook` tidak mengimpor apa pun dari Swing atau Tkinter; view bertanya kepada model untuk setiap keputusan.

Buku: Subbab 10.12.3 dan Gambar 10.20.

Aturan di dalam Model

`OrderForm.quote` adalah Persamaan (10.5)–Persamaan (10.7) dalam kode; 10^{-9} menjaga dari pembulatan biner (Kode 10.15).

Kode 10.15 `LaundryOrderWindow.java` (cuplikan): `OrderForm.quote`, aturan harga dalam aritmetika bilangan bulat.

```
    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);
    }
```

`parseKg` mengubah `4,2` menjadi 4.2 atau melempar `FormError` yang menyebut field yang harus diperbaiki.

Buku: Subbab 10.12.2–10.12.3 dan Kode 10.19.

Menghubungkan Peristiwa

Seluruh pengabelan di satu metode, satu baris per reaksi (Kode 10.16):

Kode 10.16 `LaundryOrderWindow.java` (cuplikan): `wireEvents`, satu baris per reaksi.

```
    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());
    }
```

`DocumentListener` mendengar setiap ketukan tombol (action listener baru terpicu pada Enter). Tkinter: `weight_var.trace_add("write", ...)` (Kode 10.20 di buku).

Buku: Subbab 10.12.4 dan 10.12.6, Kode 10.19.

Uji Mandiri Seluruh Jendela

```
   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
```

Kebutuhan diperiksa satu per satu: telepon yang salah dan tiga berat yang salah ditolak; tiga pesanan, total Rp86.500; sesudah ORD-0002 dihapus, $86{,}500 - 45{,}000 = 41{,}500$ dan *Remove selected* kembali nonaktif.

Buku: Subbab 10.12.5.

Profitina dalam Praktik

Produk Profitina adalah aplikasi web, tetapi peramban menjalankan model yang persis sama: satu thread GUI, `addEventListener`, sebuah gelung peristiwa; listener yang lambat membekukan halaman.

Posting optimistis adalah aturan thread yang sedang bekerja: kotak tulis langsung kosong, permintaan berjalan di latar belakang, dan bila gagal teksnya dipulihkan — bentuk `SwingWorker`.

Umpan langsung datang lewat Server-Sent Events: halaman mendaftarkan listener lalu menunggu — Pseudokode 10.1 dengan server sebagai satu lagi sumber peristiwa.

Dasbor adalah view atas sebuah model; aturannya tinggal di server, dapat diuji tanpa peramban. Jendela memberi saran; orang di loketlah yang menekan *Add order*.

Buku: Subbab 10.13.

Kesalahan yang Sering Terjadi

Pekerjaan lambat di listener; menyentuh komponen dari thread lain; lupa `EXIT_ON_CLOSE`; `setVisible(true)` sebelum komponen ditambahkan.

Menggambar di luar `paintComponent`; lupa `super.paintComponent(g)`; memanggil `paintComponent` sendiri alih-alih `repaint()`.

Mengira listener berjalan di tempat ia didaftarkan; jebakan *late binding*; `command=self.add_order()` — memanggil callback alih-alih memberikannya.

Mencampur `pack` dan `grid` pada satu induk; posisi mutlak; aturan di dalam listener alih-alih di dalam model.

Buku: Subbab 10.14.

Ringkasan: Lima Hal untuk Diingat

1. Program GUI berbasis peristiwa: bangun jendela, daftarkan penangan, serahkan kendali kepada gelung peristiwa — pembalikan kendali.
2. Jendela adalah pohon komponen; layout manager dan `pack`/`grid` menempatkannya menurut aturan; sarangkan panel untuk jendela yang rumit.
3. Lukislah dari keadaan: `paintComponent` + `repaint()` di Swing, item canvas di Tk.
4. Kelas dalam dan anonim menyimpan objek luarnya; lambda menyalin variabel lokal yang effectively final; closure Python mengikat terlambat — perbaiki perulangan dengan `lambda n=n:` atau `partial`.
5. Aturan thread: sentuh GUI hanya di thread-nya, jangan pernah menahannya; simpan aturan di model yang bebas toolkit, lengkap dengan uji mandiri.

Buku: Subbab 10.15.

Pertemuan Berikutnya

Bab 11 — Pola Model–View–Controller. Pemisahan model dan view hari ini menjadi sebuah pola dengan tiga peran.

Pola Observer di bawah MVC, penghitung Up/Down yang dibangun ulang, papan tombol telepon prediktif, dan meja depan Profitina Laundry sebagai aplikasi MVC yang lengkap.

Sebelum pertemuan: kerjakan Soal Latihan 10.1, 10.2, 10.8, dan 10.9, lalu bacalah Subbab 11.1 sampai 11.4.

Buku: Subbab 10.16 dan Bab 11.

Bab 1

Kuliah 11 — Pola Model–View–Controller

Pemrograman Berorientasi Objek (EF234302)

Buku: Irfan Subakti, *Pemrograman Berorientasi Objek dengan Java dan Python*

Dosen: Irfan Subakti — yifana@gmail.com

Buku: Bab 11, Pola Model–View–Controller. Setiap halaman handout ini diakhiri dengan penunjuk ke buku, yang menjadi rujukan utama; handout ini adalah bab tersebut dalam bentuk ringkas.

Pertemuan Hari Ini

1. Meja depan yang tahu terlalu banyak, dan tiga peran: model, view, controller — serta siapa boleh mengenal siapa (Subbab 11.2–11.3).

2. Program MVC terkecil, dan pola Observer di bawah MVC dalam Java dan Python (Subbab 11.4–11.5).

3. Dua program MVC berjendela: pencacah Up/Down dan papan tombol ponsel prediktif (Subbab 11.6–11.7).

4. Menguji model tanpa GUI (Subbab 11.8 dan 11.10).

5. Meja depan Profitina Laundry sebagai aplikasi MVC yang lengkap; MVP dan MVVM (Subbab 11.9–11.13).

Buku: Bab 11, Subbab 11.1 sampai 11.14.

Di Mana Kita Berhenti

Bab 10 memberi program kita jendela: frame dan panel, layout manager, dan gagasan pokoknya — sesudah jendela dibangun, program *menunggu*; toolkit memanggil listener kita.

Program-programnya kecil, masing-masing satu kelas: kelas itu membangun jendela, dan listener-nya mengerjakan apa pun kegunaan tombolnya. Cukup untuk dua puluh baris; runtuh pada dua ratus.

Di bawah pola hari ini ada pola yang sudah kita pakai tanpa namanya: setiap `addActionListener` mendaftarkan seorang **pengamat** pada sebuah tombol.

> **WAWASAN KUNCI.** Jendela adalah tempat yang buruk untuk menyimpan kebenaran. Taruh data dan aturan di objek yang tidak tahu apa-apa tentang jendela (model), biarkan sebanyak apa pun jendela menggambarnya (view), dan biarkan objek ketiga menerjemahkan klik menjadi permintaan kepada model (controller).

Buku: Subbab 11.1.

Meja Depan yang Tahu Terlalu Banyak

Satu kelas, `FrontDeskWindow extends JFrame`; listener *Add order* membaca field, memeriksa berat, menghitung harga, menyimpan baris tabel, dan menggambar ulang total. Lalu, dalam setahun:

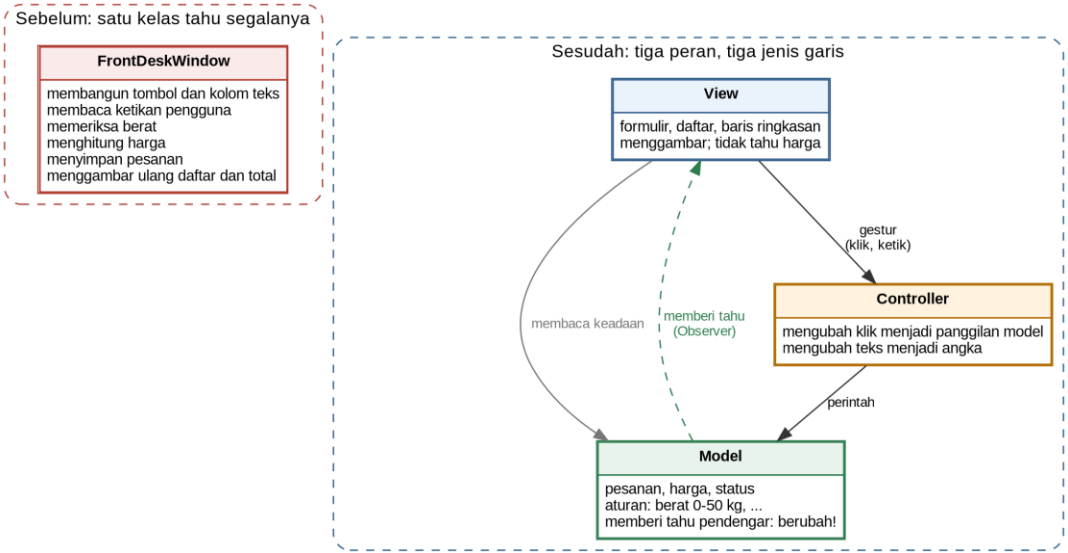
Layar kedua untuk pesanan yang siap — padahal pesanan hanya hidup sebagai larik `Object` di tabel jendela.

Aturan harga baru (berat minimum) — tersembunyi di dalam listener tombol; tanda kurung yang salah letak ditemukan oleh pelanggan.

Pengujian ("apakah 4,2 kg Wash & Fold masih Rp31.500?") — mustahil tanpa layar. **Versi web** — tidak ada yang dapat dipakai ulang.

Buku: Subbab 11.2.

Sebelum dan Sesudah MVC



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.

Gambar 11.1: tugas-tugas yang berubah karena alasan berbeda — tampilan, aturan bisnis, penyimpanan — dilas menjadi satu listener. Sesudah MVC, setiap kisah dari keempatnya hanya menyentuh satu bagian.

Buku: Subbab 11.2 dan Gambar 11.1.

Tiga Peran

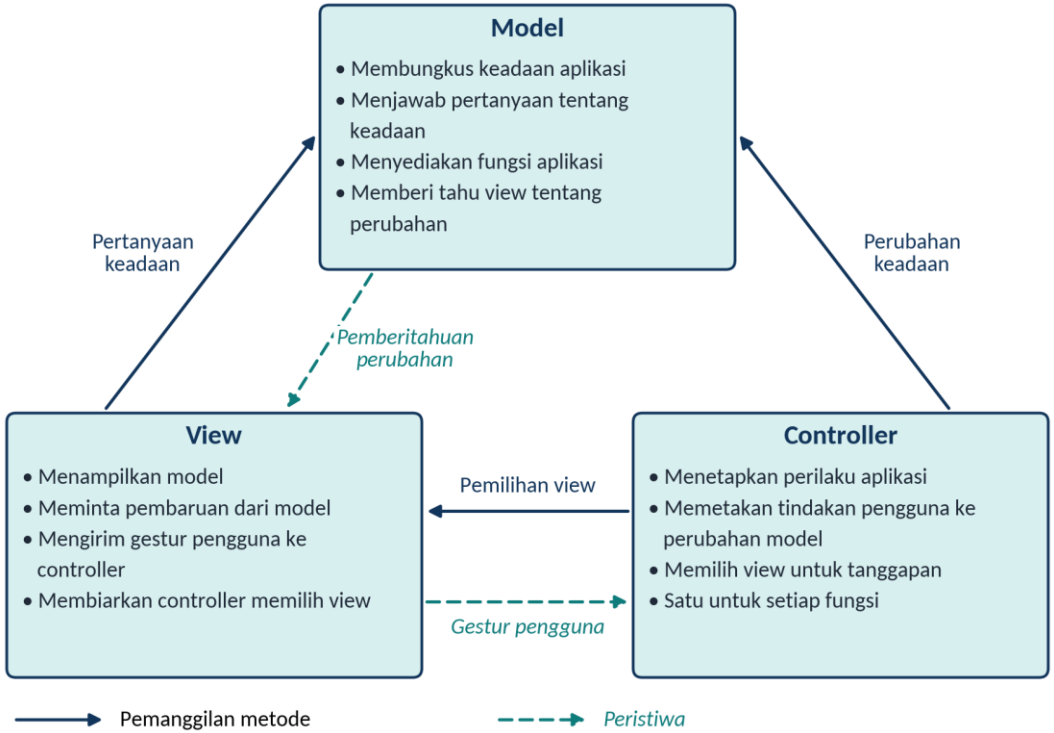
> **DEFINISI. Model** mewakili data aplikasi dan aturan yang mengatur akses serta pembaruan data itu. **View** menyajikan isi model: ia menentukan cara data ditampilkan, dan ketika data berubah, ia memperbarui tampilannya. **Controller** menerjemahkan interaksi pengguna dengan view menjadi tindakan yang dilakukan model.

Definisi model tidak menyebut piksel atau tombol; definisi view tidak menyebut apa *arti* datanya.

Controller adalah penerjemah antara bahasa gerakan ("*Add order* diklik ketika field berat berisi `4,2`") dan bahasa bisnis ("tambahkan 4,2 kg Wash & Fold untuk Sari").

Buku: Subbab 11.3.

Diagram Klasik

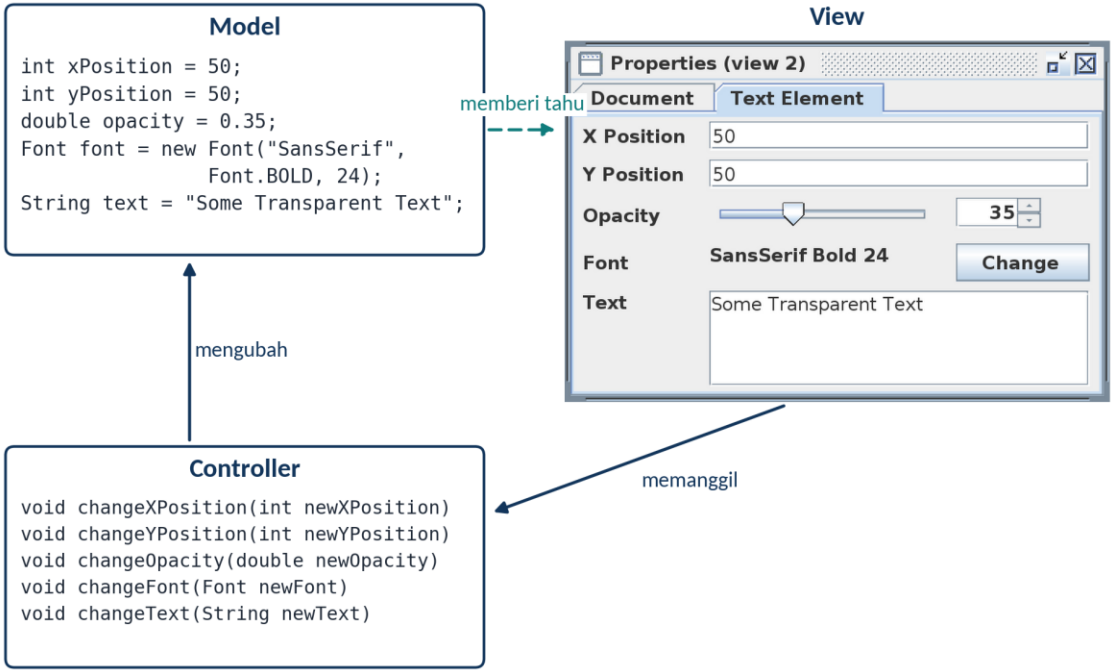


Gambar 11.2 Diagram MVC klasik. Model membungkus keadaan aplikasi, menjawab pertanyaan tentang keadaan itu, menyediakan fungsi aplikasi, dan memberi tahu view tentang perubahan; view menampilkan model dan mengirim gestur pengguna ke controller; controller memetakan tindakan pengguna menjadi perubahan model dan memilih view untuk tanggapannya.

Gambar 11.2: panah padat adalah pemanggilan metode, panah putus-putus adalah peristiwa. Model tidak pernah memanggil view dengan namanya — ia hanya mengumumkan “aku telah berubah”.

Buku: Subbab 11.3 dan Gambar 11.2.

Peran yang Dapat Ditunjuk

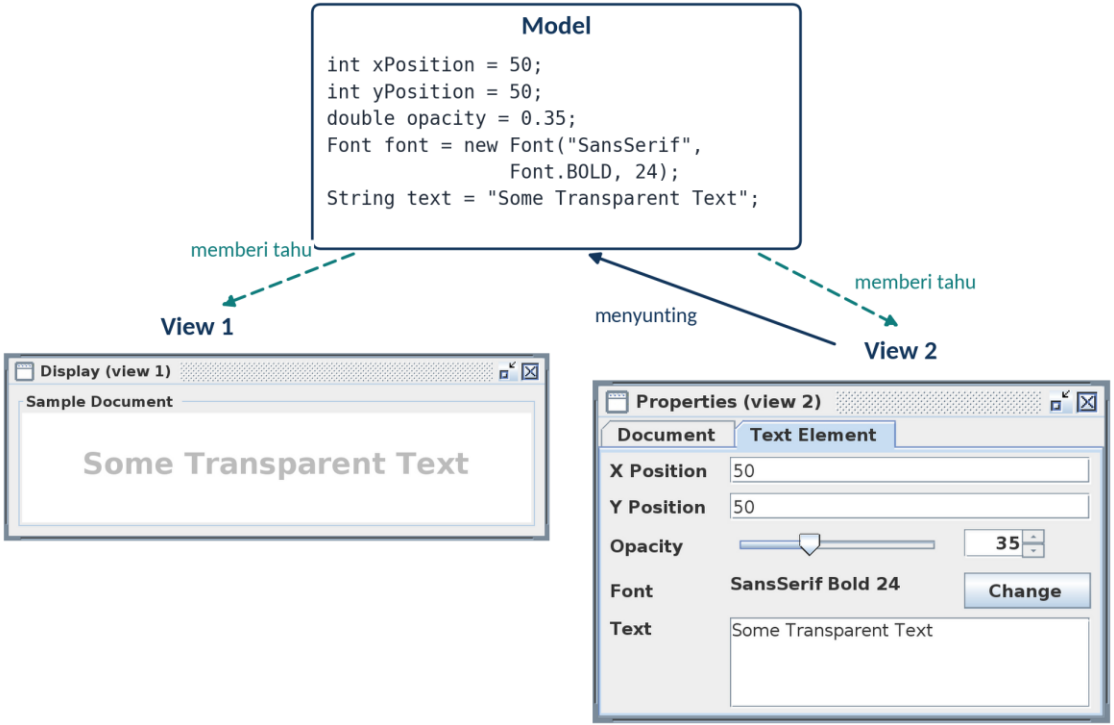


Gambar 11.3 Model, view, dan controller sebuah penyunting dokumen kecil. Modelnya lima nilai biasa; view-nya jendela properti yang menampilkan nilai-nilai itu; controller-nya menyediakan satu metode untuk setiap properti, yang dipanggil view ketika pengguna menyunting sebuah kolom.

Gambar 11.3: geser penggeser opasitas ke 50, dan *spinner* ikut berubah — *karena modelnya berubah*, bukan karena penggeser menyuruhnya. Dua widget yang menampilkan satu nilai tidak mungkin berselisih.

Buku: Subbab 11.3.1 dan Gambar 11.3.

Satu Model, Banyak View



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.

Gambar 11.4: jendela *Display* dan *Properties* tidak saling mengenal; masing-masing hanya mengenal model.

Buku: Subbab 11.3.1 dan Gambar 11.4.

View Adalah Fungsi dari Model

Pada saat t model berada dalam keadaan S_t; idealnya view adalah fungsi penyaji R dari keadaan itu, Persamaan (11.1):

$$V_t = R(S_t) \tag{11.1}$$

Dua view adalah $R_1(S_t)$ dan $R_2(S_t)$: keduanya tidak mungkin saling bertentangan.

Pengujian model tidak memerlukan view: ia memeriksa S_t secara langsung.

Field teks yang memegang nilai yang tidak ada di tempat lain melanggar Persamaan (11.1) — di situlah dua layar suatu hari akan berselisih.

Buku: Subbab 11.3.1 dan Persamaan (11.1).

Siapa Boleh Mengenal Siapa

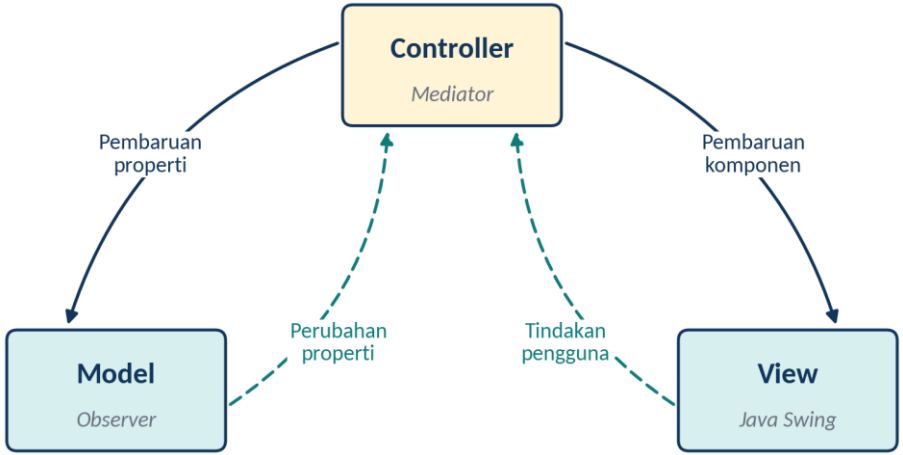
Tabel 11.1 Tanggung jawab dan ketergantungan yang diizinkan bagi ketiga peran.

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

Tabel 11.1, sel terpentingnya: **model tidak boleh mengenal toolkit GUI** — tanpa `javax.swing`, tanpa `java.awt`, tanpa `tkinter`. Mudah diperiksa, dan menjamin keterujian, pemakaian ulang, serta kebebasan dari urutan penggambaran ulang.

Buku: Subbab 11.3.2 dan Tabel 11.1.

Controller sebagai Mediator



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.

Gambar 11.5: model dan view tidak pernah bertemu. Papan tombol membiarkan view-nya mengamati model; meja laundry menyalurkan segalanya melalui controller perantara.

Buku: Subbab 11.3.2 dan Gambar 11.5.

Asal-usul Gagasan Ini

1979: Trygve Reenskaug di Xerox PARC, mengerjakan Smalltalk — satu pengguna, informasi yang sama dilihat dan diubah dengan beberapa cara sekaligus.

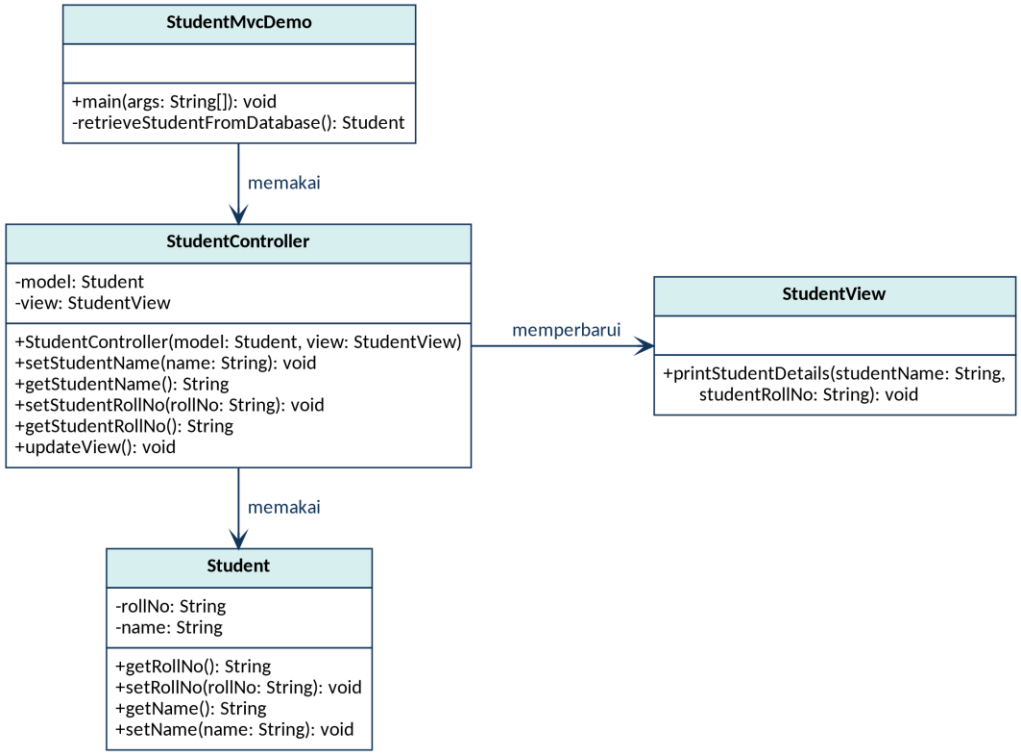
Dibangun ke dalam pustaka kelas Smalltalk-80; pola **Observer** yang menghubungkan model dengan view dikatalogkan dalam *Design Patterns* (1994).

Swing sendiri bercita rasa MVC: setiap `JTable` punya `TableModel`, setiap `JList` sebuah `ListModel`, setiap `JButton` sebuah `ButtonModel`.

Controller masa kini menerima "tombol *Add order* diklik", bukan "tetikus ditekan di piksel (412, 37)": toolkit yang mengurus masukan mentah.

Buku: Subbab 11.3.3.

Program MVC Terkecil



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).

Gambar 11.6: `Student` tidak punya panah ke mana pun — model tidak mengenal siapa-siapa. Sama sekali tanpa GUI: MVC adalah soal *tanggung jawab*, bukan soal Swing.

Buku: Subbab 11.4 dan Gambar 11.6.

Controller Mengenal Keduanya

Kode 11.1: satu-satunya kelas yang memegang model *dan* view, dalam field `final`.

Kode 11.1 `StudentMvcDemo.java` (cuplikan): `StudentController`, dibangun dari sebuah model dan sebuah view.

```
// 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);
    }
```

`updateView()` meminta nilai-nilai model lalu menyerahkannya kepada view.

Buku: Subbab 11.4 dan Kode 11.1.

Empat Langkah, dan View yang Basi

Setiap `main` MVC: **bangun model, bangun view, bangun controller dengan keduanya, biarkan controller mengemudi** (**Kode 11.2**).

Kode 11.2 `StudentMvcDemo.java` (cuplikan): `main`, keempat langkah itu.

```
    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();
    }
```

Sesudah `setStudentName("John")` model berkata *John* sementara layar masih berkata *Robert* — sampai seseorang ingat memanggil `updateView()`. Dalam program sungguhan, seseorang akan lupa.

Buku: Subbab 11.4 dan Kode 11.1.

Pola Observer

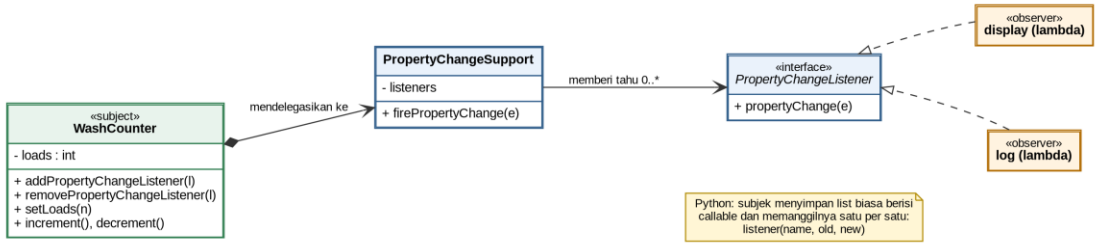
> **DEFINISI.** Dalam pola **Observer**, sebuah **subjek** menyimpan daftar **pengamat**. Pengamat mendaftar dengan memanggil metode seperti `addListener`, dan boleh berhenti berlangganan. Setiap kali keadaan subjek berubah, ia **memberi tahu** setiap pengamat terdaftar dengan memanggil satu metode yang disepakati pada masing-masing. Subjek hanya tahu bahwa pengamatnya memiliki metode itu.

`JButton` adalah subjek bagi `ActionListener`; `StringVar` Tkinter adalah subjek bagi callback `trace_add`; sebuah buletin adalah subjek bagi para pelanggannya.

Subjek: simpan daftar, sediakan *tambah* dan *hapus*, panggil semua yang terdaftar ketika berubah. Pengamat: miliki metodenya.

Buku: Subbab 11.5.

Pola Observer di Java



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*.

Gambar 11.7: `WashCounter` mendelegasikan daftar listener-nya kepada `java.beans.PropertyChangeSupport`; pengamatnya dua lambda.

Buku: Subbab 11.5 dan Gambar 11.7.

Dari Observable ke PropertyChangeSupport

`java.util.Observable` **usang** sejak Java 9: sebuah *kelas* (merampas pewarisan tunggal Anda), argumen `Object` polos, tanpa nama properti.

`PropertyChangeSupport` dipakai lewat komposisi, mengirim nama, nilai lama, dan nilai baru — dan **tidak menembakkan peristiwa bila tidak ada yang berubah** (**Kode 11.3**).

Kode 11.3 `ObserverDemo.java` (cuplikan): `setLoads` memvalidasi, menyimpan, dan menembakkan; `increment` dan `decrement` melewatinya.

```
    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);
    }
```

Buku: Subbab 11.5.1 dan Kode 11.3.

Contoh Tuntas: Membaca Pemberitahuannya

```
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
```

Setiap `increment` sampai ke `display` lalu `log`, menurut urutan pendaftaran; pencacahnya sendiri tidak mencetak apa pun. `setLoads(2)` tidak mencetak apa pun: nilai lama dan baru sama-sama 2. Sesudah `removePropertyChangeListener(log)`, hanya tampilan yang mendengar. `setLoads(-1)` ditolak *sebelum* apa pun berubah atau siapa pun diberi tahu.

Buku: Subbab 11.5.1 dan Kode 11.3.

Python: Pengamat dalam Selusin Baris

Fungsi adalah objek, sehingga subjek menyimpan daftar *fungsi* dan memanggil masing-masing (**Kode 11.4**).

Kode 11.4 `observer_demo.py` (cuplikan): validasi, simpan, tembakkan hanya pada perubahan sungguhan — atas salinan daftarnya.

```
    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)
```

`list(self._listeners)`: sebuah listener mungkin berhenti berlangganan saat diberi tahu, dan menghapus dari list selama gelung `for` diam-diam melompati yang berikutnya.

Buku: Subbab 11.5.2 dan Kode 11.4.

Pola Itu dalam Dua Bahasa

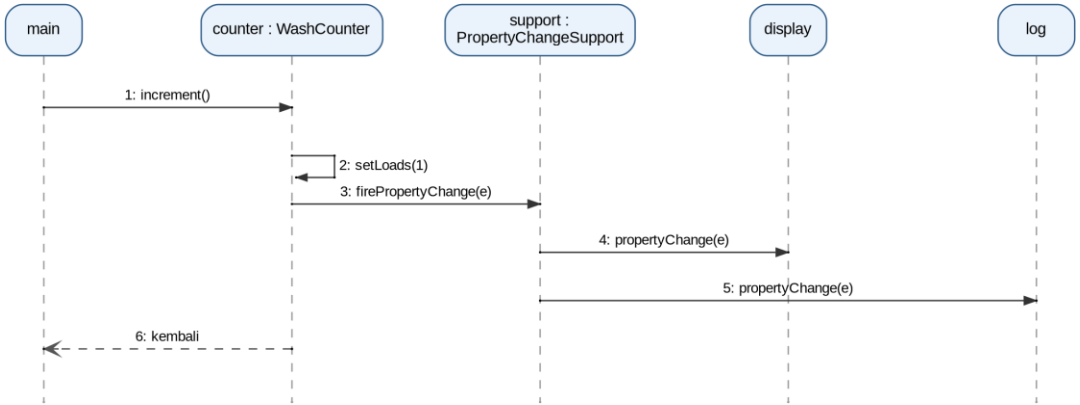
Tabel 11.2 Pola Observer dalam kedua bahasa.

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)`

Tabel 11.2: apa yang diberikan `PropertyChangeSupport` kepada Java secara cuma-cuma, ditulis Python masing-masing dalam satu baris.

Buku: Subbab 11.5.2 dan Tabel 11.2.

Satu Perubahan, Banyak Panggilan



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`.

Gambar 11.8: pemberitahuan bersifat **sinkron** — pesan 4 dan 5 terjadi *di dalam* pesan 1. Dengan k pengamat dan m perubahan: Persamaan (11.2).

$$C = m \cdot k \tag{11.2}$$

Buku: Subbab 11.5.3, Gambar 11.8, dan Persamaan (11.2).

Didorong atau Ditarik?

Dorong: setiap peristiwa membawa nilai baru (dan lama); pengamat bereaksi tanpa bertanya. Nyaman bila mereka membutuhkan persis nilai itu.

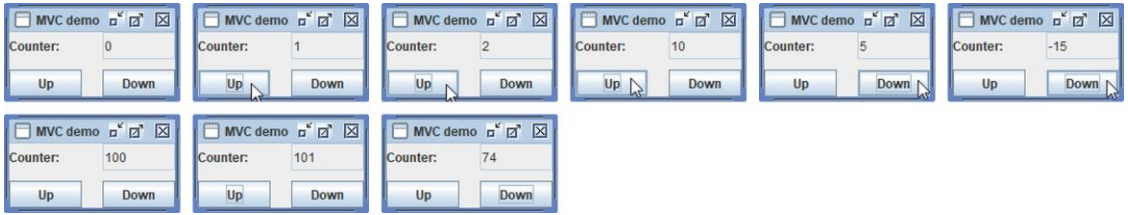
Tarik: peristiwanya hanya berkata "aku berubah"; setiap pengamat menanyakan kepada subjek apa yang ia perlukan. Lebih sederhana untuk model besar dengan view yang berbeda-beda, dan tidak mungkin mengantarkan data basi.

Model sebaiknya memberi tahu sekali per perubahan logis: tiga peristiwa untuk satu klik melipattigakan m pada Persamaan (11.2) tanpa guna.

Pencacah mendorong satu bilangan bulat; papan tombol dan meja laundry menarik.

Buku: Subbab 11.5.3–11.5.4.

Pencacah, Dibangun Ulang



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.

Gambar 11.9: kelas-kelas aslinya menaruh kerja controller di kelas bernama `View`, membiarkan `Controller` hampir kosong, dan memperluas `Observable` yang usang. Versi yang dibangun ulang mempertahankan jendelanya dan memulangkan setiap tugas ke tempatnya.

Buku: Subbab 11.6 dan Gambar 11.9.

Model Mendorong Sebuah Bilangan Bulat

`CounterModel` menyimpan pengamat `IntConsumer` dan memanggil `accept(value)` pada setiap perubahan; `increment` dan `decrement` melewati `setValue` (**Kode 11.5**).

Kode 11.5 `CounterMvc.java` (cuplikan): model pencacah — satu tempat nilai berubah, satu tempat pengamat diberi tahu.

```
    void addListener(IntConsumer listener) {
        listeners.add(listener);
    }

    void increment() {
        setValue(value + 1);
    }

    void decrement() {
        setValue(value - 1);
    }

    void setValue(int newValue) {
        value = newValue;
        for (IntConsumer listener : listeners) {
            listener.accept(value);  // beri tahu setiap pengamat
        }
    }
```

Tak ada Swing sama sekali: model tidak tahu apakah ia ditampilkan di jendela, dicetak, atau tidak ditampilkan sama sekali.

Buku: Subbab 11.6.1 dan Kode 11.5.

Kontrak View dan Controller Empat Baris

Kode 11.6 `CounterMvc.java` (cuplikan): antarmuka `CounterView` dan controller yang menyambungkan model dengan view.

```
// 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
    }
}
```

Kode 11.6: `Runnable` adalah "kode yang tidak menerima apa-apa dan tidak mengembalikan apa-apa", sehingga `model::increment` cocok. Baris terakhir menggambar **gambar pertama** — lupakan, dan jendela terbuka kosong.

Buku: Subbab 11.6.1 dan Kode 11.5.

Uji Mandiri: View yang Mencetak

Controller melihat view hanya melalui `CounterView`, sehingga `TextCounterView` yang mencetak dapat menggantikan jendela:

```
  [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
```

Terbukti: model menghitung, *Up* berarti menambah, pengamatnya terdaftar, gambar pertama tergambar. **Tidak** terbukti: bahwa jendelanya tampak benar — maka jaga view setipis mungkin agar sulit keliru.

Python (Kode 11.6 di buku): *duck typing* menggantikan antarmuka, dan `import tkinter` diletakkan *di dalam* konstruktor view sehingga uji mandiri tetap berjalan tanpa Tkinter.

Buku: Subbab 11.6 dan 11.6.3 serta Kode 11.5.

Kata, Tombol, dan Tanda Tangan

Sebelum layar sentuh: tombol 2–9 membawa huruf; **T9** menebak kata dari satu tekanan per huruf. *hello* adalah 4-3-5-5-6.

Dengan κ yang memetakan huruf ke tombolnya, **tanda tangan** (*signature*) adalah Persamaan (11.3):

$$\sigma(c_1 c_2 \ldots c_n) = \kappa(c_1)\,\kappa(c_2) \ldots \kappa(c_n) \tag{11.3}$$

$\sigma(\text{hello}) = 43556$, $\sigma(\text{wash}) = 9274$, $\sigma(\text{good}) = \sigma(\text{home}) = 4663$. σ tidak satu-satu — karena itu ada tombol bintang. Map dari tanda tangan ke kata-kata terurut membuat setiap pencarian cukup satu akses map.

Buku: Subbab 11.7.1 dan Persamaan (11.3).

Apa yang Dilakukan Setiap Tombol

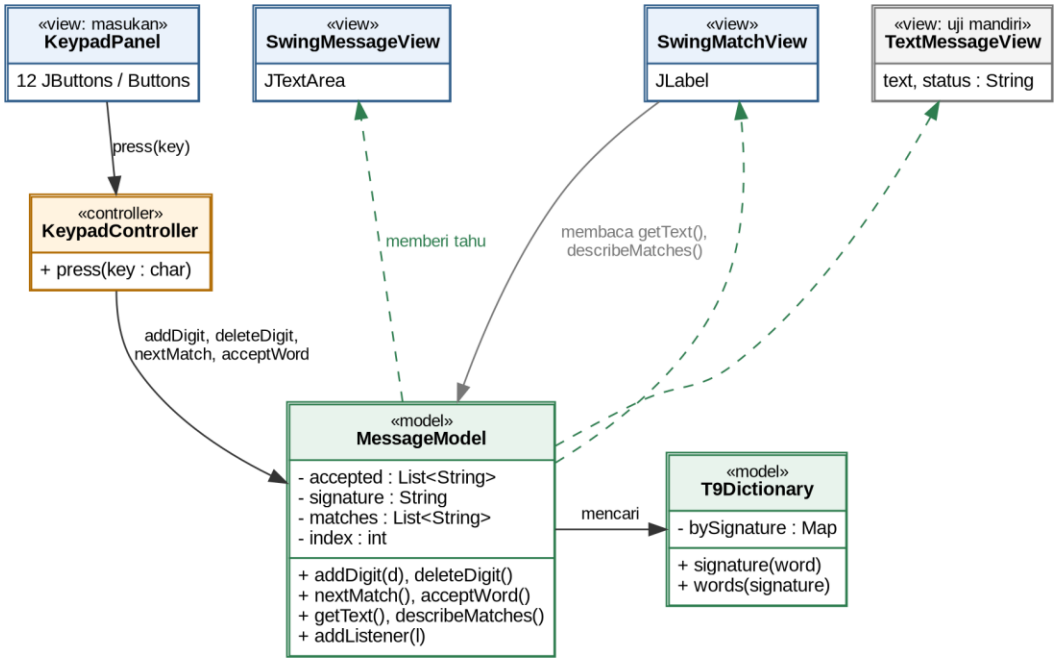
Tabel 11.3 Apa yang dilakukan setiap tombol papan tombol terhadap model pesan.

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

Tabel 11.3: bila tanda tangan tidak cocok dengan kata apa pun — `435` di tengah *hello* — model menampilkan digit-digitnya sendiri.

Buku: Subbab 11.7.1 dan Tabel 11.3.

Kelas-kelas Papan Tombol



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.

Gambar 11.10: panel melaporkan tekanan tombol kepada controller; model memakai kamus dan memberi tahu tiga view, yang menarik apa yang mereka perlukan.

Buku: Subbab 11.7.1 dan Gambar 11.10.

Model Melindungi Invariannya

Setiap operasi memeriksa apakah ada yang perlu dikerjakan, mengubah keadaan, lalu menembakkan — sekali (**Kode 11.7**).

Kode 11.7 `PredictiveKeypad.java` (cuplikan): `addDigit` menolak apa pun selain 2–9; `deleteDigit` menembakkan hanya pada perubahan sungguhan.

```
    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();
        }
    }
```

Buku: Subbab 11.7.2 dan Kode 11.7.

Controller Adalah Sebuah Tabel

Kode 11.8 adalah **Tabel 11.3** yang ditulis sebagai `switch`: bahwa tombol 0 berarti "terima kata" adalah fakta tentang *papan tombol ini*, bukan tentang pesan.

Kode 11.8 `PredictiveKeypad.java` (cuplikan): `KeypadController.press`.

```java
// Controller: menerjemahkan tekanan tombol menjadi operasi model.
class KeypadController {
    private final MessageModel model;

    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);
        }
    }
```

Label tombol berasal dari tabel `LETTERS` yang sama dengan pencarian, sehingga keduanya tidak mungkin berselisih (pada aslinya, tombol 9 bertuliskan *vwxy*).

Buku: Subbab 11.7.2 dan Kode 11.7.

Contoh Tuntas: Uji Mandiri sebagai Film

```
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)
```

`4`: hanya *i*. `43`: *he* dan *if*, menurut abjad *he* lebih dulu. `435`, `4355`: tidak ada kata, digit ditampilkan. `43556`: *hello*. `4663` memberi *gone*, dua bintang memberi *good*, lalu *home* (3 dari 6).

Buku: Subbab 11.7.4.

Papan Tombol yang Sama dalam Python

Model dan controller-nya terjemahan baris demi baris; `TkKeypadWindow` memegang kedua view dan kedua belas tombol (Kode 11.8 di buku).

Perintah setiap tombol adalah `lambda k=key: controller.press(k)`. Tanpa `k=key`, kedua belas lambda berbagi variabel gelung yang berakhir sebagai `"C"` — setiap tombol akan menghapus satu digit.

Widget `Text` diubah ke `"normal"` untuk diganti isinya lalu kembali ke `"disabled"` agar pengguna tidak dapat mengetik di dalamnya.

`python3 predictive_keypad.py --selftest` mencetak keluaran yang identik dengan Java, karakter demi karakter.

Buku: Subbab 11.7.5.

Menguji Model Tanpa GUI

Unit test memeriksa satu perilaku secara terpisah, bernama deskriptif, dan melaporkan perilaku mana yang gagal. Model papan tombol tidak mengimpor GUI, sehingga diuji seperti kelas lainnya.

Ujilah setiap *janji*, termasuk apa yang *tidak* akan dilakukan model: satu peristiwa per perubahan sungguhan, tidak ada untuk perubahan yang bukan perubahan (**Kode 11.9**).

Kode 11.9 `test_keypad_model.py` (cuplikan): satu peristiwa per perubahan sungguhan, dan tombol yang ditolak diuji dengan `assertRaises`.

```
    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")
```

> **WAWASAN KUNCI.** “Bagaimana saya menguji GUI saya?” biasanya pertanyaan yang keliru. Tanyakan “seberapa sedikit logika yang dapat saya tinggalkan di GUI saya?”

Buku: Subbab 11.8 dan Kode 11.10.

Meja Depan: Kebutuhannya

Nama, layanan, berat, *Express*; **Add order** mencatat nomor berikutnya dan menampilkan harga.

Ditolak tanpa mengubah apa pun: nama kosong, berat yang bukan bilangan, ≤ 0 atau di atas 50 kg; `2,6` berarti 2,6.

List menampilkan nomor, pelanggan, layanan, berat, kilat, harga, status; **Advance status** memajukan pesanan terpilih satu langkah: *Received*, *Washing*, *Ready*, *Picked up*.

Baris ringkasan: jumlah pesanan, berapa yang *Received*, berapa yang *Ready*, dan total pendapatan.

Buku: Subbab 11.9.

Contoh Tuntas: Harga dengan Tangan

Berat yang ditagihkan Persamaan (11.4), harga dasar Persamaan (11.5), kilat Persamaan (11.6):

$$\widehat{w} = \frac{\lceil 2\max(w, w_s) \rceil}{2} \tag{11.4}$$

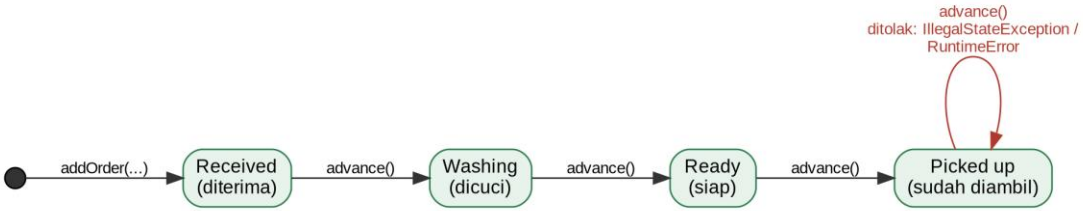
$$p = r_s \cdot \widehat{w} \tag{11.5}$$

$$p_{\text{ex}} = \lfloor \frac{3p}{2} \rfloor \tag{11.6}$$

Sari, 4,2 kg Wash & Fold: $\max(4.2,3) = 4.2$, $\lceil 8.4 \rceil = 9$, 4.5 kg, $p = 31{,}500$. **Budi**, 2,6 kg WDI kilat: 3 kg, $p = 30{,}000$, $p_{\text{ex}} = 45{,}000$. **Citra**, 1,3 kg setrika: 2 kg, 10,000. Total **Rp86.500**.

Buku: Subbab 11.9.1 dan Persamaan (11.4)–(11.6).

Riwayat Sebuah Pesanan

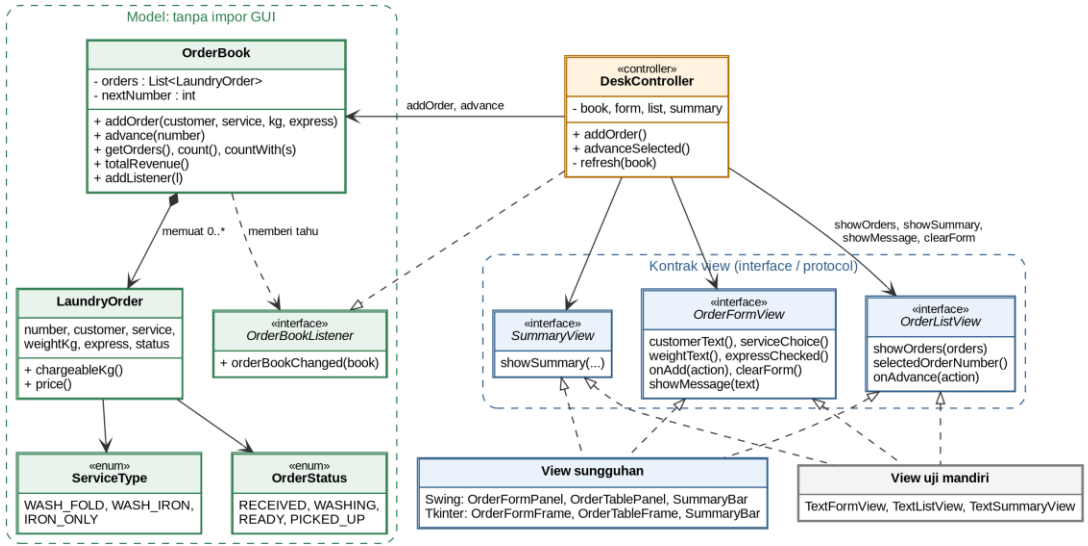


Gambar 11.11 Riwayat hidup sebuah pesanan binatu. Setiap `advance()` menggeser pesanan satu keadaan ke kanan; pesanan yang sudah diambil menolak untuk maju.

Gambar 11.11: sebuah **mesin keadaan** kecil. Setiap `advance()` bergerak satu keadaan ke kanan; *Picked up* menolak.

Buku: Subbab 11.9.1 dan Gambar 11.11.

Rancangan Meja Depan



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.

Gambar 11.12: satu kontrak view kecil untuk setiap bagian jendela — televisi di ruang tunggu hanya memerlukan list, tidak lebih.

Buku: Subbab 11.9.2 dan Gambar 11.12.

Validasi, Ubah, Beri Tahu, Kembalikan

Kode 11.10: setiap penolakan keluar *sebelum* apa pun berubah — tanpa nomor pesanan terpakai, tanpa view yang digambar ulang.

Kode 11.10 `OrderBook.java` (cuplikan): `addOrder` memvalidasi, mengubah, memberi tahu sekali, lalu kembali.

```
    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;
    }
```

`!(weightKg > 0 && weightKg <= MAX_KG)` juga menolak `NaN`, yang membuat *setiap* perbandingan bernilai salah.

Buku: Subbab 11.9.3 dan Kode 11.11.

Dua Eksepsi, Dua Makna

Kode 11.11 `OrderBook.java` (cuplikan): `advance` menolak pesanan yang sudah diambil.

```
    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();
    }
```

Kode 11.11: `IllegalArgumentException` — "Anda bertanya tentang sesuatu yang tidak ada"; `IllegalStateException` — "permintaannya wajar, tetapi tidak dalam keadaan ini".

`getOrders` mengembalikan view yang tak dapat diubah; `fireChanged` menelusuri `List.copyOf(listeners)`. Python: `orders()` mengembalikan tupel, `advance` melempar `RuntimeError`.

Buku: Subbab 11.9.3 serta Kode 11.11 dan 11.12.

Controller Mengubah Teks Menjadi Nilai

Kode 11.12 `LaundryDesk.java` (cuplikan): controller mengurai berat; model yang menilainya.

```
    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 11.12: `abc` sama sekali bukan berat, sehingga model tidak pernah melihatnya; apakah 75 kg dapat diterima adalah aturan *bisnis*, dan model yang memutuskan.

Buku: Subbab 11.9.4 dan Kode 11.13.

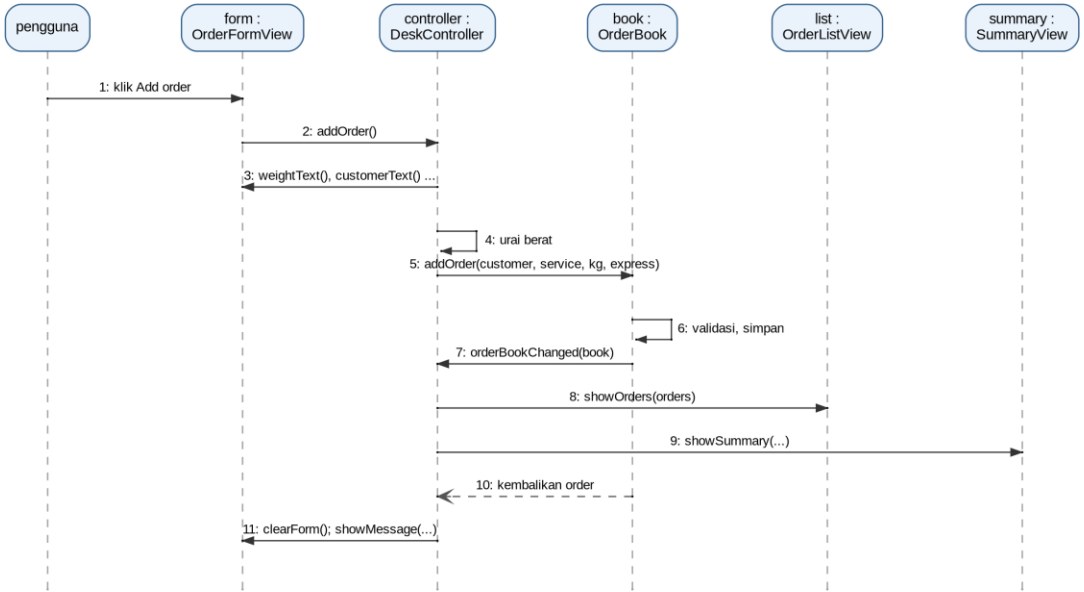
Uji Mandiri: Satu Giliran Jaga di Meja

```
> 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
```

Pesanan yang ditolak hanya mencetak pesan: model tidak berubah, tidak ada view yang digambar ulang, dan tidak ada nomor yang terpakai — pesanan Citra bernomor #3. Totalnya Rp86.500 yang kita hitung dengan tangan.

Buku: Subbab 11.9.4.

Satu Klik pada Add Order



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.

Gambar 11.13: pesan 7–9 (beri tahu, gambar ulang) terjadi *di dalam* pesan 5, sebelum model kembali; pesan 11 memperbarui formulir.

Buku: Subbab 11.9.4 dan Gambar 11.13.

View Swing dan Tkinter

Tidak ada satu `if` pun tentang bisnis di seluruh berkas view: setiap metode membangun widget atau memindahkan data antara widget dan parameter (Kode 11.14 di buku).

Pelajaran yang tak terlihat oleh uji mandiri: kedelapan komponen formulir dalam *satu* baris `FlowLayout` — wilayah `NORTH` hanya memberi tinggi satu baris, dan **Add order** terbungkus keluar dari pandangan. Hanya melihat jendelanya yang mengungkapkannya.

Python: kontrak view berupa `typing.Protocol` — *duck typing* yang dituliskan; modul Tkinter diimpor hanya ketika jendela dibuka (Kode 11.15 dan 11.16 di buku).

Buku: Subbab 11.9.5–11.9.6.

Unit Test untuk Meja Depan

`setUp` membangun tiga pesanan yang sama sebelum *setiap* pengujian; `subTest` melaporkan setiap kasus penolakan dengan namanya (**Kode 11.13**).

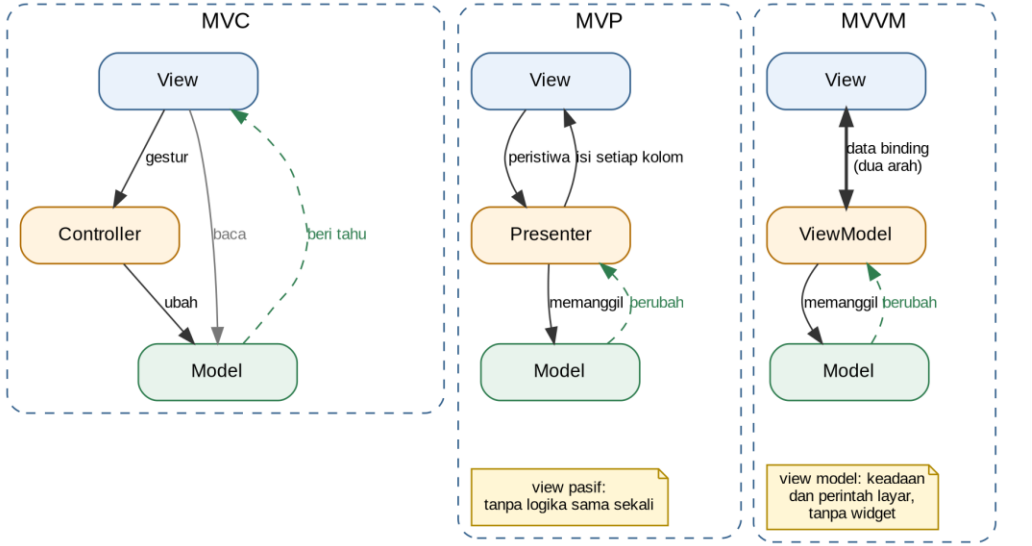
Kode 11.13 `test_order_book.py` (cuplikan): lima penolakan dalam satu gelung dengan `subTest`.

```
    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)
```

Listener-nya adalah *bound method* `self.count_event`, sehingga `remove_listener` menerima objek yang *sama nilainya* — dua lambda dengan badan yang sama adalah dua objek berbeda.

Buku: Subbab 11.10 dan Kode 11.18.

MVP dan MVVM



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.

Gambar 11.14: dalam **MVP** view pasif melapor kepada *presenter* yang mengisi setiap kolom — meja laundry kita sebenarnya MVP. Dalam **MVVM** view *diikat* pada *view model* berisi properti yang dapat diamati.

Buku: Subbab 11.11 dan Gambar 11.14.

Profitina dalam Praktik

Aplikasi web Profitina (Python, FastAPI) memperlihatkan MVC dalam skala besar. **Dasbornya** adalah banyak view atas model-model berbeda — posting, umpan balik, buku besar PROFIT, langganan — yang dirangkai oleh satu `GET /api/me/dashboard`.

Itulah Persamaan (11.1): dasbor yang menyimpan salinan hitungannya sendiri suatu hari akan berselisih dengan kanal yang diringkasnya.

Umpan komunitasnya memakai **Server-Sent Events** — subjek yang memberi tahu pendengar melintasi jaringan; posting bersifat optimistis, dibatalkan bila gagal.

Lebih dari seribu pengujian otomatis, kebanyakan atas model dan penangan tanpa peramban, menjadi gerbang setiap rilis. Platform memberi saran; penggunalah yang memutuskan.

Buku: Subbab 11.12.

Kesalahan yang Sering Terjadi

Aturan bisnis di listener tombol; model yang mengimpor toolkit GUI; lupa menggambar gambar pertama.

Memberi tahu sebelum memvalidasi, atau sekali per field; mengubah daftar pendengar ketika sedang memberi tahu (telusuri salinannya).

Mendaftarkan lambda yang tak dapat dicabut langganannya (*lapsed listener*); membuat komponen Swing di luar *event dispatch thread*; lambda variabel gelung (`lambda k=key:`).

Percaya bahwa uji mandiri membuktikan jendela sudah benar; kelas bernama `View` yang mengerjakan tugas controller. Tanyakan kepada setiap metode: apakah ia menggambar, memutuskan, atau menerjemahkan?

Buku: Subbab 11.13.

Ringkasan: Lima Hal untuk Diingat

1. Model = data dan aturan, tanpa impor GUI; view = penyajian, tanpa aturan; controller = penerjemahan gerakan menjadi operasi model.

2. View adalah fungsi dari model, $V = R(S)$; beberapa view dari satu model tidak mungkin berselisih.

3. Observer: model memberi tahu tanpa mengenal view-nya — `PropertyChangeSupport` di Java, list berisi callable di Python; beri tahu sekali per perubahan logis, atas salinan daftarnya.

4. Metode model: validasi, ubah, beri tahu, kembalikan. Controller mengubah teks menjadi nilai; model menilai nilai-nilai itu.

5. Kontrak view memungkinkan `--selftest` dan unit test yang cepat; MVP dan MVVM juga menjauhkan aturan dari jendela.

Buku: Subbab 11.14.

Pertemuan Berikutnya

Bab 12 — Evaluasi dan Pembahasan III: Teks Prediktif dengan List, Map, Pohon, dan Papan Tombol. Tanpa teori baru: Bab 9, 10, dan 11 dikerjakan lewat soal-soal, langkah demi langkah.

Papan tombol hari ini, dibangun ulang di atas empat kamus — prototipe lambat, list terurut, map, dan pohon — diukur satu terhadap yang lain, lalu ditempatkan dalam MVC.

Sebelum pertemuan: kerjakan Soal Latihan 11.1, 11.3, 11.4, dan 11.8, lalu bacalah Subbab 12.1 sampai 12.3.

Buku: Subbab 11.15 dan Bab 12.

Bab 1

Kuliah 12 — Evaluasi dan Pembahasan III: Teks Prediktif dengan List, Map, Pohon, dan Papan Tombol

Pemrograman Berorientasi Objek (EF234302)

Buku: Irfan Subakti, *Pemrograman Berorientasi Objek dengan Java dan Python*

Dosen: Irfan Subakti — yifana@gmail.com

Buku: Bab 12, Evaluasi dan Pembahasan III: Teks Prediktif dengan List, Map, Pohon, dan Papan Tombol. Setiap halaman handout ini diakhiri dengan penunjuk ke buku, yang menjadi rujukan utama.

Pertemuan Hari Ini

1. Tanpa fitur bahasa baru. Bab 9, 10, dan 11 dimantapkan dengan mengerjakan enam belas soal, masing-masing dari halaman kosong, dalam Java dan Python (Subbab 12.1).

2. Metode enam langkah untuk soal koleksi dan antarmuka (Subbab 12.2).

3. Satu proyek, dikerjakan tuntas: teks prediktif ponsel lama — empat kamus, pengukuran waktunya, dan papan tombol dalam MVC (Subbab 12.3–12.7, Soal 12.1–12.10).

4. Enam soal baru di Profitina Laundry: koleksi yang dipilih dengan mengukur, kelas generik berbatas, rantai comparator, penghitung yang teruji, dua bug observer, tombol yang dibuat dalam gelung (Subbab 12.8–12.9, Soal 12.11–12.16).

5. Perbandingan soal-soal, Profitina dalam praktik, kesalahan yang sering terjadi, dan ringkasan (Subbab 12.10–12.13).

Buku: Bab 12, Subbab 12.1 sampai 12.13.

Titik Tolak Kita

Bab 9: koleksi — list, set, map — dan generik; `Comparable`, `Comparator`, dan `key=`.

Bab 10: jendela dan peristiwa; listener dan callback, sering berupa lambda atau kelas dalam.

Bab 11: Model–View–Controller; model memberi tahu setiap view terdaftar ketika ia berubah — pola observer.

Setiap soal menempuh satu rute: **soal**, **dengan bahasa kita sendiri**, **rancangan**, **penyelesaian Java**, **penyelesaian Python**, **baris demi baris**, **penyelesaian keliru yang menggoda**, **uji diri**.

> **CARA MEMAKAI PERTEMUAN INI.** Baca sebuah soal, tutup sisanya, luangkan sepuluh menit. Untuk soal kamus, tuliskan dulu *operasi apa saja* dan *seberapa sering*; untuk soal GUI, *apa yang harus diingat model*. Lalu bandingkan.

Buku: Subbab 12.1.

Metode Enam Langkah

DEFINISI. *Biaya* sebuah operasi adalah banyaknya langkah dasar yang diperlukannya, sebagai fungsi dari ukuran data 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.

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?

Tabel 12.1: “simpan kamusnya” bukanlah spesifikasi; “temukan kata-kata sebuah signature pada setiap tekanan tombol” adalah spesifikasi.

Buku: Subbab 12.2 dan Tabel 12.1.

Tombol, Huruf, dan Signature



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.

Gambar 12.1: delapan tombol membawa huruf; tombol 7 dan 9 membawa empat. Dengan $\kappa(c)$ tombol huruf c, signature sebuah kata adalah Persamaan (12.1):

$$\text{sig}(c_1 c_2 \ldots c_m) = \kappa(c_1)\,\kappa(c_2)\ \ldots\ \kappa(c_m) \tag{12.1}$$

Buku: Subbab 12.3.1, Gambar 12.1, dan Persamaan (12.1).

Kamus dan Rancangannya

sig(hello) = 43556; sig(home) = sig(good) = sig(gone) = 4663 — karena itu ponsel menawarkan kandidat: `*` berikutnya, `0` terima, `#` hapus.

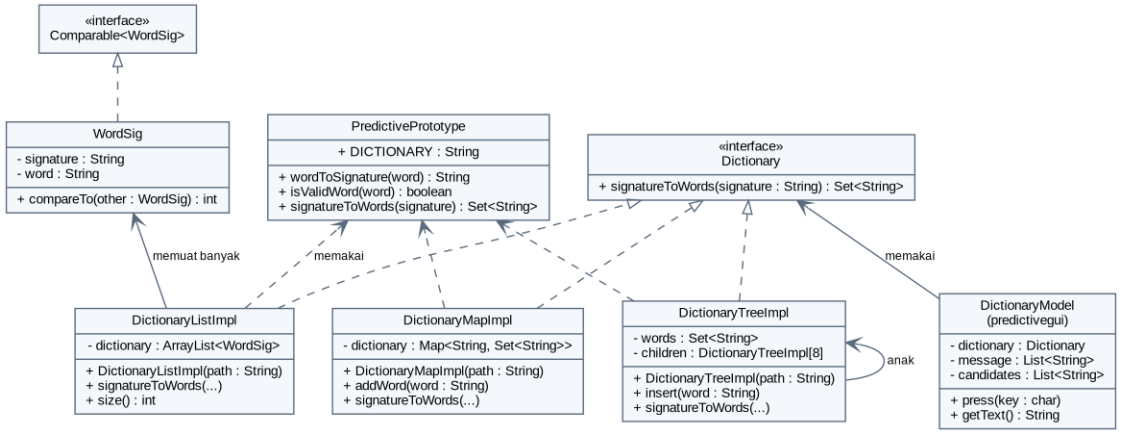
`words.txt` sengaja dibuat kotor: satu aturan untuk setiap program — pangkas, jadikan huruf kecil, simpan hanya kata a–z yang tidak kosong. 93 dari 99 baris lolos.

Paket Java `predictive` dan `predictivegui`: `javac predictive/*.java ...`, lalu `java predictive.Sigs2WordsList 4663 43556`. `PREDICTIVE_DICT` memilih berkas kamusnya.

Tabel 12.2, di halaman berikutnya, memetakan kesepuluh soal proyek ke empat bagian.

Buku: Subbab 12.3.1–12.3.3.

Seluruh Rancangan Sekilas



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.

Gambar 12.2: satu antarmuka, `Dictionary`, tiga implementasi — list terurut, hash map, dan pohon.

Buku: Subbab 12.3.4 dan Gambar 12.2.

Empat Bagian, Sepuluh Soal

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

Buku: Subbab 12.3.4 dan Tabel 12.2.

Soal 12.1 — Dari Kata ke Signature-nya

> **SOAL.** Tulislah `public static String wordToSignature(String word)`: `"home"` menghasilkan `"4663"`; karakter yang bukan huruf menjadi spasi. Bangun hasilnya dalam `StringBuffer`, dan jelaskan alasannya.

Telusuri kata dari kiri ke kanan; untuk setiap karakter, tambahkan tombolnya. **Tabel 12.3** menelusuri `"Home"`:

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`

Buku: Subbab 12.4.1 dan Tabel 12.3.

Mengapa StringBuffer?

`String` imutabel: `s = s + '6'` menyalin setiap karakter yang sudah ada. Kata sepanjang n huruf berbiaya Persamaan (12.2), sekitar $n^2/2$ salinan:

$$1 + 2 + \cdots + n = \sum_{k=1}^{n} k = \frac{n(n+1)}{2} \tag{12.2}$$

`StringBuffer` tumbuh di tempat: totalnya tetap sebanding dengan n (**Kode 12.1**).

Kode 12.1 `PredictivePrototype.java` (cuplikan): `wordToSignature` dengan `StringBuffer`, dan `keyOf` sebagai ekspresi switch.

```
    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';
            default -> ' ';
        };
    }
```

Buku: Subbab 12.4.1 dan Kode 12.1.

Penyelesaian Keliru yang Menggoda: Aritmetika

"Tiga huruf per tombol": $\kappa_{\text{wrong}}(c) = 2 + \lfloor \text{index}(c)/3 \rfloor$. Untuk `"you"` hasilnya `:68`:

```
wrong 'you'      : :68   right: 968
wrong 4663 list  : ['gone', 'good', 'goof', 'home', 'Home', 'hone', 'hood', 'hoof']
```

Tombol 7 membawa *empat* huruf, sehingga mulai dari `s` rumusnya maju satu tombol — salah tepat untuk s, v, y, z. Ia lolos setiap uji yang dibangun dari "home" dan "hello". Tabel tidak mungkin membuat kesalahan ini.

Python: `str` juga imutabel; kumpulkan digit dalam list lalu `"".join(...)` sekali. Jalan pintas `+=` di tempat milik CPython lenyap begitu ada satu referensi tambahan.

> **UJI DIRI.** `wordToSignature("Laundry")`? `"e-mail"`? — `"5286379"` dan `"3  6245"` (tanda hubung tetap menempati posisinya sebagai spasi).

Buku: Subbab 12.4.1 serta Kode 12.2 dan 12.4.

Soal 12.2 — Dari Signature ke Kata-katanya, Cara Lambat

SOAL. Tulislah `Set<String> signatureToWords(String signature)` yang membaca berkas kamus pada setiap panggilan, mengembalikan kata berhuruf kecil tanpa duplikat, dan jelaskan mengapa ia tidak efisien.

`Set` menegakkan "tanpa duplikat"; try-with-resources menutup berkas apa pun yang terjadi. Satu pencarian membaca semuanya, Persamaan (12.3):

$$T_{\text{proto}}(n) = O(n\,\bar{m}) \quad \text{per pencarian,} \tag{12.3}$$

4663: gone, good, goof, home, hone, hood, hoof; 5477: kiss dan lips (**Kode 12.2**).

Kode 12.2 `PredictivePrototype.java` (cuplikan): pencarian lambat yang membaca seluruh berkas untuk setiap pertanyaan.

```
    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;
    }
```

Buku: Subbab 12.4.2 dan Kode 12.1.

Soal 12.2: Jebakan-jebakannya

Keliru: simpan baris yang cocok dalam `List` tanpa diperkecil atau dipangkas — jawabannya memuat `Home` di samping `home`, dan `soap` hilang. Data uji yang bersih tidak menemukan keduanya.

Keliru: menutup secara manual, `finally { fileScan.close(); }` — berkas yang tidak ada membuat `fileScan` tetap null, dan `NullPointerException`-nya menyembunyikan galat yang sebenarnya.

`&&` berhenti pada `false` pertama, sehingga baris yang tidak sah tidak pernah dikonversi; string dibandingkan dengan `equals`, bukan `==`.

> **UJI DIRI.** Tambahkan satu uji murah yang melewati sebagian besar kata. Apakah Persamaan (12.3) berubah? — Ujilah `word.length() == signature.length()` lebih dulu; seluruh berkas tetap dibaca, sehingga biayanya tetap sebanding dengan n.

Buku: Subbab 12.4.2.

Soal 12.3 — Dua Program Baris Perintah

`Words2SigProto home hello world my name is` dan `Sigs2WordsProto` dengan signature yang dicetaknya (**Kode 12.3**); keluaran diurutkan (`TreeSet`, `sorted()`) karena urutan hash tidak dapat diulang:

Kode 12.3 `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)));
        }
    }
}
```

```
4663: [gone, good, goof, home, hone, hood, hoof]
43556: [hello]
96753: [world]
69: [my, ow, ox]
6263: [mane, name, oboe]
47: [is]
```

Buku: Subbab 12.4.3 dan Kode 12.6.

Soal 12.3: Mencetak Argumen

Keliru di Java: `"input: " + args` mencetak `input: [Ljava.lang.String;@659e0bfd` — larik tidak meng-*override* `toString()`. Pakailah `Arrays.toString(args)`.

Keliru di Python: `print("input:", sys.argv)` diawali nama berkas programnya. `sys.argv[0]` adalah programnya; `args[0]` di Java sudah argumen pertama.

> **UJI DIRI.** `Words2SigProto don't panic`? — `input: [don't, panic]` dan `output: 366 8 72642`; bila ditempelkan ke `Sigs2WordsProto`, spasi di `366 8` membuatnya menjadi tiga argumen.

Buku: Subbab 12.4.3.

Soal 12.4 — List Terurut dan Pencarian Biner

> **SOAL.** `DictionaryListImpl` membaca berkas sekali ke dalam `ArrayList<WordSig>`, diurutkan sekali menurut urutan *numerik* signature; `signatureToWords` memakai `Collections.binarySearch` dan mengembalikan *semua* kecocokan, dengan memindai ke atas dan ke bawah indeks yang ditemukan.

Kartu-kartu diurutkan menurut signature, seperti buku telepon: kartu dengan signature yang sama berdampingan.

Jebakan: `String.compareTo` menaruh `"22"` sebelum `"3"`. Signature tidak pernah berawalan nol, sehingga lebih sedikit digit berarti lebih kecil — Persamaan (12.4):

$$a < b \Leftrightarrow |a| < |b| \;\text{ atau }\; (|a| = |b| \;\text{ dan }\; a <_{\text{str}} b) \tag{12.4}$$

Buku: Subbab 12.5.1 dan Persamaan (12.4).

Urutan itu dalam Kode

Kode 12.4 `WordSig.java` (cuplikan): `compareTo` dalam urutan numerik signature, tanpa konversi ke bilangan.

```java
    @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);
    }
```

Kode 12.4: panjang dibandingkan dengan `Integer.compare`, yang tidak dapat melimpah; panjang yang sama kembali ke urutan string. Kedua field `private final`: mengubah kartu di dalam list terurut akan merusak urutan yang diandalkan pencarian biner.

Python: `__lt__` membandingkan tupel `(len(signature), signature)` — aturan yang sama dalam satu baris.

Buku: Subbab 12.5.1 dan Kode 12.9.

Contoh Tuntas: Pencarian Biner untuk 4663

Setiap tengokan membelah dua bagian yang masih dicari, sehingga paling banyak Persamaan (12.5) kartu diperiksa — 7 untuk 93 kartu, 18 untuk 200.000:

$$\lceil \log_2(n+1) \rceil \tag{12.5}$$

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

Tabel 12.4: pencarian berhenti di `goof`, indeks 40, di *tengah* blok 38–45. Berjalan ke bawah menemukan `good`, `gone`; ke atas `home` (dua kali), `hone`, `hood`, `hoof`.

Buku: Subbab 12.5.1, Persamaan (12.5), dan Tabel 12.4.

Pencarian Biner Ditambah Berjalan

Kode 12.5 `DictionaryListImpl.java` (cuplikan): satu pencarian biner, lalu berjalan ke bawah dan ke atas.

```
    @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());
        }
        for (int i = index + 1; i < dictionary.size() && matches(i, signature); i++) {
            result.add(dictionary.get(i).getWord());
        }
        return result;
    }
```

Kode 12.5: pasangan penguji `new WordSig(signature, "")` tidak memerlukan kata; bila tidak ditemukan hasilnya −(titik sisip) − 1. Biayanya Persamaan (12.6):

$$T_{\text{build}}(n) = O(n \log n), \qquad T_{\text{list}}(n) = O(\bar{m} \log n + k) \quad \text{per pencarian,} \tag{12.6}$$

Buku: Subbab 12.5.1 dan Kode 12.10.

Soal 12.4: Dua Penyelesaian Keliru yang Menggoda

```
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 2222222222222223:aaaaaaaaaaaaaaad
```

Berhenti di tempat pencarian biner berhenti: `[goof]` — satu kata dari tujuh. Selalu ujilah signature yang memiliki beberapa kata.

Membandingkan sebagai bilangan: `double` memuat 15–16 digit; `internationalization` punya 20 huruf. Tiga signature 17 digit dianggap *sama*, dan pencarian mendarat di `...223` ketika diminta `...222`. Persamaan (12.4) tepat.

> **UJI DIRI.** Untuk 5477 (kiss, lips), apa yang diputuskan tengokan pertama? — `4766:iron` lebih kecil, jadi `low = 47`; di mana pun pencarian mendarat di blok dua kartu itu, berjalan ke dua arah menemukan keduanya.

Buku: Subbab 12.5.1 dan Kode 12.11.

Soal 12.5 — Antarmuka Dictionary

Untuk setiap anggota publik list, tanyakan: apakah *pengguna* memerlukannya? Hanya pertanyaan “kata mana yang cocok?” — ditambah kontrak yang cermat untuk kasus-kasus yang janggal (**Kode 12.6**).

Kode 12.6 `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);
}
```

Keliru: menyalin wajah publik implementasi pertama, `ArrayList<WordSig> getDictionary()` — map tidak punya list semacam itu. Rancanglah dari kebutuhan pengguna.

Buku: Subbab 12.5.2 dan Kode 12.15.

Soal 12.6 — Pengukuran Waktu Pertama

`Sigs2WordsList` adalah program prototipe dengan satu baris diubah. Untuk adu cepatnya: 200.000 kata rekaan dari pembangkit Persamaan (12.7), identik di Java dan Python:

$$x_{i+1} = (1103515245\, x_i + 12345) \bmod 2^{31} \tag{12.7}$$

Membangun sekali, lalu 2.000 pencarian (prototipe hanya mendapat 5):

```
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
```

Prototipe memerlukan 62.283 µs per pencarian, list 2,70 — lebih dari dua puluh ribu kali lebih kecil. List membayar 420 ms sekali, seharga sekitar tujuh pencarian prototipe.

Buku: Subbab 12.5.3, Persamaan (12.7), dan Kode 12.19.

Adu Cepat di Baris Perintah

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

Tabel 12.5: empat detik lawan satu di Java, lima belas lawan satu di Python — tetapi satu detik milik list hampir seluruhnya waktu mulai dan membangun; 53 pencariannya hanya sekitar 150 µs.

Keliru: mengukur enam signature contoh pada `words.txt` — keduanya melaporkan `real 0m0.16s`: yang terukur adalah waktu mulai JVM.

> **UJI DIRI.** Sesudah berapa pencarian list balik modal terhadap prototipe? Terhadap map? — 420/62 ≈ 7; tidak pernah terhadap map, yang membangun *dan* menjawab lebih cepat. Keunggulan list adalah urutan.

Buku: Subbab 12.5.3 dan Tabel 12.5.

Soal 12.7 — Map Bernilai Jamak

SOAL. `DictionaryMapImpl` menyimpan map dari setiap signature ke koleksi kata-katanya: pencarian cepat tanpa memindai, dan penyisipan cepat. Pilihlah kelas `Map`-nya dan beri alasannya.

Sebuah lemari berlaci yang diberi label signature. `HashMap`: kode hash → ember, $O(1)$ harapan, tanpa urutan. `TreeMap`: $O(\log n)$, kunci terurut. Kita tidak pernah memerlukan urutan: `HashMap`, dengan Persamaan (12.8):

$$T_{\text{map}}(n) = O(\bar{m} + k) \text{ harapan per pencarian}, \qquad T_{\text{insert}} = O(\bar{m}) \text{ harapan}, \tag{12.8}$$

Buku: Subbab 12.6.1 dan Persamaan (12.8).

Temukan atau Buat Lacinya

Kode 12.7 `DictionaryMapImpl.java` (cuplikan): `addWord` menemukan atau membuat lacinya; pencarian menyerahkan view hanya-baca.

```
    /** 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 12.7: kelasnya `final` karena konstruktornya memanggil metode publik `addWord`. Python: `setdefault(signature, set()).add(word)` dan salinan `frozenset`.

Keliru: `map.put(signature, new HashSet<>(Set.of(word)))` *mengganti* lacinya — hanya `home` yang bertahan untuk 4663. **Keliru:** mengembalikan laci itu sendiri — `clear()` milik pemanggil mengosongkan kamusnya.

Buku: Subbab 12.6.1 dan Kode 12.23.

Soal 12.8 — Pohon yang Mencocokkan Prefiks

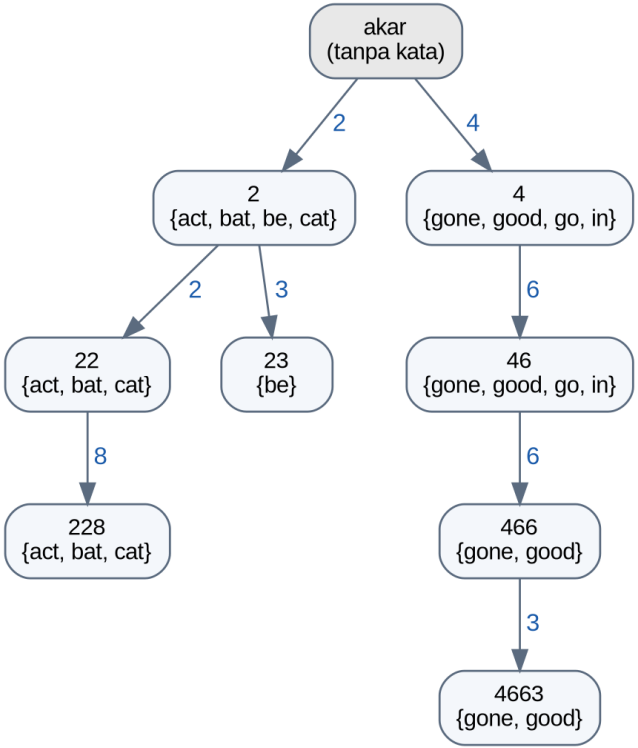
> **SOAL.** `DictionaryTreeImpl`: paling banyak delapan cabang per simpul, satu untuk setiap tombol 2–9; setiap simpul menyimpan kata-kata yang signature-nya *diawali* jalurnya; akar tidak menyimpan apa pun. Pencarian mengembalikan kata dan prefiks kata yang cocok, dipotong sepanjang signature.

Jalan raya: `good` disimpan dengan melaju 4, 6, 6, 3 dan meninggalkan salinan di *setiap* persimpangan — `4`, `46`, `466`, `4663`. Maka persimpangan `466` mengenal `good`, `gone`, dan `home`.

Pohon seperti ini, yang jalurnya mengeja kunci, disebut **trie** (dari re*trie*val).

Buku: Subbab 12.6.2.

Trie untuk Delapan Kata



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.

Gambar 12.3: `"228"` menemukan `{act, bat, cat}`; `"46"` menemukan empat kata, dipotong menjadi `{go, in}`; `"229"` tidak menemukan jalan dan mengembalikan himpunan kosong.

Buku: Subbab 12.6.2 dan Gambar 12.3.

Rekursi dengan Delegasi

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)`

Tabel 12.6 adalah resep Bab 8. Setiap kata disimpan di $|w|$ simpul, sehingga berlaku Persamaan (12.9):

$$T_{\text{tree}}(s) = O(\ell + W_s\,\ell) \text{ per pencarian}, \qquad \text{memori} = O\left(\sum_{w} |w|\right) \text{ referensi kata}, \tag{12.9}$$

90 kata berbeda, 354 huruf: 354 referensi di 120 simpul. String-nya dipakai bersama, tidak disalin.

Buku: Subbab 12.6.2, Tabel 12.6, dan Persamaan (12.9).

Trie dalam Kode

Kode 12.8 `DictionaryTreeImpl.java` (cuplikan): `insert` menambahkan kata ke *anak*, lalu mendelegasikan; `find` mengikuti jalurnya.

```
    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);
    }
```

Kode 12.8: kata ditambahkan ke anak, tidak pernah ke `this`, sehingga akar tetap kosong. `trim` memotong setiap kata menjadi n huruf di dalam sebuah set, sehingga `good` dan `goof` melebur menjadi `goo`.

Buku: Subbab 12.6.2 dan Kode 12.27.

Soal 12.8: Prefiks, dan Pohon Ujung-Saja

```
4663: [gone, good, goof, home, hone, hood, hoof]
43556: [hello]
96753: [worke, world]
69: [my, ow, ox]
6263: [mane, name, oboe]
47: [ir, is]
```

Pohon menambahkan `worke` untuk 96753 (dari `worked`, `worker`) dan `ir` untuk 47 (dari `iron`).

Keliru: menyimpan setiap kata hanya di *ujung* jalurnya — untuk 9675, di tengah mengetik `world`, jawabannya hanya `[work]`: tujuan pohon itu hilang.

> **UJI DIRI.** Sisipkan hanya `home` dan `hood`. Berapa simpulnya? `signatureToWords("466")`? — 5 simpul (keduanya 4663), dan `{hom, hoo}`.

Buku: Subbab 12.6.2.

Soal 12.9 — Satu Baris yang Diubah

`Sigs2WordsList`, `Sigs2WordsMap`, dan `Sigs2WordsTree` hanya berbeda pada kelas sesudah `new` (**Kode 12.9**):

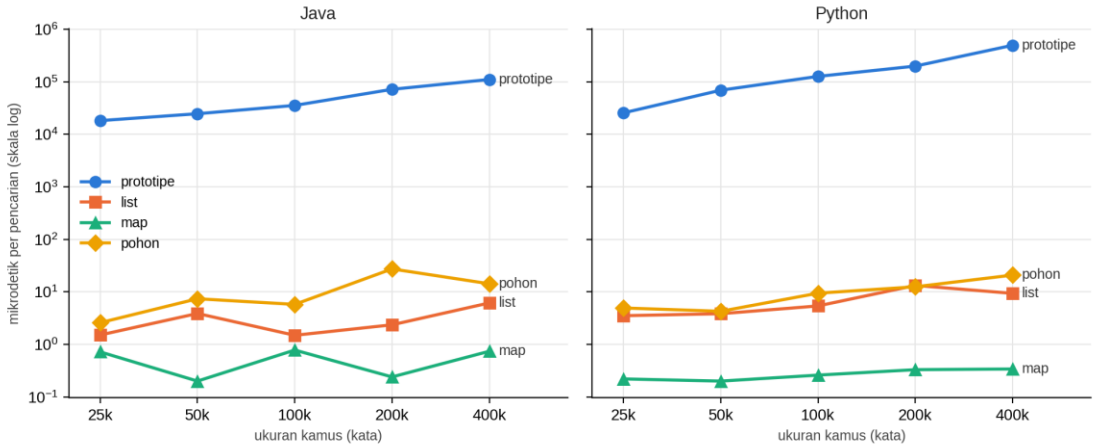
Kode 12.9 `Sigs2WordsMap.java` (cuplikan): satu-satunya baris yang berbeda adalah baris yang membuat kamusnya.

```
    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)));
        }
    }
```

Keliru: `DictionaryListImpl d = ...`, lalu kemudian `d.size()` — kini peralihan memerlukan dua perubahan, dan yang kedua tidak lolos kompilasi untuk map. Deklarasikan dengan tipe antarmuka.

Buku: Subbab 12.6.3 dan Kode 12.31.

Bagaimana Setiap Kamus Membesar



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 12.4: prototipe naik seiring ukuran, Persamaan (12.3); map tetap datar, Persamaan (12.8); list merayap naik dengan log$n$; pohon berada di atas list karena menjawab pertanyaan yang lebih besar.

Buku: Subbab 12.6.3 dan Gambar 12.4.

Keempat Kamus Dibandingkan

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

Tabel 12.7: list lawan map, dan pohon lawan map, terasa bedanya pada banyak pencarian — tetapi hanya pohon yang menjawab *kata apa yang mungkin sedang diketik pengguna saat ini*.

> **UJI DIRI.** 13,57 µs per pencarian pohon lawan 0,22 untuk map, pada empat tekanan tombol per detik — apakah berpengaruh? — Tidak: 56 µs kerja per detik.

Buku: Subbab 12.6.3 dan Tabel 12.7.

Soal 12.10 — GUI Teks Prediktif

> **SOAL.** Dalam paket `predictivegui`, GUI ber-MVC untuk `Dictionary` apa pun. Model boleh menyimpan kamus dan kata-kata yang telah dimasukkan, tetapi **tanpa signature atau spasi**. Controller-nya dua belas tombol; view-nya jendela teks yang tidak dapat diketiki pengguna. `2` memberi a, b, atau c; `*` bergilir; `0` menuntaskan kata; `#` menghapus huruf terakhir.

Apa yang harus diingat model? Kata-kata yang selesai, kandidat kata yang sedang diketik, dan kandidat mana yang ditampilkan. Digit yang ditekan tidak perlu disimpan: kata yang ditampilkan punya satu huruf per tombol, sehingga signature-nya *adalah* digit-digit itu.

Buku: Subbab 12.7.1.

Apa yang Dilakukan Setiap Tombol pada Model

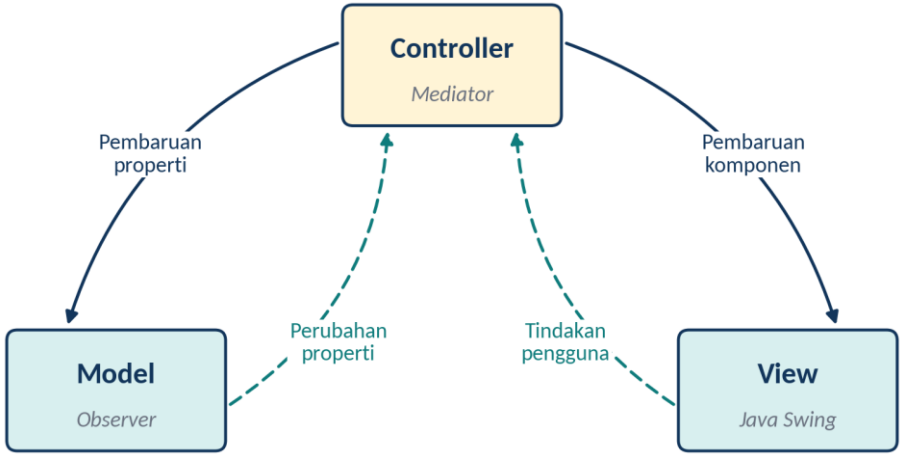
Tabel 12.8 Tiga *field* model (kata-kata yang selesai, kandidat, indeks kandidat yang ditampilkan) dan pengaruh setiap tombol terhadapnya. Tidak ada signature yang pernah disimpan.

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

Tabel 12.8: tombol yang tidak cocok dengan apa pun diabaikan — ponsel yang menolak tombol yang mustahil berperilaku seperti ponsel. Sesudah 4, 6, 6 pohon memberi `gon`, `goo`, `hom`, `hon`, `hoo`, `inn`.

Buku: Subbab 12.7.1 dan Tabel 12.8.

Model, View, Controller



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

Gambar 12.5: controller memanggil `model.press(key)`; model berubah dan memberi tahu para listener-nya; view meminta teksnya kepada model. Pemberitahuan memakai `List<Runnable>`, bukan `Observable` yang usang.

Buku: Subbab 12.7.1 dan Gambar 12.5.

Label yang Diketik dengan Tangan



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`.

Gambar 12.6: tombol 9 bertuliskan `vwxy`, padahal **Gambar 12.1** menaruh w, x, y, z pada 9. Labelnya diketik terpisah dari huruf yang dipakai kode signature; controller yang baru mengambil labelnya dari satu tabel.

Buku: Subbab 12.7.1 dan Gambar 12.6.

Model: Signature Dihitung Ulang

Kode 12.10 `DictionaryModel.java` (cuplikan): `typeKey` menghitung ulang signature dari kata yang ditampilkan; `nextCandidate` berputar dengan `%`.

```
    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
        }
    }
```

Kode 12.10: "digit-digit sejauh ini, ditambah yang baru" adalah `wordToSignature(currentWord()) + key`. `lookUp` mengurutkan jawabannya melalui `TreeSet`.

Buku: Subbab 12.7.1 dan Kode 12.35.

Controller: Dua Belas Tombol dari Satu Tabel

Kode 12.11 `KeypadController.java` (cuplikan): satu tombol per baris tabel; setiap listener menangkap tombolnya sendiri.

```
    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);
        }
    }
```

Kode 12.11: `char key` adalah variabel *baru* pada setiap putaran, tidak pernah ditugasi ulang, sehingga lambda boleh menangkapnya (Soal 12.16). Python: `make_command(model, key)` memberi setiap tombol fungsinya sendiri.

Buku: Subbab 12.7.1 dan Kode 12.37.

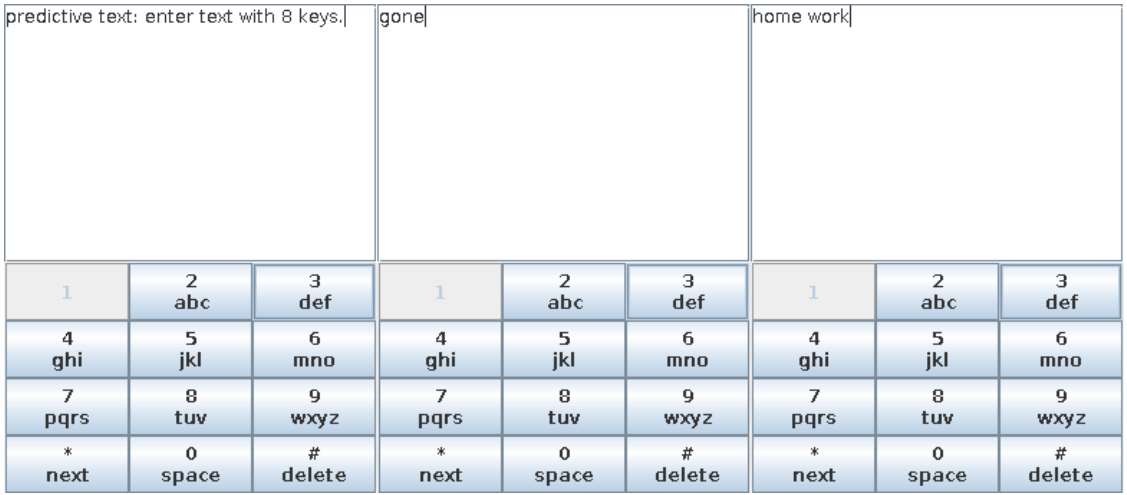
Contoh Tuntas: Uji Mandiri sebagai Film

```
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 "
```

`g`, `go`, `gon` (yang pertama dari enam prefiks), `gone`; `*` berjalan sampai `home`; `0` menetapkannya. Lalu `3` memberi `worke` (dari `worked`), dan `#` mencabut `e`-nya: 9675 dicari ulang dan memberi `work`.

Buku: Subbab 12.7.1 dan Kode 12.38.

Jendela Swing



Gambar 12.7 Jendela Swing saat mulai (kiri), sesudah menekan 4, 6, 6, 3 (tengah), dan sesudah `* * * 0 9 6 7 5 3 #` (kanan). Tombol 1 nonaktif karena tidak membawa huruf.

Gambar 12.7: jendela sungguhan, dibuka pada layar maya di tiga saat.

Buku: Subbab 12.7.1 dan Gambar 12.7.

Jendela Tkinter



Gambar 12.8 Jendela Tkinter pada tiga saat yang sama seperti pada **Gambar 12.7**.

Gambar 12.8: model tidak mengimpor apa pun dari Tkinter; view dan controller mengimpornya di dalam konstruktornya, sehingga uji mandiri tetap berjalan di tempat yang tidak memiliki Tkinter.

Buku: Subbab 12.7.1 dan Gambar 12.8.

Soal 12.10: Jebakan-jebakannya

Keliru: menjadikan area teks *sebagai* model — membacanya kembali dengan `getText()`, mengubahnya, menuliskannya. Lalu `*` menebak kandidat dari teks, `#` menebak di mana kata dimulai, pengguna dapat merusak satu-satunya salinan data, dan tidak ada yang dapat diuji tanpa jendela.

Keliru: menyimpan `currentSig` di samping kata-kata. Dua field yang menggambarkan satu fakta dapat berselisih; satu field yang dihitung ulang bila diperlukan tidak.

Model diuji sendirian dengan kamus palsu yang ditulis sebagai lambda — `Dictionary` hanya punya satu metode (Kode 12.39 di buku).

> **UJI DIRI.** Sesudah `4 3 5 5 6 0 # 9 6`? — `hello wo`: `#` tidak berbuat apa-apa, karena kata yang sudah selesai tidak disunting.

Buku: Subbab 12.7.1.

Soal 12.11 — Memilih Koleksi dengan Mengukurnya

SOAL. Di loket pengambilan ada 100.000 nomor pesanan yang siap: katakan seketika apakah nomor yang dipindai sudah siap; cetak semua nomor secara terurut di akhir hari; catat setiap nomor yang dipindai sekali saja, menurut urutan kemunculan pertamanya.

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

Tabel 12.9: satu struktur untuk setiap tugas. Python: list terurut dengan `bisect` menggantikan `TreeSet`; `list(dict.fromkeys(items))` menjaga urutan kemunculan pertama.

Buku: Subbab 12.8.1 dan Tabel 12.9.

Soal 12.11: Hasil Pengukurannya

```
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
```

10.000 pertanyaan: `ArrayList` 1.470 ms, `HashSet` 5,5 ms — faktor sekitar 270 (Python: sekitar 5.000). Hanya koleksi berurutan-sisip dan terurut yang *menjanjikan* urutan.

Keliru: `if (!seen.contains(id)) seen.add(id);` pada `ArrayList` — $O(n^2)$, berlipat empat setiap kali datanya berlipat dua. Satu perubahan tipe memperbaikinya: `LinkedHashSet`.

Buku: Subbab 12.8.1 dan Kode 12.45.

Soal 12.12 — Kelas Generik Berbatas

`Bin<T extends Weighed>` memegang barang sejenis; `heaviest()` mengembalikan `T`, sehingga `shirts.heaviest().owner()` tidak memerlukan *cast* (**Kode 12.12**).

Kode 12.12 `GenericBin.java` (cuplikan): kelas generik yang parameter tipenya dibatasi oleh `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>
        }
```

`totalKg(List<? extends Weighed>)` menerima list kemeja atau handuk; `<T extends Comparable<? super T>> T maxOf(List<? extends T>)` adalah metode generik. Barang-barangnya berupa **record**.

Buku: Subbab 12.8.2 dan Kode 12.47.

Soal 12.12: Invariansi, Sekali Lagi

Keliru: `static double totalKg(List<Weighed> things)`, dipanggil dengan `List<Towel>`:

```
WrongTotal.java:12: error: incompatible types: List<Towel> cannot be converted to List<Weighed>
        System.out.println(totalKg(towels));
                                   ^
```

Seandainya `List<Towel>` adalah `List<Weighed>`, metode itu dapat menambahkan kemeja ke handuk milik pemanggil. `? extends Weighed` merelakan hak menulis ke list demi keluwesan; mypy berkata *"list" is invariant — Consider using "Sequence"*.

> **UJI DIRI.** Metode yang menambahkan tiga handuk dan menerima `List<Towel>`, `List<Weighed>`, dan `List<Object>`? — `static void addTowels(List<? super Towel> into)`: produsen `extends`, konsumen `super`.

Buku: Subbab 12.8.2.

Soal 12.13 — Rantai Comparator

Kilat dulu; lalu waktu siap yang lebih awal; lalu nama pelanggan; lalu nomor pesanan. **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

Java: setiap kunci menyatakan arahnya sendiri (**Kode 12.13**); Python: kunci `(not o.express, o.ready_at, o.customer, o.id)`.

Kode 12.13 `OrderSorting.java` (cuplikan): urutan papan dengan empat kunci; hanya kunci kilat yang dibalik.

```java
    /** 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);
```

Buku: Subbab 12.8.3, Tabel 12.10, dan Kode 12.49.

Soal 12.13: Jebakan-jebakannya

```
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]
```

Keliru: setiap kunci maju, lalu `.reversed()` di akhir — *seluruh* rantai terbalik: waktu siap yang lebih lambat dulu, nama dari Z ke A. Balikkan hanya kunci yang memerlukannya.

Keliru: `(a, b) -> (int) (a.kg() - b.kg())` — 6,0 dikurangi 5,2 adalah 0,8, di-*cast* menjadi 0, "sama". Pakailah `comparingDouble`.

UJI DIRI. Terberat dulu, berat yang sama menurut id menaik, di Python? — `key=lambda o: (-o.kg, o.id)`.

Buku: Subbab 12.8.3.

Soal 12.14 — Penghitung dengan Model yang Teruji

Kantong yang diserahkan hari ini; satu mesin memuat 8 kantong, sehingga mesin adalah kantong dibagi 8 *dibulatkan ke atas*, Persamaan (12.10):

$$\lceil \frac{b}{8} \rceil = \lfloor \frac{b+7}{8} \rfloor \tag{12.10}$$

Kode 12.14 `BagCounterApp.java` (cuplikan): aturan model — mesin dibulatkan ke atas, tidak pernah di bawah nol, memberi tahu hanya pada perubahan sungguhan.

```
        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 12.14: dua view label membaca model yang *sama*, sehingga tidak mungkin berselisih. **Keliru:** tanpa model — urai label, tambah satu, tulis kembali: sesudah sembilan klik, `Machines needed: 0`.

Buku: Subbab 12.9.1, Persamaan (12.10), dan Kode 12.51.

Soal 12.15 — Dua Bug Observer

Aturan 1: ubah keadaan dulu, baru beri tahu. **Aturan 2:** beri tahu *salinan* daftar listener — sebuah listener mungkin berhenti berlangganan sendiri (alarm sekali pakai) (**Kode 12.15**).

Kode 12.15 `ObserverBugs.java` (cuplikan): cabang yang cermat dan cabang yang ber-bug dari `use`.

```
        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;
            }
        }
```

```
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
```

Buku: Subbab 12.9.2 dan Kode 12.53.

Soal 12.15: Bug yang Berisik, Bug yang Diam

`ArrayList` Java mendeteksi perubahan dan melempar `ConcurrentModificationException`: tampilan melewatkan pencucian itu — *display shows 11 l, tank has 8 l*.

`for` Python melanjutkan ke indeks berikutnya dan *melompati* tampilan tanpa sepatah kata — *display shows 11 l, tank has 5 l*. Versi yang diam justru lebih berbahaya.

Perbaikan keliru: menangkap eksepsinya atau menggelung dengan indeks — listener tetap terlewat. `CopyOnWriteArrayList` memang dibuat untuk daftar listener.

> **UJI DIRI.** Sebuah listener *menambahkan* listener lain selama pemberitahuan di tangki yang cermat. Apakah yang baru ikut dipanggil sekarang? — Tidak: gelung menelusuri salinannya; ia baru mendengar mulai peristiwa berikutnya.

Buku: Subbab 12.9.2.

Soal 12.16 — Tombol yang Dibuat dalam Gelung

Java menangkap nilai: hanya variabel lokal yang *effectively final*; `for (int i ...; i++)` dengan `keys.charAt(i)` di dalam lambda tidak lolos kompilasi. **Python menangkap variabel:** lambda membaca nilainya ketika *dijalankan* (**Kode 12.16**).

Kode 12.16 `keypad_closures.py` (cuplikan): bug *late binding*, argumen bawaan, dan fungsi pabrik.

```python
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]
```

Keliru di Java: larik satu elemen “memperbaiki” galat kompilasi — mengeklik 4, 6, 6, 3 mencatat `####`, bug Python yang dihadirkan kembali dengan tangan.

Buku: Subbab 12.9.3 dan Kode 12.56.

Keenam Belas Soal Dibandingkan

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: $O(n^2)$

Soal	Topik	Gagasan utama	Yang terlewat oleh penyelesaian keliru
12.12	generik	parameter tipe berbatas; `? extends`	`List<Towel>` bukan `List<Weighed>`

Tabel 12.11 (Lanjutan) *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
12.13	*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 `#`

Tabel 12.11: bacalah kolom terakhir dari atas ke bawah — setiap penyelesaian keliru yang menggoda lolos uji yang jelas dan gagal pada satu masukan tertentu atau satu urutan peristiwa tertentu.

Buku: Subbab 12.10 dan Tabel 12.11.

Profitina dalam Praktik

Koleksi yang tepat untuk pertanyaan yang sering: "sudah pernah melihat halaman ini?" untuk setiap tautan — himpunan hash berisi alamat kanonis; tren skor mingguan — deret terurut yang diekspor ke CSV.

Satu antarmuka, banyak implementasi: setiap *sinyal* dinilai dan dijelaskan di balik satu antarmuka, seperti `Dictionary`; sinyal baru adalah kelas baru.

Model, view, dan observer: penghitung "N baru" adalah view atas model; Server-Sent Events mendorong perubahan; posting optimistis mengubah model lokal lebih dulu dan memulihkannya bila gagal.

Sistem menyarankan; manusia memilih — seperti teks prediktif menawarkan kandidat dan pengguna mengonfirmasi salah satunya.

Buku: Subbab 12.11.

Kesalahan yang Sering Terjadi

Memilih koleksi karena kebiasaan; membandingkan bilangan sebagai string atau string sebagai bilangan; hanya menyimpan temuan pertama pencarian biner.

Menyerahkan koleksi milik sebuah objek; mengganti alih-alih menambah di map bernilai jamak; mengharapkan urutan dari himpunan hash.

Tipe konkret di tempat antarmuka sudah cukup; membalik seluruh rantai comparator; mengurangkan untuk membandingkan.

Keadaan di view; memberi tahu sebelum berubah, atau menelusuri daftar yang hidup; menangkap variabel gelung yang berubah; mengukur hal yang salah.

Buku: Subbab 12.12.

Ringkasan: Lima Hal untuk Diingat

1. Soal koleksi adalah soal operasi: daftarkan operasinya beserta frekuensinya, cocokkan masing-masing dengan struktur, sembunyikan pilihannya di balik antarmuka.

2. Prototipe $O(n\bar{m})$ per pencarian; list terurut $O(\bar{m}\log n + k)$; hash map $O(\bar{m} + k)$ harapan; trie — satu-satunya yang menjawab prefiks.

3. Ukurlah fase yang berbeda, dengan data besar, beberapa kali.

4. GUI ber-MVC menyimpan setiap fakta sekali, di model yang diuji tanpa jendela; view menampilkan, controller meneruskan.

5. Generik bersifat invarian; kunci comparator menyatakan arahnya sendiri; observer berubah dulu lalu memberi tahu salinannya; listener yang dibuat dalam gelung menangkap nilainya sendiri.

Buku: Subbab 12.13.

Pertemuan Berikutnya

Bab 13 — Iterator, Imutabilitas, dan Koleksi Lanjutan. Hari ini kita menelusuri koleksi dengan `for`; berikutnya kita menulis penelusurnya sendiri.

Iterator dan `Iterable`, koleksi iterable buatan sendiri, mengubah koleksi selagi menelusurinya; salinan, imutabilitas dari nol, objek nilai; stream, comprehension, dan `itertools`.

Sebelum pertemuan: kerjakan Soal Latihan 12.1, 12.3, 12.6, dan 12.8, lalu bacalah Subbab 13.1 sampai 13.3.

Buku: Subbab 12.14 dan Bab 13.

Kuliah 13 — Iterator, Imutabilitas, dan Koleksi Lanjutan

Pemrograman Berorientasi Objek (EF234302)

Bab 13 buku ini dalam bentuk ringkas, untuk dipakai di kelas.

Buku: Irfan Subakti, *Pemrograman Berorientasi Objek dengan Java dan Python*.

Pengajar: Irfan Subakti — yifana@gmail.com

Hari ini kita membuka tutup perulangan `for`, belajar mengurutkan objek yang tidak memiliki urutan yang jelas, dan berkenalan dengan objek yang tidak dapat berubah, yang ternyata termasuk alat paling berguna yang kita miliki. Buku: Bab 13, paragraf pembuka.

Agenda

1. **Menelusuri.** Protokol iterator di Java dan Python, perulangan for-each, `ListIterator`, dan iterable buatan sendiri, ditulis dengan tangan dan sebagai *generator* Python.
2. **Mengubah selagi menelusuri.** Mengapa Java melemparkan `ConcurrentModificationException`, mengapa Python diam-diam melompati elemen, dan pola-pola yang aman.
3. **Urutan.** `Comparable`, `Comparator`, fungsi kunci Python, kontrak yang harus dipegang setiap perbandingan, dan perbandingan yang hanya tampak benar.
4. **Salinan dan imutabilitas.** Aliasing, salinan dangkal dan dalam; kelas imutabel, record, dataclass beku, tampilan, dan salinan defensif.
5. **Nilai dan pipeline.** Objek nilai dan hashing; stream, comprehension, dan `itertools`; buku besar Profitina Laundry.

Halaman-halaman ini mengikuti urutan buku, dan soal latihan di akhir bab adalah pekerjaan rumahnya. Buku: Subbab 13.2 sampai 13.11.

Di Mana Kita Berhenti

Bab 9 memberi kita tiga keluarga besar koleksi, yaitu list, set, dan map, beserta generik yang membuat `List<String>` hanya dapat memuat string.

Bab 10 dan 11 menampilkan jendela di layar dan membagi program menjadi model, view, dan controller; **Bab 12** mengerjakan soal-soal yang memadukan semuanya.

Selama itu kita memakai koleksi dengan cara sederhana: menambah, mencari, dan menelusuri dengan perulangan `for`, **tanpa bertanya bagaimana perulangan itu mengunjungi setiap elemen**.

> **WAWASAN KUNCI.** Sebuah *iterator* memisahkan kegiatan **menelusuri** koleksi dari kegiatan **menyimpannya**; *imutabilitas* memisahkan kegiatan **menciptakan** objek dari kegiatan **memakainya**. Setiap bagian program perlu mengetahui lebih sedikit hal dan memiliki lebih sedikit cara untuk keliru.

Kini laundry menginginkan riwayat setiap pesanan, daftar pesanan seorang pelanggan, laporan mingguan, dan harga yang tidak mungkin berubah secara tidak sengaja. Buku: Subbab 13.1.

Apa Itu 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.

`HashSet` tidak mempunyai "elemen nomor 2"; pohon atau berkas yang dibaca baris demi baris sama sekali tidak berindeks. Kita memerlukan satu cara yang berlaku untuk **setiap** koleksi: *berikan elemennya satu per satu sampai habis*.

Java: `java.util.Iterator<E>` dengan `boolean hasNext()` dan `E next()`; `next()` sesudah elemen terakhir melemparkan `NoSuchElementException`. `remove()` bersifat opsional. Setiap koleksi mempunyai `iterator()`.

Pola Iterator dari Gamma dkk. berakar pada bahasa CLU tahun 1970-an; Java dan Python sama-sama membangun perulangan `for` di atasnya.

Buku: Subbab 13.2 dan 13.2.1.

Protokolnya di Java

Kode 13.1 `IteratorBasics.java`, blok 1: mencetak setiap string dengan iterator eksplisit.

```
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());
        }
```

```
Luffy, Monkey D.
Zoro, Roronoa
Robin, Nico
```

Pada Kode 13.1, `crew` dideklarasikan dengan tipe paling umum yang memadai, yaitu `Collection`. `iterator()` menciptakan objek tersendiri; `hasNext()` ditanyakan sebelum setiap langkah; `next()` mengembalikan string *dan* bergerak maju.

Memanggil `next()` dua kali dalam satu badan perulangan menghabiskan dua elemen. Buku: Subbab 13.2.1 dan Kode 13.1.

Satu Elemen Terlalu Banyak

Kode 13.2 `IteratorBasics.java`, blok 4: `next()` keempat pada koleksi berisi tiga elemen.

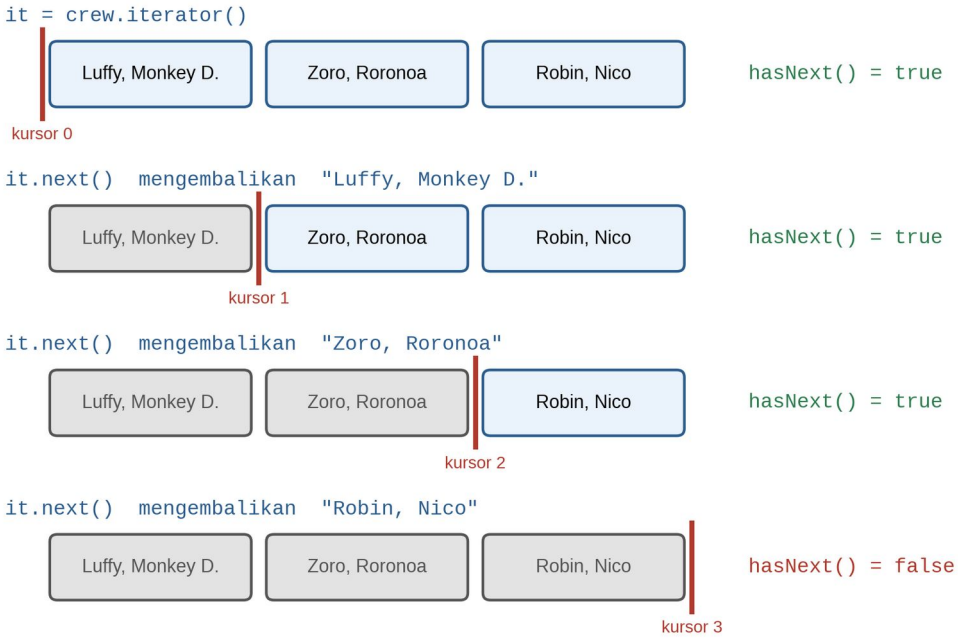
```
        // 4. Meminta satu elemen terlalu banyak.
        Iterator<Integer> it3 = c.iterator();
        it3.next();
        it3.next();
        it3.next();
        System.out.println("hasNext() after three calls to next(): " + it3.hasNext());
        try {
            it3.next();
        } catch (NoSuchElementException e) {
            System.out.println("fourth next(): " + e.getClass().getSimpleName());
        }
```

```
hasNext() after three calls to next(): false
fourth next(): NoSuchElementException
```

Blok 2 program yang sama mengubah salinan lokal: `element += " and Zoro, Roronoa"` membangun string baru dan mengikat ulang `element`; koleksinya tetap mencetak `Luffy, Monkey D.`. Blok 3 menjumlahkan `[3, 2, 4]` menjadi 9 dengan perulangan for-each.

Pada Kode 13.2, program yang benar tidak pernah bertemu `NoSuchElementException`: ia bertanya `hasNext()` terlebih dahulu. Dengan elemen yang mutabel, `element.setName(...)` akan mengubah objek di dalam koleksi (Subbab 13.7). Buku: Subbab 13.2.1.

Kursor Berada di Antara Elemen



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`.

Pada Gambar 13.1, beri nomor celah-celahnya $0, 1, \ldots, n$. Kursor c mulai di 0 dan setiap `next()` menambahkan satu:

$$\text{hasNext\(\) } \Leftrightarrow c \neq n \tag{13.1}$$

Persamaan (13.1) tampak terlalu jelas untuk dituliskan; beberapa halaman lagi rumus ini menjelaskan bug yang diam-diam melompati sebuah elemen. Buku: Subbab 13.2.1, Gambar 13.1, dan Persamaan (13.1).

Protokol yang Sama di Python

Kode 13.3 `iterator_basics.py`, paruh pertama: `iter()`, `next()`, dan `StopIteration`, dituliskan seperti yang dikerjakan perulangan `for` di balik layar.

```
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)
```

Kode 13.3: tidak ada `hasNext()`; Anda *mencoba* mengambil elemen berikutnya, dan `StopIteration` adalah jawaban "tidak ada lagi" (PEP 234). `next(it, "no more")` mengembalikan nilai bawaan; `iter(it) is it` bernilai `True`.

Tidak ada yang menulis perulangan `while True` ini dalam kode sungguhan, tetapi menuliskannya sekali menunjukkan bahwa `for element in crew:` bukan sihir. Buku: Subbab 13.2.2.

Dua Protokol Berdampingan

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`

Tabel 13.1: iterator biasa hanya bergerak **maju** dan hanya **membaca**. Kesederhanaan itu justru keunggulan: setiap koleksi sanggup menepati janji sekecil itu, bahkan barisan yang dihitung saat diminta dan tidak pernah disimpan.

Buku: Subbab 13.2.2 dan 13.2.3, serta Tabel 13.1.

For-Each Adalah Iterator yang Menyamar

Spesifikasi Bahasa Java mendefinisikan perulangan yang disempurnakan melalui penerjemahan. `for (Integer i : c) { sum += i; }` menjadi:

```
for (Iterator<Integer> $it = c.iterator(); $it.hasNext(); ) {
    Integer i = $it.next();
    sum += i;
}
```

Apa pun yang mengimplementasikan `Iterable<T>` (satu metode, `Iterator<T> iterator()`) boleh ditulis sesudah titik dua. `for i in c:` di Python diterjemahkan dengan cara yang sama menjadi perulangan pada Kode 13.3.

> **PERINGATAN.** Di dalam perulangan for-each Anda **tidak memegang iteratornya**, sehingga tidak dapat memanggil `remove()` miliknya.

Untuk larik, perulangan yang disempurnakan di Java memakai indeks, karena larik tidak mengimplementasikan `Iterable`. Buku: Subbab 13.3.

Menghapus Duplikat yang Bersebelahan

Kode 13.4 `RemoveDuplicates.java`, metode klasik: mengingat elemen sebelumnya dan mengumpulkan elemen unik dalam list baru.

```
    // 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;
    }
```

Dua cacat tersembunyi pada Kode 13.4, dari satu pengandaian yang ceroboh. **List kosong:** `input.get(0)` melemparkan `IndexOutOfBoundsException`. **`null` sebagai penanda:** untuk `[null, null, "Nami"]`, `previous` tetap `null` selamanya, dan `"Nami"` hilang.

Setiap kali nilai sungguhan seperti `null`, `0`, atau `-1` dipakai untuk berarti "belum ada apa-apa", tanyakan apakah nilai itu juga dapat muncul sebagai data; jika dapat, pakailah penanda terpisah. Buku: Subbab 13.3.1.

Di Tempat, dengan ListIterator

Kode 13.5 `RemoveDuplicates.java`, metode di tempat: `ListIterator.remove()` menghapus elemen yang baru saja dikembalikan `next()`.

```
    // 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;
        }
    }
```

Versi aman yang menyalin, `dedupSorted`, memperbaiki kedua cacat dengan `boolean first` tersendiri dan `Objects.equals`. Pada Kode 13.5, `ListIterator` juga dapat bergerak mundur (`hasPrevious`, `previous`), melaporkan indeks, serta melakukan `set`, `add`, dan `remove` di posisi kursor; `listIterator(list.size())` dimulai sesudah elemen terakhir. Buku: Subbab 13.3.1 dan 13.3.2.

Duplikat di Python

Kode 13.6 `remove_duplicates.py`: versi dengan penanda, dan penghapusan di tempat dengan menelusuri indeks dari belakang.

```python
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]
```

Pada Kode 13.6, menghapus `items[i]` hanya menggeser elemen *sesudah* `i`, yang sudah dikunjungi perulangan. `[key for key, _ in groupby(my_list)]` memberi jawaban yang sama dalam satu baris; `lst[i] = e` mengerjakan tugas `ListIterator.set()`.

Python tidak memiliki `ListIterator` *karena tidak memerlukannya: list berindeks, dan* `reversed(range(len(xs)))` *menelusuri mundur. Buku: Subbab 13.3.2 dan Kode 13.4.*

Iterable Buatan Sendiri

12,5 kg cucian, mesin berkapasitas 5 kg: rencananya 5, 5, 2,5. Rencana itu tidak pernah perlu **menyimpan** muatannya; muatan berikutnya dihitung dari sisa yang ada.

Kode 13.7 `LoadPlanDemo.java`, koleksi `LoadPlan`: ia hanya memegang apa yang mendefinisikan rencana, dan `iterator()` mengembalikan `LoadIterator` baru.

```
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();
    }
```

*Pada Kode 13.7, pisahkan **koleksi** (yang mendefinisikan data) dari **iterator** (posisi satu penelusuran). Buku: Subbab 13.4 dan 13.4.1.*

Kelas Iterator Dalam

Kode 13.8 `LoadPlanDemo.java`, kelas dalam privat `LoadIterator`, yang membaca field `LoadPlan`-nya secara langsung.

```
    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;
        }
    }
```

```
a is at its third load: 2.5 kg
b is still at its first: 5.0 kg
```

Keluaran Kode 13.8 memperlihatkan tiga sifat: perulangan for-each berjalan, iterator-iteratornya ***independen*** *(`a` dan `b`), dan rencananya* ***malas****: 133.334 muatan dari rencana sejuta kilogram dihitung satu per satu dan tidak pernah disimpan. Buku: Subbab 13.4.1.*

Python: Generator Menuliskan Iteratornya

Kode 13.9 `load_plan.py`, `GeneratorLoadPlan`: iterable yang sama, dengan `__iter__` yang ditulis sebagai fungsi generator.

```
class GeneratorLoadPlan(LoadPlan):
    """Iterable yang sama; fungsi generator menuliskan iteratornya untuk kita."""

    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
```

Berkas yang sama juga menulis `LoadIterator` dengan tangan, lengkap dengan `__init__`, `__iter__`, `__next__`, dan `StopIteration` eksplisit; pada Kode 13.9 semua itu menyusut menjadi perulangan empat baris dengan `yield` (PEP 255). Buku: Subbab 13.4.2.

Melihat Generator Berhenti Sejenak

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 sampai `yield` berikutnya, mengembalikan nilainya, lalu berhenti lagi; ketika badan fungsi berakhir, generator membangkitkan `StopIteration`.

```
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]
```

Sesudah `g = iter(lazy)` belum ada yang dihitung; `next(g)` berjalan sampai `yield` pertama; `list(g)` mendorong generator sampai habis.

Java tidak mempunyai `yield` untuk iterator; kelas dalam eksplisit seperti pada Kode 13.8 tetap menjadi cara bakunya. Buku: Subbab 13.4.2.

Iterable Bukanlah Iterator

`it = iter(plan)`: `list(it)` memberi `[5.0, 5.0, 2.5]`, dan `list(it)` kedua memberi `[]`. Iteratornya **sudah habis**; tidak ada cara untuk memutarnya kembali.

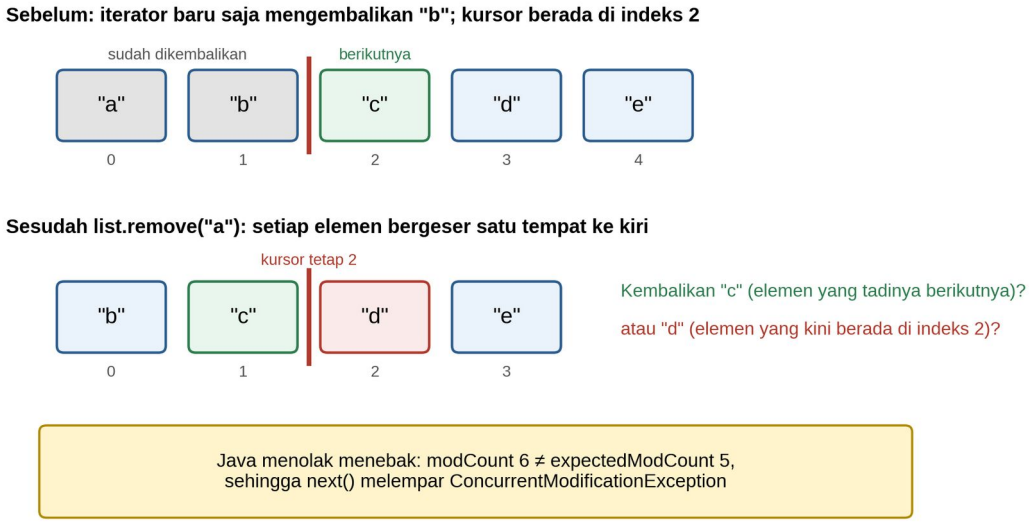
`list(plan)` berhasil setiap kali: `plan` adalah **iterable**, dan setiap pemanggilan `plan.__iter__()` mengembalikan iterator baru.

Sistem tipe Java memaksakan perbedaan itu: for-each menerima `Iterable`, tidak pernah `Iterator`. `for` di Python menerima keduanya, sehingga fungsi yang menelusuri generator dua kali **diam-diam tidak mengerjakan apa-apa pada kali kedua**.

> **WAWASAN KUNCI.** Untuk menelusuri sesuatu lebih dari sekali, peganglah sebuah **iterable** (list, tuple, objek seperti `LoadPlan`); jika ragu, panggil `list(...)` sekali dan simpan list-nya.

`itertools.islice(huge, 2)` mengambil dua elemen secara malas, dan `sum(1 for _ in huge)` menghitung tanpa membangun list. Buku: Subbab 13.4.3.

Mengubah Koleksi Selagi Menelusurinya



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.

Pada Gambar 13.2 iterator baru saja mengembalikan `"b"`; seseorang memanggil `list.remove("a")`. Haruskah `next()` mengembalikan `"c"` atau `"d"`? Tidak ada jawaban yang baik, maka koleksi Java **menolak menebak**.

“Bersamaan” di sini tidak berarti dua thread (itu Bab 14): artinya perubahan terjadi selama iterasi berlangsung. Buku: Subbab 13.5 dan Gambar 13.2.

Bagaimana Iterator Mengetahuinya

`ArrayList` mencacah setiap perubahan **struktural** (`add`, `remove`, tetapi bukan `set`) dalam `modCount`. Iterator baru menyalinnya ke `expectedModCount`; `remove()` milik iterator memperbarui keduanya.

Di awal setiap `next()` iterator memeriksa

$$\text{modCount} = \text{expectedModCount} \tag{13.2}$$

dan melemparkan `ConcurrentModificationException` jika Persamaan (13.2) salah.

Jadi selagi perulangan for-each menelusuri sebuah `List`: `set` **boleh**, sedangkan `add` dan `remove` melalui list **terlarang**.

Buku: Subbab 13.5 dan Persamaan (13.2).

Lompatan Senyap

Kode 13.10 `ConcurrentModification.java`, blok 4: menghapus elemen kedua dari belakang mengakhiri perulangan terlalu dini, tanpa eksepsi.

```
        // 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());
```

```
no exception, but visited only: Ace Luffy Sanji Zoro
```

Pada Kode 13.10: sesudah `next()` mengembalikan `Zoro`, kursor $c=4$; penghapusan menyusutkan list menjadi $n=4$. Menurut Persamaan (13.1), `hasNext()` bertanya $4 \neq 4$: salah. Perulangan **berakhir** sebelum `next()` sempat memeriksa Persamaan (13.2), dan Nami tidak pernah dikunjungi.

Blok 1–3 program yang sama membuktikan aturannya: `set` tidak apa-apa, `add` dan `remove` melemparkan eksepsi pada `next()` tersembunyi berikutnya. Pemeriksaannya hanya "upaya terbaik". Buku: Subbab 13.5 dan Kode 13.7.

Cara yang Benar

Kode 13.11 `ConcurrentModification.java`, blok 5: penghapusan melalui iterator itu sendiri, dan `removeIf`.

```
        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);
```

```
Iterator.remove(): [Luffy, Sanji, Zoro, Nami]
removeIf():        [Luffy, Sanji, Zoro, Nami]
```

Kode 13.11: `remove()` milik iterator memperbarui `expectedModCount`, sehingga semuanya tetap konsisten; `removeIf(predicate)` menyatakan maksud kita dalam satu baris.

Untuk **menambah** selama penelusuran, iterasikan salinannya: `for (String name : new ArrayList<>(myList))`.

*Aturannya bukan "tangkap `ConcurrentModificationException`", melainkan **jangan pernah mengubah ukuran koleksi selagi perulangan for-each menelusurinya**. Buku: Subbab 13.5.*

Python Tidak Memeriksa

Kode 13.12 `concurrent_modification.py`, blok 2 dan 3: menghapus dari list selama perulangan `for` memberi jawaban yang salah, tanpa galat.

```
    # 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)
```

CONTOH. Pada Kode 13.12, hapus bilangan genap dari `[2, 4, 6, 7]`. Indeks 0: hapus 2, list menjadi `[4, 6, 7]`. Indeks 1 kini berisi **6**, jadi **4 terlompati**. Hapus 6: `[4, 7]`. Indeks 2 sudah melewati ujung. Hasilnya `[4, 7]`, bukan `[7]`.

Dictionary dan set lebih ketat: mengubah ukurannya selagi diiterasi membangkitkan `RuntimeError: dictionary changed size during iteration`. *Buku: Subbab 13.5 dan Kode 13.8.*

Cara Aman Mengubah Selagi Menelusuri

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`

Pada Tabel 13.2, `lst[:] = [...]` mengganti isi list **di tempat**, sehingga semua pihak lain yang memegang list itu melihat perubahannya; `list(stock)` adalah salinan kunci-kunci yang diambil sebelum perulangan dimulai.

Buku: Subbab 13.5 dan Tabel 13.2.

Comparable dan Comparator

`Comparable<T>`: kelas mempunyai satu **urutan alami** yang disepakati semua orang (bilangan, string, tanggal), melalui `int compareTo(T other)`.

`Comparator<T>`: objek *terpisah* dengan `int compare(T a, T b)`. Mahasiswa menurut nama untuk daftar hadir, menurut usia untuk tim olahraga, menurut nilai untuk hadiah: satu urutan alami, comparator sebanyak yang diperlukan.

Hanya **tanda** hasilnya yang penting: negatif "yang pertama lebih dulu", nol "setara dalam urutan ini", positif "yang pertama belakangan". Dengan sgn sebagai tanda:

$$\operatorname{sgn}(\operatorname{cmp}(a,b)) = -\operatorname{sgn}(\operatorname{cmp}(b,a)) \text{ untuk semua } a, b \tag{13.3}$$

$$\operatorname{cmp}(a,b) > 0 \text{ dan } \operatorname{cmp}(b,c) > 0 \Rightarrow \operatorname{cmp}(a,c) > 0 \tag{13.4}$$

Persamaan (13.3) adalah antisimetri, Persamaan (13.4) transitivitas; aturan ketiga menyatakan bahwa jika $\operatorname{cmp}(a,b) = 0$ *maka* a *dan* b *dibandingkan dengan cara yang sama terhadap elemen lain mana pun. Buku: Subbab 13.6 dan 13.6.1.*

Jangan Membandingkan dengan Pengurangan

`return s1.getAge() - s2.getAge();` memberi tanda yang benar untuk usia. Tetapi `int` hanya memuat -2^{31} sampai $2^{31}-1$:

$$2^{31}-1-(-1)=2^{31} \overset{\text{luapan}}{\rightarrow} -2^{31}<0 \tag{13.5}$$

Menurut Persamaan (13.5), `int` terbesar dinyatakan *lebih kecil* daripada -1, sehingga Persamaan (13.3) dilanggar.

> **WAWASAN KUNCI.** Obatnya tidak berbiaya apa-apa: `Integer.compare(a, b)`, `Long.compare`, `Double.compare`, atau `Comparator.comparingInt`.

Buku: Subbab 13.6.1.

Comparator Berdasarkan Usia

Kode 13.13 `AgeSort.java`, `AgeComparator` tulisan tangan: `null` lebih dulu, lalu `Integer.compare`.

```
    // 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 13.14 `AgeSort.java`, urutan yang sama dirakit dari bagian-bagian: `comparingInt`, `reversed`, `nullsFirst`.

```
        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);
```

Kode 13.13 menangani kasus-kasus khusus lebih dulu; Kode 13.14 memperlihatkan mengapa kombinator itu ada: tidak seorang pun perlu menulis bagian yang rawan galat dengan tangan. Keluarannya diakhiri `oldest first: [Nami (20), Chopper (15), Usopp (12)]`*. Buku: Subbab 13.6.2.*

Python Mengurutkan dengan Kunci

Kode 13.15 `age_sort.py`: fungsi kunci dengan `attrgetter`, `reverse=True`, dan comparator dua argumen yang diadaptasi dengan `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)
```

Fungsi kunci pada Kode 13.15 melihat *satu* elemen; Python menghitung setiap kunci sekali dan membandingkan kunci-kunci itu dengan `<`. Kunci **tidak mungkin** melanggar Persamaan (13.3) atau Persamaan (13.4), dan lebih cepat.

Kedua pengurutan bersifat **stabil**: kunci yang setara mempertahankan urutannya, sehingga Anda dapat mengurutkan menurut nama depan, lalu secara stabil menurut nama belakang.

`(a > b) - (a < b)` adalah idiom Python untuk tanda sebuah perbandingan, karena `True` dan `False` dihitung sebagai 1 dan 0. Buku: Subbab 13.6.2.

compareTo yang Hanya Tampak Benar

Kode 13.16 `NameOrder.java`, `BrokenName.compareTo`: sebuah `switch` atas -1, 0, dan 1.

```
        @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;
        }
```

Pada Kode 13.16, `"Luffy".compareTo("Garp")` bernilai **5** ('L' − 'G' = 76 − 71), bukan 1. Hampir setiap pasangan jatuh ke `Integer.MIN_VALUE`, "lebih kecil", apa pun urutannya: $a<b$ *dan* $b<a$.

Buku: Subbab 13.6.3.

Pengurutan Hanya Membalik Masukan

```
"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.
```

Timsort milik Java mencari **run** yang menurun tegas lalu membaliknya. Perbandingan yang selalu menjawab “lebih kecil” membuat seluruh larik menjadi satu run, sehingga hasilnya adalah **masukan yang dibalik**, dan berbeda antara Kasus 1 dan Kasus 2.

> **PERINGATAN.** Tanda khas perbandingan yang rusak: **hasilnya bergantung pada urutan masukan**. Pada larik besar, Java bahkan dapat melemparkan `Comparison method violates its general contract!`

Pengurutan Python adalah Timsort yang asli dan berperilaku persis sama dengan perbandingan rusak yang sama. Buku: Subbab 13.6.3.

Field demi Field, dengan Benar

Kode 13.17 `NameOrder.java`: record yang meneruskan tandanya, comparator nama belakang lalu nama depan, dan comparator string tersambung yang rapuh.

```
    // 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);
        }

        @Override
        public String toString() {
            return firstName + " " + lastName.charAt(0) + ".";
        }
    }

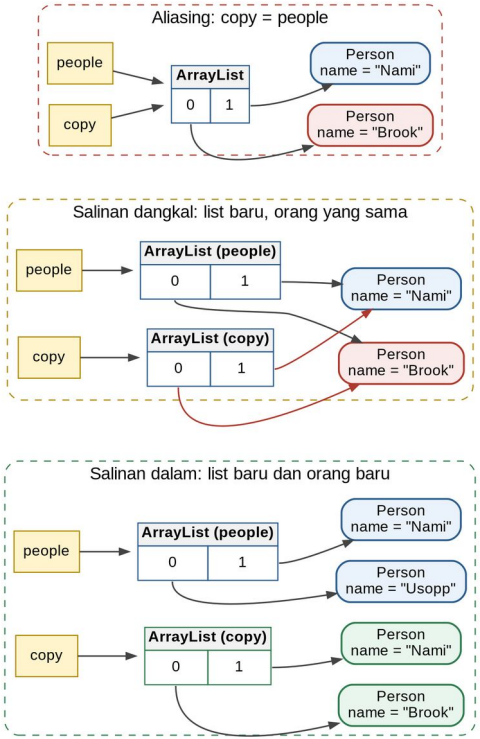
    static final Comparator<Name> LAST_THEN_FIRST =
            Comparator.comparing(Name::lastName).thenComparing(Name::firstName);

    // Merekatkan kedua nama lalu membandingkan string hasil rekatan.
    static final Comparator<Name> CONCATENATED =
            (n1, n2) -> (n1.lastName() + n1.firstName()).compareTo(n2.lastName() + n2.firstName());
```

Pada Kode 13.17, `"Monkey D.Luffy"` melawan `"MonkeyZed"`: spasi (32) bertemu `Z` (90), sehingga penyambungan menaruh "Monkey D." lebih dulu, padahal `"Monkey"` adalah awalan dan harus lebih dulu. Perbandingan field demi field tidak pernah mengalami masalah ini.

Python: `@dataclass(frozen=True, order=True)` membandingkan field sesuai urutan deklarasinya, dan kunci tuple `lambda n: (n.last_name, n.first_name)` adalah padanan `comparing(...).thenComparing(...)`. Buku: Subbab 13.6.3 dan Kode 13.12.

Seberapa Dalam Sebuah Salinan?



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**: dua referensi ke satu objek. **Salinan dangkal**: koleksi baru yang memegang *referensi elemen yang sama*. **Salinan dalam**: koleksi *dan* setiap elemennya disalin, sehingga tidak ada objek mutabel yang dipakai bersama.

Ganti nama Usopp menjadi Brook di salinan pada Gambar 13.3: aslinya menjadi `[Brook, Nami]` untuk aliasing dan salinan dangkal, dan tetap `[Usopp, Nami]` hanya untuk salinan dalam. Buku: Subbab 13.7 dan Gambar 13.3.

Dangkal dan Dalam di Java

Kode 13.18 `CopyDemo.java`: salinan dangkal dan salinan dalam dari list berisi objek `Person`.

```
    // 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;
    }
```

Pada Kode 13.18, `Collections.copy` sesudah konstruktor tidak berbahaya tetapi berlebihan; perlu diperhatikan saat membaca kode lama. Java tidak mempunyai salinan dalam yang umum: setiap kelas memutuskannya sendiri, biasanya dengan konstruktor salinan. Buku: Subbab 13.7.

Tiga Percobaan, Tiga Jawaban

```
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]]
```

Python: `list(people)` (atau `people.copy()`, `people[:]`, `copy.copy`) bersifat dangkal; `copy.deepcopy(people)` menyalin semuanya, termasuk menangani referensi bersama dan siklus.

> **WAWASAN KUNCI.** Dengan elemen yang **imutabel**, salinan dangkal dan salinan dalam tidak dapat dibedakan, dan salinan yang murah sudah cukup.

Buku: Subbab 13.7 dan Kode 13.14.

Imutabilitas dari Nol

> **DEFINISI.** Sebuah objek bersifat **imutabel** jika keadaannya tidak dapat berubah setelah objek itu diciptakan; jika dapat berubah, ia **mutabel**. *Variabel* yang merujuk ke objek imutabel tetap dapat dibuat merujuk ke objek lain.

Imutabel: `String`, `Integer`, `LocalDate`; di Python `int`, `str`, `tuple`, `frozenset`. Mutabel: `ArrayList`, `HashMap`, `StringBuilder`, `list`, `dict`, `set`, dan setiap kelas yang mempunyai setter.

Empat manfaat. 1. **Kunci** yang baik. 2. **Argumen yang aman**: metode tidak dapat merusaknya. 3. Dapat dipakai bersama oleh **thread** tanpa kunci. 4. **Sederhana**: satu keadaan seumur hidup, invarian terjaga dengan sendirinya.

"Mengandalkan objek imutabel semaksimal mungkin diakui luas sebagai strategi yang sehat untuk menciptakan kode yang sederhana dan andal." Rekening bank dan mesin cuci tetap mutabel: keduanya mempunyai identitas dan riwayat hidup. Buku: Subbab 13.8 dan 13.8.1.

Metode yang Mengurutkan List Anda

Kode 13.19 `MedianDemo.java`: median yang mengurutkan list milik pemanggil, dan median yang mengurutkan salinannya.

```
    // 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);
    }
```

```
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
```

Pada Kode 13.19, versi yang ceroboh mengembalikan median yang benar, 44,5, sekaligus merusak data sepekan: tidak ada yang tahu lagi hari mana yang Senin. Tampilan tak-termodifikasi mengubah efek samping tersembunyi itu menjadi `UnsupportedOperationException` *yang nyaring. Python:* `values.sort()` *lawan* `sorted(values)`*. Buku: Subbab 13.8.1.*

final Bukan Berarti Imutabel

Kode 13.20 `ImmutableCollections.java`, blok 2 dan 3: **tampilan** tak-termodifikasi dan **salinan** tak-termodifikasi.

```
        // 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"));
```

```
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
```

*Blok 1 menambahkan `"DC"` ke list `final` tanpa keberatan. Pada Kode 13.20, **tampilan** bersifat hanya-baca tetapi memperlihatkan perubahan di belakangnya; **salinan** adalah potret yang tidak dapat diubah siapa pun. `List.of`, `Set.of`, `Map.of` menolak `null`, dan semuanya bersifat **dangkal**. Buku: Subbab 13.8.2.*

Alat-Alat Imutabilitas

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

Python pada Tabel 13.3: `typing.Final` hanya petunjuk bagi pemeriksa tipe; `MappingProxyType` adalah tampilan hanya-baca sebuah dict; `tuple` dan `frozenset` adalah salinan imutabel dan dapat menjadi kunci dictionary, sedangkan `list` tidak.

Buku: Subbab 13.8.2 dan Tabel 13.3.

Lima Aturan Kelas Imutabel

1. **Tanpa mutator**: tanpa setter, tanpa `add`, tanpa `increment`.
2. **Kelasnya tidak dapat diturunkan** (`final`), sebab subkelas dapat menambahkan keadaan yang mutabel.
3. **Semua field `private final`**, sehingga kompilator menjamin setiap field diisi sekali saja.
4. **Jaga setiap komponen mutabel**: **salinan defensif** saat masuk, dan jangan pernah menyerahkan aslinya saat keluar.
5. **Bangun objeknya sepenuhnya di konstruktor**; jangan biarkan `this` lolos. Untuk "mengubah" sebuah nilai, kembalikan nilai baru: `withX(5)`, `toUpperCase()`, `plusDays(1)`.

Buku: Subbab 13.8.3.

Sebuah Kebocoran, dan Record yang Menyalin

Kode 13.21 `ImmutableClasses.java`, `LeakyCustomer`: ia menyimpan list milik pemanggil dan menyerahkan list miliknya sendiri.

```
    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
        }
    }
```

Kode 13.22 `ImmutableClasses.java`, record `Customer`: konstruktor ringkasnya memeriksa nama dan menyalin list-nya.

```
    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
        }
    }
```

*Kode 13.21 bocor dua kali: `add` belakangan dari pemanggil dan `add` orang asing melalui getter sama-sama masuk ke dalamnya. **Record** seperti pada Kode 13.22 bersifat `final`, dengan field `private final`, aksesor, `equals`, `hashCode`, dan `toString`. Satu-satunya yang tidak dapat dikerjakannya untuk Anda adalah salinan defensif. Buku: Subbab 13.8.3.*

Dataclass Beku

Kode 13.23 `immutable_classes.py`: dataclass beku yang `__post_init__`-nya membuat salinan defensif.

```python
@dataclass(frozen=True)
class Customer:
    name: str
    phones: tuple

    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))
```

Pada Kode 13.23, `frozen=True` membuat setiap penugasan membangkitkan `FrozenInstanceError`; `dataclasses.replace(p, x=5)` adalah pola `withX`; `typing.NamedTuple` adalah tipe nilai imutabel yang lain.

> **PERINGATAN.** Imutabilitas di Python adalah konvensi yang kuat: `object.__setattr__` bekerja dari mana saja. Ia melindungi dari *kekeliruan*, bukan dari perusakan yang disengaja; demikian pula `final` di Java.

Buku: Subbab 13.8.3.

Objek Nilai dan Hashing

DEFINISI. Objek nilai sepenuhnya dijelaskan oleh field-fieldnya: field yang sama berarti objek yang dapat dipertukarkan, dan ia (hampir selalu) imutabel. **Entitas** mempunyai identitas yang bertahan selagi field-fieldnya berubah.

Uang, tanggal, berat, warna: nilai. Pelanggan, mesin, rekening: entitas. Kontrak hash dan pemilihan ember:

$$a.\text{equals}(b) \Rightarrow h(a) = h(b) \tag{13.6}$$

$$i = h(k) \bmod m \tag{13.7}$$

Tabel hash hanya mencari di ember i dari Persamaan (13.7); seandainya Persamaan (13.6) tidak berlaku, dua kunci yang sama dapat jatuh ke ember yang berbeda. Buku: Subbab 13.9.

Kunci yang Berubah

Kode 13.24 `MutableKey.java`: `Spot(1, 2)` yang mutabel disimpan dalam map dan set, lalu field `row`-nya diubah.

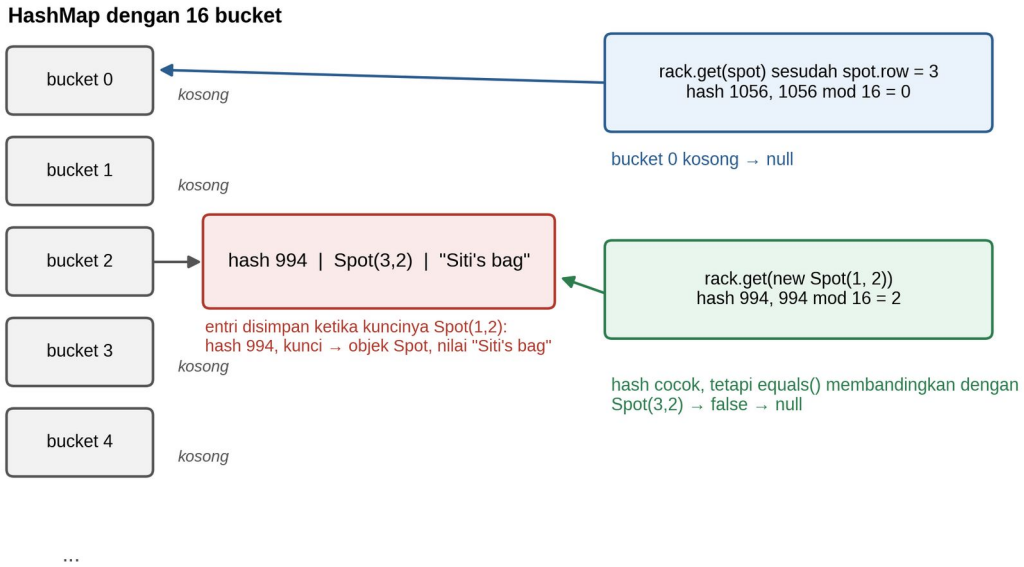
```
Spot spot = new Spot(1, 2);
Map<Spot, String> rack = new HashMap<>();
rack.put(spot, "Siti's bag");
Set<Spot> used = new HashSet<>();
used.add(spot);
System.out.println("hashCode before: " + spot.hashCode());

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());
```

```
hashCode before: 994
hashCode after:  1056
rack.get(spot):             null
rack.get(new Spot(1, 2)):   null
rack.containsKey(spot):     false
```

`Objects.hash(row, col)` *sama dengan* $31 \cdot (31 + \text{row}) + \text{col}$*: 994 sebelumnya, 1056 sesudahnya. Buku: Subbab 13.9.*

Tasnya di Rak, Tak Terjangkau



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

Pada Gambar 13.4, `rack.get(spot)` mencari di ember 0: kosong. `rack.get(new Spot(1, 2))` menemukan ember 2 tetapi gagal pada `equals`. Set kini memuat **dua elemen yang sama**.

Pada keluaran Kode 13.24, entrinya masih tercetak ketika semua ember ditelusuri, tetapi tidak satu pencarian pun dapat menemukannya: kebocoran memori dan tas yang hilang karena satu penugasan. Buku: Subbab 13.9 dan Gambar 13.4.

Obatnya Adalah Imutabilitas

Kode 13.25 `mutable_key.py`, kelas `Shelf`: `frozen` ditambah `__eq__` yang dibangkitkan menghasilkan kunci yang aman dan dapat di-hash.

```
@dataclass(frozen=True)
class Shelf:                                # frozen + eq  =>  kunci yang aman dan dapat di-hash
    row: int
    col: int
```

Kode 13.26 `mutable_key.py`: untuk “memindahkan” tas, keluarkan dari kunci lama dan simpan di bawah kunci baru; list ditolak sebagai kunci.

```
    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)
```

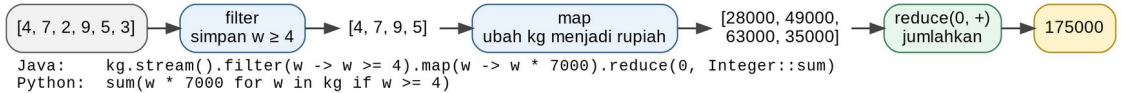
> **WAWASAN KUNCI.** Pakailah sebagai kunci hanya objek yang kesamaannya **tidak dapat berubah**: string, bilangan, tanggal, enum, record, dataclass beku, tuple berisi hal-hal tersebut. Entitas diberi kunci berupa id yang tidak berubah.

Kode 13.25 adalah obatnya; Kode 13.26 memperlihatkan Python melangkah lebih jauh: `list`, `dict`, *dan* `set` *tidak dapat di-hash, sehingga* `{[1, 2]: "bag"}` *langsung gagal dengan* `TypeError`*. Buku: Subbab 13.9.*

Saring, Petakan, Reduksi

"Pendapatan dari muatan minimal 4 kg dengan tarif Rp7.000 per kilogram": **saring** $w \geq 4$, **petakan** $w \mapsto 7000\,w$, **reduksi** dengan $+$ mulai dari 0:

$$\text{reduce}(f, e, [x_1, x_2, \ldots, x_n]) = f(\cdots f(f(e, x_1), x_2) \cdots, x_n) \tag{13.8}$$



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.

Untuk $f = +$ dan $e = 0$, Persamaan (13.8) adalah $x_1 + \cdots + x_n$; untuk $f = \max$, elemen terbesar. Gambar 13.5 adalah pipeline itu dalam satu baris setiap bahasa. Buku: Subbab 13.10.

Stream di Java

Kode 13.27 `StreamsDemo.java`: saring, petakan, dan reduksi sebagai stream, serta kemalasan yang dibuat terlihat dengan `peek`.

```
        // 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));
```

```
loop:   175000
stream: 175000
pipeline built, nothing printed yet
 saw 4 saw 7 -> first heavy load: 7
```

*Stream tidak menyimpan apa pun: sumber, operasi antara, satu operasi terminal. Pada Kode 13.27, `findFirst` berhenti di 7; 2, 9, 5, dan 3 tidak pernah diperiksa. Stream **sekali pakai**; kolektor seperti `groupingBy` dan `joining` mengubahnya kembali menjadi map atau string. Buku: Subbab 13.10.*

Comprehension di Python

Kode 13.28 `streams_demo.py`: perulangan, ekspresi generator, `map`/`filter`/`reduce`, dan kemalasan.

```
    # 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))
```

```
comp:   175000
reduce: 175000
pipeline built, nothing printed yet
 saw 4 saw 7 -> first heavy load: 7
```

*Pada Kode 13.28, kurung siku membangun list secara **bersemangat** (eager); kurung biasa memberi generator yang **malas**. `groupby` memerlukan masukan terurut; `count`, `islice`, dan `accumulate` melengkapi perkakasnya. Kejelasan, bukan mode, yang menentukan pilihan antara perulangan dan pipeline. Buku: Subbab 13.10 dan Tabel 13.4.*

Buku Besar Laundry

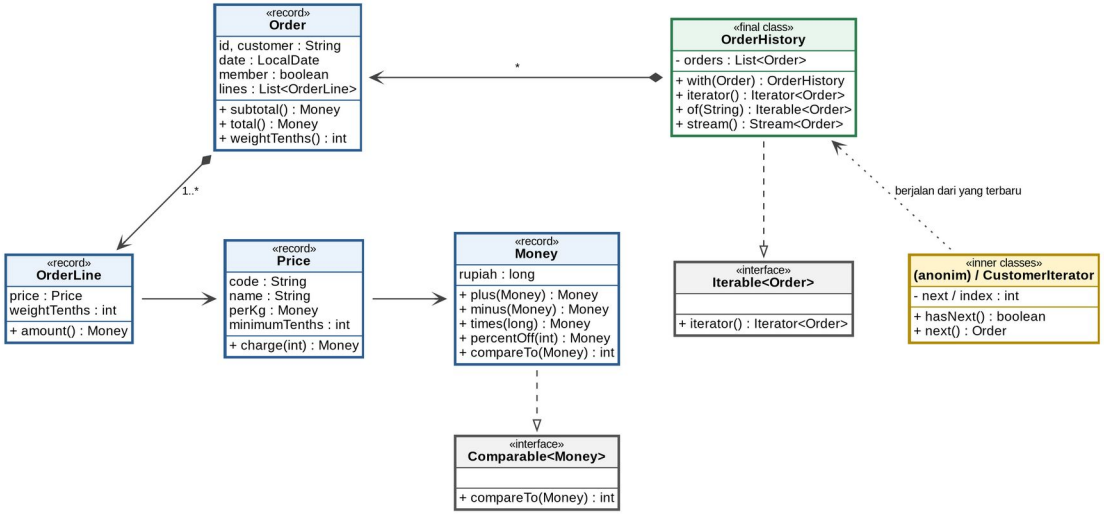
Berat dalam **persepuluhan kilogram**, uang dalam rupiah bulat bertipe `long`, tidak pernah `double`. Berat yang ditagihkan:

$$\hat{w} = 5 \left\lceil \frac{\max(w, w_{\min})}{5} \right\rceil \tag{13.9}$$

CONTOH. 4,2 kg Wash & Fold menurut Persamaan (13.9): $\max(42, 30) = 42$, $\lceil 42/5 \rceil = 9$, $\hat{w} = 45$, harga $7000 \cdot 45/10 = 31.500$. Dalam bilangan bulat, $\lceil a/5 \rceil$ dihitung sebagai `(a + 4) / 5`.

Aturan harganya sama dengan Bab 7: `WF` Rp7.000 (minimum 3 kg), `IR` Rp5.000 (2 kg), `WDI` Rp10.000 (3 kg), anggota mendapat potongan 10%, dibulatkan ke bawah. Buku: Subbab 13.11.

Rancangannya



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.

Pada Gambar 13.6, empat record adalah nilai: `Money`, `Price`, `OrderLine`, `Order`. `OrderHistory` adalah `Iterable<Order>` yang imutabel dan hanya-tambah, dengan dua jenis iterator.

Buku: Subbab 13.11 dan Gambar 13.6.

Money: Sebuah Objek Nilai

Kode 13.29 `LaundryLedger.java`, record `Money`: konstruktor ringkasnya melarang jumlah negatif, dan 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
    }
```

Pada Kode 13.29, `minus` tidak memerlukan pemeriksaan sendiri: konstruktornya menolak -2000*. `addExact` dan `multiplyExact` melemparkan eksepsi alih-alih berputar balik seperti pada Persamaan (13.5). `Money` mengimplementasikan `Comparable<Money>`. Buku: Subbab 13.11.*

Harga dan Pesanan Beku

Kode 13.30 `LaundryLedger.java`, record `Price`: Persamaan (13.9) dalam kode.

```
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);
    }
```

Kode 13.31 `LaundryLedger.java`, record `Order`: salinan defensif tak-termodifikasi atas baris-barisnya, dan total yang dihitung dengan stream kecil.

```
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);
    }

    Money total() {
        return member ? subtotal().percentOff(10) : subtotal();
    }
```

Kode 13.30 menagih satu baris; Kode 13.31 menjamin bahwa pesanan yang sudah dibangun tidak dapat bertambah atau berkurang barisnya: `first.lines().add(...)` *melemparkan eksepsi. Buku: Subbab 13.11.*

Riwayat yang Hanya Bertambah

Kode 13.32 `LaundryLedger.java`, `OrderHistory`: `with(order)` mengembalikan riwayat **baru** yang lebih panjang satu pesanan.

```
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();
    }
```

Kode 13.32 adalah pola `withX` yang diterapkan pada koleksi: laporan, layar kasir, dan cadangan dapat memakai satu riwayat bersama tanpa risiko apa pun. Buku: Subbab 13.11.

Terbaru Dahulu, dan Satu Pelanggan

Kode 13.33 `LaundryLedger.java`: `iterator()` mengembalikan iterator anonim terbaru-dahulu.

```
    @Override
    public Iterator<Order> iterator() {
        return new Iterator<>() {
            private int next = orders.size() - 1;

            @Override
            public boolean hasNext() {
                return next >= 0;
            }

            @Override
            public Order next() {
                if (!hasNext()) {
                    throw new NoSuchElementException();
                }
                return orders.get(next--);
            }
        };
    }
```

Kode 13.33 membuat layar kasir cukup berupa satu perulangan for-each. `of(customer)` *mengembalikan* `Iterable` *yang ditulis sebagai lambda,* `() -> new CustomerIterator(customer)`*, yang memakai* ***pandang-depan****:* `skipOthers()` *melewati pesanan pelanggan lain sehingga* `hasNext()` *cukup satu perbandingan. Tidak ada yang disalin. Buku: Subbab 13.11.*

Apa yang Dicetak Laporan

```
== 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]
```

Pendapatan per layanan (Rp341.500) dihitung sebelum potongan; pendapatan total Rp329.650 sudah dikurangi tiga potongan anggota milik Siti sebesar Rp4.150, Rp3.500, dan Rp4.200. Buku: Subbab 13.11.

Buku Besar di Python

Kode 13.34 `laundry_ledger.py`, `OrderHistory`: sebuah tuple, `with_order`, generator untuk urutan terbaru-dahulu, dan ekspresi generator untuk satu pelanggan.

```
    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)
```

`Money` adalah `@dataclass(frozen=True, order=True)` dengan `__add__`, `__sub__`, `__mul__`, sehingga `sum(..., ZERO)` berjalan: Persamaan (13.8) dengan e = `ZERO`. Seluruh iterator anonim Java pada Kode 13.33 menjadi dua baris pada Kode 13.34.

Keluarannya sama dengan program Java, kecuali cara mencetak list dan nama eksepsi. Buku: Subbab 13.11.

Profitina dalam Praktik

Hasil pemindaian sebagai nilai. Pemindaian yang selesai adalah fakta tentang masa lalu; hasil yang imutabel membuat dasbor analitik historis dapat dipercaya, dan ekspor CSV menuliskan persis apa yang ditampilkan grafik.

Menelusuri riwayat besar secara malas. Tren sebuah merek adalah penelusuran terbaru-dahulu yang tersaring atas riwayat pemindaian yang terus bertambah, seperti iterator `OrderHistory`.

Laporan sebagai pipeline. Rata-rata per pekan dan sinyal yang paling membaik adalah comprehension saring–petakan–reduksi atas hasil yang imutabel.

Data pribadi. Sesuai UU PDP No. 27/2022, ekspor menerima potret yang imutabel. Dasbor menjelaskan; pengguna yang memutuskan.

Buku: Subbab 13.12.

Kesalahan yang Sering Terjadi

1. `next()` **dua kali dalam satu badan perulangan. 2. Mengubah ukuran koleksi di dalam for-each** (Tabel 13.2). **3. Menelusuri iterator Python dua kali**: kosong pada kali kedua, tanpa galat.

4. `compareTo` **yang hanya menangani** -1, 0, 1 (Kode 13.16). **5. Membandingkan dengan pengurangan** (Persamaan (13.5)). **6. Menganggap salinan dangkal itu independen.**

7. Mengira `final` **membuat objek imutabel. 8. Mencampuradukkan tampilan dengan salinan. 9. Tanpa salinan defensif** (`LeakyCustomer`).

10. Mengubah kunci selagi tersimpan (Gambar 13.4). **11. Efek samping di dalam stream atau comprehension. 12. Memakai ulang stream.**

Setiap kesalahan memiliki obatnya di buku, di samping program yang memperlihatkannya. Buku: Subbab 13.13.

Ringkasan — Lima Hal

1. **Iterator menelusuri; iterable menghasilkan iterator.** `hasNext`/`next` (Persamaan (13.1)), `iter`/`next`/`StopIteration`; generator menuliskan iterator untuk Anda, secara malas. Iterator hanya dipakai sekali.
2. **Jangan mengubah ukuran koleksi selagi menelusurinya.** Java mendeteksinya dengan `modCount` (Persamaan (13.2)) sebagai upaya terbaik; list Python melompat tanpa suara.
3. **Perbandingan harus antisimetris dan transitif**; hanya tandanya yang penting; jangan mengurangkan. Python mengurutkan dengan kunci; kedua pengurutan stabil.
4. **Objek imutabel** adalah kunci yang aman, argumen yang aman, aman dipakai bersama, dan menjaga invariannya. `final` mengunci referensi; record dan dataclass beku memerlukan salinan defensif atas bagian yang mutabel.
5. **Objek nilai** di-hash menurut field-fieldnya (Persamaan (13.6)); kunci tidak boleh berubah. **Stream dan comprehension** adalah pipeline saring–petakan–reduksi yang malas dan sekali pakai.

Buku: Subbab 13.14.

Pertemuan Berikutnya

Bab 14 — Thread, Race, Deadlock, dan Livelock. Beberapa potong kode berjalan pada waktu yang sama, dan tiga cara hal itu menjadi kacau.

Manfaat ketiga imutabilitas hari ini, *dipakai bersama oleh thread tanpa kunci*, berubah menjadi keharusan: keadaan bersama yang mutabel adalah sumber race.

Pekerjaan rumah. Soal Latihan 13.1–13.13; mulailah dengan 13.4 (meramalkan `ConcurrentModificationException` dengan Persamaan (13.1) dan Persamaan (13.2)) dan 13.8 (menemukan kebocoran pada `Voucher`).

Penyelesaiannya ada di Subbab 13.16; cobalah setiap soal sebelum membaca penyelesaiannya. Buku: Subbab 13.15 dan 13.16.

Kuliah 14 — Thread, Race, Deadlock, dan Livelock

Pemrograman Berorientasi Objek (EF234302)

Bab 14 buku ini dalam bentuk ringkas, untuk dipakai di kelas.

Buku: Irfan Subakti, *Pemrograman Berorientasi Objek dengan Java dan Python*.

Pengajar: Irfan Subakti — yifana@gmail.com

Program konkuren dapat mencetak hal yang berbeda setiap kali dijalankan; hari ini kita memperlihatkan hasil jalan yang sungguhan dan mengatakan dengan tepat apa yang berubah-ubah dan apa yang tidak boleh berubah. Buku: Bab 14, paragraf pembuka.

Agenda

1. **Thread.** Apa itu thread, konkurensi lawan paralelisme, menciptakan dan menghentikan thread di Java dan Python, serta kehidupan sebuah thread.
2. **Kecepatan.** Hukum Amdahl, dan Global Interpreter Lock Python: kapan thread membantu dan kapan tidak.
3. **Race.** Pencacah bersama yang kehilangan pembaruan, mengapa `c++` bukan satu langkah, dan obatnya: `synchronized`, kunci, atomik.
4. **Menunggu.** `wait`/`notifyAll`, `Condition`, dan antrean pemblokir untuk produsen dan konsumen.
5. **Keberlangsungan.** Deadlock, keempat syaratnya dan pencegahannya; livelock dan starvation; serta laundry dengan tiga thread mesin.

Halaman-halaman ini mengikuti urutan buku; soal latihan di akhir bab adalah pekerjaan rumahnya. Buku: Subbab 14.2 sampai 14.14.

Di Mana Kita Berhenti

Setiap program sejauh ini mengerjakan **satu hal pada satu waktu**. Bab 10 adalah pengecualian yang diam-diam: jendela Swing atau Tkinter tetap menggambar ulang dirinya selagi handler kita berjalan, karena thread *milik pihak lain* menjalankan perulangan peristiwanya.

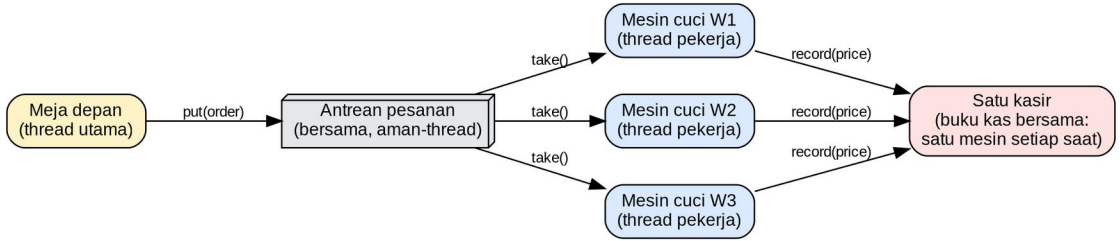
Bab 13 membuat objek **imutabel**, yang ternyata termasuk pertahanan terbaik kita hari ini.

Profitina Laundry membeli dua mesin cuci lagi, tetapi perangkat lunaknya masih memulai satu cucian, menunggu, mencatat pembayaran, baru kemudian memulai cucian berikutnya: dua mesin menganggur.

> **WAWASAN KUNCI.** Thread mudah *dijalankan* dan sulit *dikoordinasikan*. Hampir setiap bug konkurensi berasal dari satu keadaan: dua thread menyentuh data yang sama, dan sedikitnya satu di antaranya mengubahnya.

Buku: Subbab 14.1.

Laundry Adalah Sebuah Komputer



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.

Pada Gambar 14.1, ketiga mesin mencuci **serentak**; kasir yang hanya satu mencatat **satu pembayaran pada satu waktu**. Gambar ini juga, nyaris kata demi kata, program pada Subbab 14.14.

Buku: Subbab 14.2 dan Gambar 14.1.

Proses dan Thread

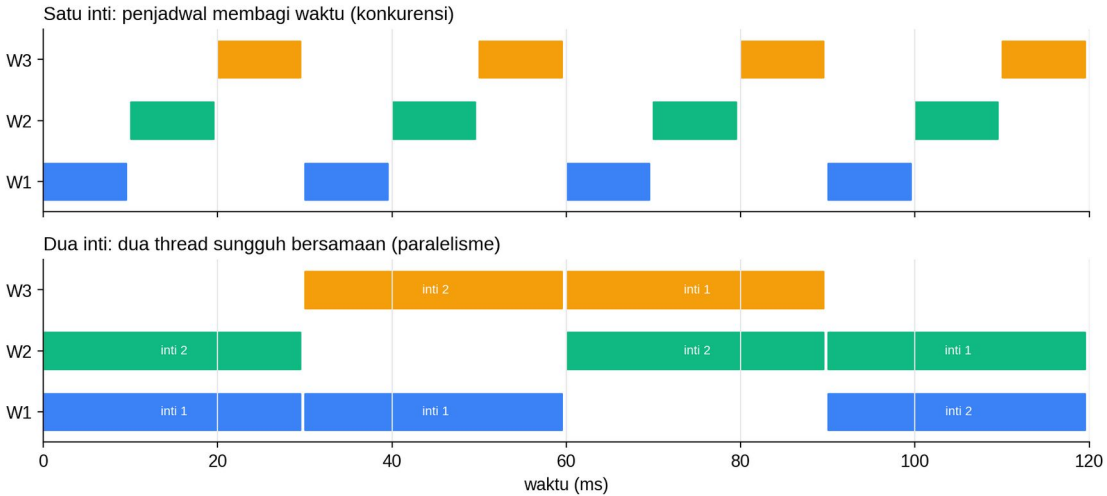
DEFINISI. Sebuah **proses** adalah program yang sedang berjalan, dengan memori sendiri, berkas terbuka sendiri, dan paling sedikit satu thread. Sebuah **thread** adalah satu urutan instruksi mandiri di dalam proses. Setiap thread memiliki *call stack* sendiri; semua thread dalam satu proses berbagi *heap* yang sama.

Variabel lokal sebuah mesin (pesanan mana, berapa kilogram) ada di stack-nya; keranjang pesanan dan buku kasir adalah objek bersama di heap. Berbagi itulah yang membuat thread **berguna** sekaligus **berbahaya**.

Mengapa thread? Daya tanggap (jendela yang tidak membeku), menunggu secara paralel (satu thread per klien), memakai setiap inti, dan model yang lebih sederhana (satu perulangan per kegiatan).

Setiap program mempunyai sedikitnya satu thread: `main` *di Java,* `MainThread` *di Python. Buku: Subbab 14.2 dan 14.2.2.*

Konkurensi lawan Paralelisme



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.

Bukan Anda yang menentukan urutannya: penjadwal yang menentukan, berbeda pada setiap jalan. **Thread dapat dihentikan sejenak di mana saja**, bahkan di tengah `count++`.

Pada Gambar 14.2, konkurensi berarti beberapa tugas sedang berlangsung; paralelisme berarti beberapa tugas dieksekusi pada saat yang sama di inti yang berbeda. Buku: Subbab 14.2.1 dan Gambar 14.2.

Induk yang Tak Dapat Merebut Kembali Kendali

```
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++));
        }
    }
}
Child one = new Child("Luffy");
one.begin();
Child two = new Child("Ace");
two.begin();
```

`one.begin()` tidak pernah kembali: satu-satunya thread milik induk sibuk di dalam perulangan tak berujung milik si anak. Anak kedua tidak pernah dibuat, dan anak pertama tidak pernah dapat dihentikan.

Kita ingin `begin` memulai hitungannya di tempat lain dan segera mengembalikan kendali. Itulah tepatnya yang dikerjakan sebuah thread. Buku: Subbab 14.3.

Anak sebagai Runnable

Kode 14.1 `ParentChild.java`, `Child.run()`: mencetak, tidur selama waktu acak, dan berakhir ketika diinterupsi.

```
@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");
    }
}
```

Pada Kode 14.1, `Thread.sleep` melemparkan `InterruptedException` ketika thread lain memanggil `interrupt()` pada thread yang sedang tidur; eksepsi itu keluar dari perulangan dan `run()` kembali. Buku: Subbab 14.3.1.

Induk Memulai, Tidur, Menginterupsi

Kode 14.2 `ParentChild.java`, `main`: `start()` langsung kembali; `interrupt()` meminta, `join()` menunggu.

```
        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();
        }
        threadOne.join();  // tunggu sampai setiap anak benar-benar selesai
        threadTwo.join();
        System.out.println("Parent ended");
```

Kode 14.2: kita tidak pernah memanggil `run()` sendiri; `start()` meminta JVM membuat thread baru, dan thread itulah yang memanggil `run()`. Blok `finally` meminta anak-anak berhenti bahkan bila tidur induk terpotong. Buku: Subbab 14.3.1.

Satu Jalan yang Sungguhan

```
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
```

Baris-baris Luffy dan Ace **berselang-seling**; tidur-tidur pertama Ace yang singkat membuatnya menghitung sampai 5 sebelum Luffy bangun. `Parent ended` muncul terakhir karena `join()`.

> **PERINGATAN.** Interupsi adalah *permintaan*, bukan perintah: ia memasang sebuah bendera dan membangunkan thread yang tidur. `Thread.stop()` yang sudah usang meninggalkan objek dalam keadaan setengah berubah.

Seed menetapkan waktu tidurnya; yang berubah-ubah hanyalah pewaktuan halus di sekitar satu milidetik. Polanya sama pada setiap jalan; rinciannya tidak. Buku: Subbab 14.3.1.

Python: Event sebagai Ganti Interupsi

Kode 14.3 `parent_child.py`, `Child.run()`: `stop.wait(timeout)` tidur seperti `time.sleep`, tetapi bangun lebih awal ketika induk memanggil `stop.set()`.

```
    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
```

Thread Python tidak dapat diinterupsi. Pada Kode 14.3, sebuah `threading.Event` bersama menggantikan `interrupt()`: `set()` menaikkan bendera, `wait(t)` tidur sampai bendera naik atau `t` berlalu.

Berikan metode terikatnya, `target=child.run`, jangan pernah `target=child.run()`, yang akan menjalankan perulangan tanpa akhir itu di thread induk. Buku: Subbab 14.3.1.

Jebakan Visibilitas

Kode 14.4 `StopFlag.java`, paruh pertama: sebuah thread berputar pada bendera `boolean` biasa.

```
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());
```

Kode 14.4 mencetak `still running 1 s later? true`: pemutar itu **tidak pernah berhenti**. Kompilator JIT boleh membaca `plainStop` sekali, menyimpannya di register, dan mengubah perulangannya menjadi `while (true)`.

Perulangan yang sama pada bendera `volatile` langsung berhenti: setiap pembacaan field volatile melihat operasi tulis terakhir. Buku: Subbab 14.3.2.

Visibilitas dan Happens-Before

DEFINISI. Masalah **visibilitas** terjadi ketika satu thread mengubah variabel bersama dan thread lain tidak melihat perubahan itu, atau melihatnya terlambat. Di Java, operasi tulis dijamin terlihat hanya melalui relasi *happens-before*: operasi tulis dan baca `volatile`, kunci yang sama, atau `start`/`join`.

Di CPython, atribut biasa *memang* menjadi terlihat (karena GIL), tetapi itu fakta tentang interpreter hari ini, bukan janji bahasa; lagi pula perulangan yang berputar menghabiskan satu inti.

Idiom. Java: bendera `volatile`, atau interupsi. Python: `threading.Event` dengan `set()`, `is_set()`, dan `wait(timeout)`.

Buku: Subbab 14.3.2.

Empat Cara Menciptakan Thread

1. **Menurunkan** `Thread` dan menimpa `run()`: gaya tertua; kelasnya tidak dapat menurunkan kelas lain, dan mencampur *apa* dengan *bagaimana*.
2. **Mengimplementasikan** `Runnable` dan menyerahkannya ke `new Thread(r, name)`: *apa* yang dikerjakan terpisah dari *bagaimana* menjalankannya. Rancangan yang dianjurkan.
3. **Lambda**: `Runnable` adalah antarmuka fungsional, sehingga `new Thread(() -> ..., "iron-1")` berjalan.
4. `ExecutorService`: kumpulan thread pekerja yang dapat dipakai ulang dan diberi makan tugas.

Java 21 juga menawarkan thread virtual, `Thread.ofVirtual().start(task)`; race, kunci, dan deadlock bekerja dengan cara yang sama pada thread itu. Buku: Subbab 14.4.

Kumpulan Thread dan Future-nya

Kode 14.5 `CreateThreads.java`, cara 4: kumpulan tetap berisi dua thread menghitung empat harga sebagai tugas `Callable`.

```
        // 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);
```

```
pool finished: true
total of 4 loads = Rp147000
```

Pada Kode 14.5, `submit` mengembalikan sebuah `Future`, tiket untuk suatu hasil; `get()` menunggunya. $21 \times 7000 = 147.000$. *`shutdown()` tidak menerima tugas baru dan membiarkan tugas lama selesai. Buku: Subbab 14.4.*

run() Bukan start()

Kode 14.6 `CreateThreads.java`: memanggil `run()` mengeksekusi tugas di thread saat ini; hanya `start()` yang menciptakan thread baru.

```
        // 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();
```

```
run() executed by: main
run() executed by: helper
```

Kode 14.6 memperlihatkan kesalahan pemula yang paling umum: jawabannya benar, hanya lambat dan tanpa konkurensi sama sekali, itulah sebabnya kesalahan ini bertahan lama.

> **PERINGATAN.** Sebuah thread hanya dapat dimulai sekali: `start()` kedua melemparkan `IllegalThreadStateException`. Untuk menjalankan pekerjaan itu lagi, buat `Thread` baru atau serahkan tugasnya lagi.

Buku: Subbab 14.4.

Kosakata Thread

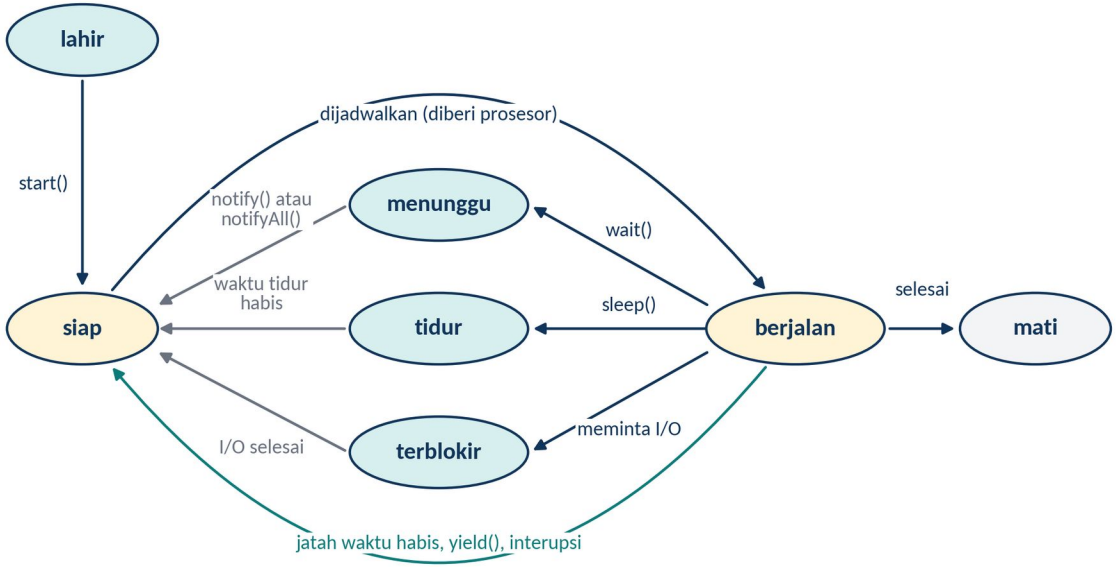
Tabel 14.1 Kosakata dasar thread di Java dan Python.

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()`

Python pada Tabel 14.1: fungsi target adalah `Runnable`-nya; `ThreadPoolExecutor` di dalam blok `with` dimatikan dan ditunggu. Di dalam perulangan, tulis `lambda kg=kg: kg * 7000`, atau setiap lambda melihat `kg` yang terakhir.

*Program berakhir ketika thread **non-daemon** terakhirnya berakhir; daemon adalah pembantu latar belakang yang boleh ditinggalkan. Buku: Subbab 14.4 dan Tabel 14.1.*

Kehidupan Sebuah Thread

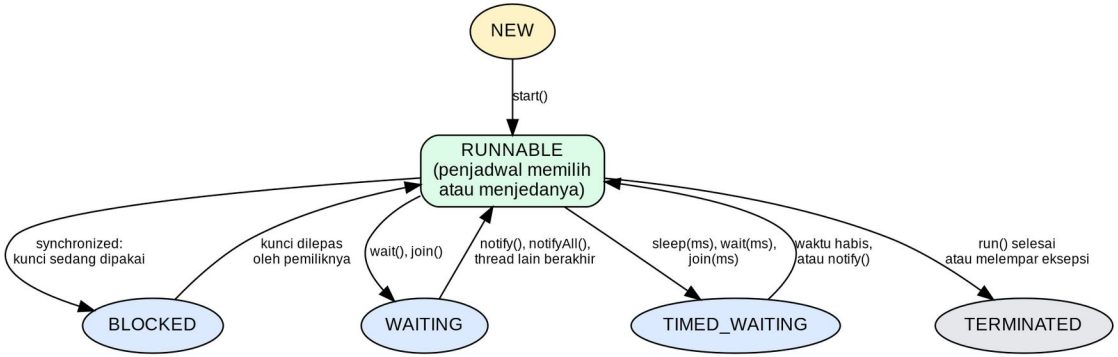


Gambar 14.3 Siklus hidup klasik sebuah thread Java. Thread lahir, menjadi siap ketika dimulai, berjalan ketika penjadwal memberinya prosesor, dan dapat meninggalkan keadaan berjalan untuk menunggu, tidur, atau terblokir oleh masukan dan keluaran sebelum siap lagi atau mati.

Gambar 14.3 membedakan *siap* (dapat berjalan) dari *berjalan* (sedang di sebuah inti). Java tidak menampakkan perbedaan itu: penjadwal berpindah ribuan kali per detik.

Buku: Subbab 14.5 dan Gambar 14.3.

Enam Nilai Thread.State



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.

Pada Gambar 14.4: **`NEW`** belum dimulai; **`RUNNABLE`** berjalan atau siap; **`BLOCKED`** menunggu kunci; **`WAITING`** menunggu tanpa batas (`wait()`, `join()`); **`TIMED_WAITING`** dengan batas (`sleep(ms)`); **`TERMINATED`** selesai untuk selamanya.

Buku: Subbab 14.5 dan Gambar 14.4.

Menangkap Thread dalam BLOCKED

Kode 14.7 `ThreadStates.java`: `main` memegang kuncinya, sehingga thread `blocked` tidak dapat masuk dan berada dalam keadaan `BLOCKED`.

```
        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();
        }
```

```
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
```

Pengamatan lain pada Kode 14.7 diatur dengan cara yang sama, dengan `sleep(100)` *singkat di* `main`*. Python tidak mempunyai enumerasi keadaan:* `is_alive()`*,* `ident`*,* `Lock.locked()`*, dan* `threading.enumerate()`*. Buku: Subbab 14.5.*

Apakah Thread Mempercepat?

Kode 14.8 `SpeedupDemo.java`: bilangan prima di bawah tiga juta dihitung oleh dua thread, satu untuk setiap paruh.

```
        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();
```

```
primes below 3000000 : 216816 and 216816
one thread           :  623 ms
two threads          :  602 ms
```

Hukum Amdahl: bila pecahan f dari pekerjaan dapat dibagi ke p inti,

$$S(p) = \frac{1}{(1-f) + \frac{f}{p}} \tag{14.1}$$

Dengan $f = 0,9$ dan $p = 4$: $S(4) = 1/0,325 \approx 3,08$, dan Persamaan (14.1) tidak pernah melampaui $1/(1-f) = 10$. Kode 14.8 dijalankan pada mesin yang sibuk; hasil 216.816 tidak pernah berubah. Buku: Subbab 14.6.

Global Interpreter Lock Python

DEFINISI. GIL adalah sebuah kunci di dalam CPython. Thread harus memegangnya untuk mengeksekusi *bytecode* Python, dan hanya satu thread yang dapat memegangnya pada satu waktu; kunci ini dilepas secara teratur (setiap 5 ms) dan setiap kali sebuah thread terblokir.

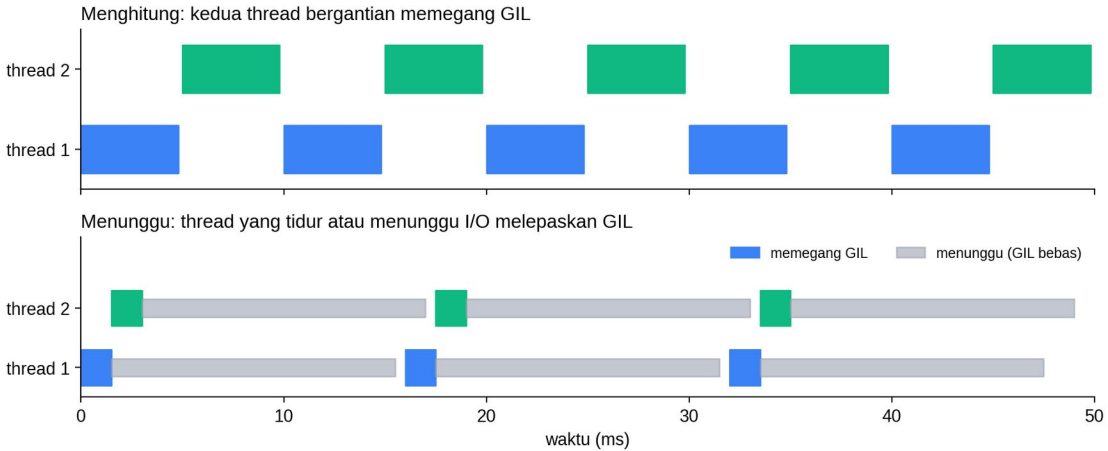
Kode 14.9 `gil_demo.py`: menghitung bilangan prima dalam thread dan dalam proses, serta empat penantian setengah detik.

```
    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)))
    with ThreadPoolExecutor(4) as pool:
        timed("four 0.5 s waits, four threads", lambda: sum(pool.map(wait_for_device, waits)))
```

```
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
```

Pada Kode 14.9, empat penantian turun dari 2,00 s menjadi 0,50 s; dua thread yang menghitung hampir tidak mendapat apa-apa. Buku: Subbab 14.6 dan 14.6.1.

Kapan Thread Python Membantu



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.

Pada Gambar 14.5: pekerjaan **terikat I/O**, thread membantu. Pekerjaan **terikat CPU** dalam Python murni, pakailah proses (`ProcessPoolExecutor`). Pustaka C yang melepas GIL, thread membantu.

*GIL **tidak** membuat kode Anda aman-thread. Python 3.13 menambahkan build bebas-thread yang masih eksperimental (PEP 703), yang didukung resmi di 3.14 (PEP 779). Buku: Subbab 14.6.1 dan Gambar 14.5.*

Tiga Anak, Satu Pencacah

Kode 14.10 `RaceDemo.java`, `Child.run()`: sesudah aba-aba mulai bersama, naikkan (dan mungkin turunkan) pencacah bersama sejuta kali.

```
        @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();
                }
            }
        }
```

```
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
```

Pada Kode 14.10, pencacah yang "harus 3.000.000" kehilangan hampir dua juta kenaikan pada putaran 3, dan sempurna pada putaran 4 dan 5. Tanpa eksepsi, tanpa peringatan. Buku: Subbab 14.7.

Race Condition

DEFINISI. Race condition adalah bug yang membuat hasil program bergantung pada pewaktuan relatif thread-threadnya. Jenis yang paling umum adalah **pembaruan yang hilang**: dua thread membaca nilai lama yang sama, dan operasi tulis kedua menimpa yang pertama.

```
  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
```

`getfield` **membaca**, `iconst_1` dan `iadd` **menambah** pada salinan pribadi, `putfield` **menulis**. Penjadwal boleh menghentikan sebuah thread di antara dua langkah mana pun.

Yang berubah-ubah: banyaknya pembaruan yang hilang. Yang tidak pernah berubah: pencacah yang hanya dinaikkan hanya dapat menjadi terlalu kecil. Buku: Subbab 14.7 dan 14.7.1.

Pembaruan yang Hilang, Langkah demi Langkah



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.

Pada Gambar 14.6: Anak 1 membaca 0; Anak 2 membaca 0 dan menulis 1; Anak 1 menulis angka 1 miliknya sendiri di atasnya. Dua kenaikan, dan pencacah menunjukkan 1. Setiap thread mengerjakan persis apa yang tertulis di kodenya; kesalahannya ada pada **selang-selingnya**.

Buku: Subbab 14.7.1 dan Gambar 14.6.

Menghitung Selang-Seling

Sebuah thread dengan m langkah dan thread lain dengan n langkah dapat berselang-seling dalam

$$\binom{m+n}{m} = \frac{(m+n)!}{m!\,n!} \tag{14.2}$$

cara. Dua kenaikan, $m = n = 3$: 20 selang-seling, hanya 2 yang benar.

Dengan T thread yang masing-masing melakukan N kenaikan, pembaruan yang hilang adalah

$$L = T \cdot N - c_{\text{end}} \tag{14.3}$$

Putaran 1: $L = 3.000.000 - 2.423.058 = 576.942$.

> **PERINGATAN.** Pencacah pada Kode 14.10 bertipe `volatile`, dan tetap kehilangan pembaruan: **`volatile` tidak membuat `c++` aman.**

Menurut Persamaan (14.2), pengujian dapat menunjukkan bahwa sebuah race ada; ia tidak pernah dapat menunjukkan bahwa race itu tidak ada. Persamaan (14.3) adalah yang dicetak program sebagai `lost`. Buku: Subbab 14.7.1.

Race yang Sama di Python

Kode 14.11 `race_demo.py`, `SlowCounter`: tiga langkah yang sama, dengan `time.sleep(0)` di antara operasi baca dan operasi tulis.

```
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
```

```
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
```

Pada Kode 14.11, `self.c += 1` biasa kebetulan tidak kehilangan apa pun di CPython 3.11, tetapi bytecode-nya tetap `LOAD_ATTR`, `BINARY_OP`, `STORE_ATTR`; `sleep(0)` hanya membuka lebar jendela bug yang lama. Buku: Subbab 14.7.2.

Saling Pengecualian

> **DEFINISI. Kunci** (*mutex*) adalah objek yang paling banyak dapat dipegang satu thread pada satu waktu. Sebuah thread **memperoleh** kunci sebelum **bagian kritis** dan **melepasnya** sesudahnya; thread yang mendapati kunci sedang dipegang akan menunggu.

```
public synchronized void increment() {
    c++;
}
synchronized (counter) {
    counter.increment();
}
```

Setiap objek Java mempunyai kunci intrinsik. **Setiap akses ke data bersama yang diubah suatu thread harus memegang kunci yang sama**, termasuk pembacanya.

Metode synchronized mengunci `this` dan melepasnya bahkan ketika melemparkan eksepsi; blok synchronized mengunci objek apa pun yang Anda sebut. Buku: Subbab 14.8.

Kunci Eksplisit dan Atomik

Kode 14.12 `SafeCounters.java`, obat 3: sebuah `ReentrantLock`, selalu dilepas di `finally`.

```
    // Obat 3: kunci eksplisit, dilepas di finally.
    static class LockCounter implements Counter {
        private final ReentrantLock lock = new ReentrantLock();
        private int c = 0;

        public void increment() {
            lock.lock();
            try {
                c++;
            } finally {
                lock.unlock();
            }
        }
```

Kode 14.13 `SafeCounters.java`, obat 4: sebuah `AtomicInteger`; perangkat keras mengerjakan baca-tambah-tulis dalam satu langkah.

```
    // 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();
        }
```

Kode 14.12: `tryLock(time, unit)` *dapat menyerah, dan* `new ReentrantLock(true)` *bersifat adil. Kode 14.13: perulangan compare-and-swap, tanpa kunci sama sekali. Buku: Subbab 14.8.*

Lima Obat, Satu Jawaban

```
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)
```

DEFINISI. Sebuah operasi bersifat **atomik** bila tidak ada thread lain yang pernah dapat melihatnya setengah jadi. `c++` tidak atomik; `incrementAndGet()` atomik; urutan pernyataan apa pun menjadi atomik bagi thread yang memakai kunci yang sama.

Kunci juga membereskan **visibilitas**: melepasnya happens-before setiap perolehan berikutnya. Kunci Java bersifat **reentrant**; `Lock` di Python tidak, `RLock` ya.

Waktu-waktunya berubah-ubah dan bukan tolok ukur; total 3.000.000 tidak pernah berubah. `LongAdder` *menyebar hitungan ke beberapa sel untuk perebutan yang padat. Buku: Subbab 14.8 dan 14.8.1.*

Kunci di Python

Kode 14.14 `safe_counters.py`, `LockedCounter`: `with self._lock:` membuat baca-ubah-tulis yang lambat itu benar.

```
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
```

```
LockedCounter       : should be 6000, is 6000  (0.42 s)
ExplicitLockCounter : should be 6000, is 6000  (0.42 s)
```

Pada Kode 14.14, `sleep(0)` tetap menyerahkan GIL di tengah pembaruan, tetapi thread berikutnya mendapati kuncinya sedang dipegang lalu menunggu. `with` sama dengan `acquire`, `try`, `finally: release()`; utamakan `with`. Buku: Subbab 14.8.2.

Kotak Perkakas Sinkronisasi

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`

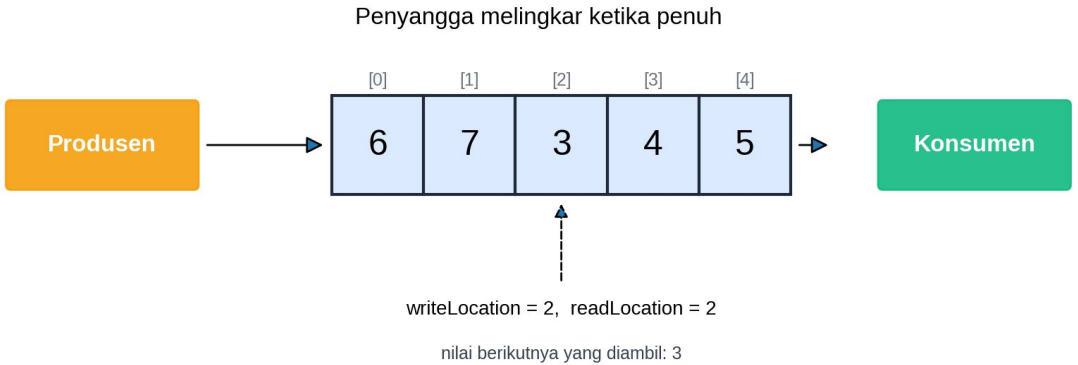
WAWASAN KUNCI. Kunci terbaik adalah kunci yang tidak Anda perlukan: data yang tidak pernah dibagi tidak dapat mengalami race, dan data imutabel (Bab 13) tidak memerlukan kunci. Kuncilah yang dibagi *dan* diubah, dan hanya itu.

Tabel 14.2 menghimpun setiap perkakas bab ini. Buku: Subbab 14.8.2 dan Tabel 14.2.

Produsen, Konsumen, dan Penyangga Melingkar

Produsen menunggu selama penyangga penuh; konsumen menunggu selama penyangga kosong. Lima sel, dipakai melingkar:

$$w \leftarrow (w+1) \bmod n, r \leftarrow (r+1) \bmod n \tag{14.4}$$



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.

Ketika kedua indeks pada Persamaan (14.4) bertemu, penyangga penuh atau kosong; bendera `writeable` dan `readable` menyatakan yang mana. Gambar 14.7 memperlihatkan keadaan penuh. Buku: Subbab 14.9 dan Gambar 14.7.

wait() dan notifyAll()

Kode 14.15 `ProCon.java`, `setSharedInt`: menunggu selama tidak ada sel kosong, menulis, memajukan indeks, dan membangunkan konsumen.

```
        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
        }
```

*`wait()` melepas kunci lalu tidur; `notifyAll()` membangunkan para penunggu, yang memperoleh kembali kuncinya sebelum `wait()` kembali. Kunci ditambah ruang tunggu adalah sebuah **monitor** (Hoare). Buku: Subbab 14.9 dan Kode 14.15.*

while, Bukan if

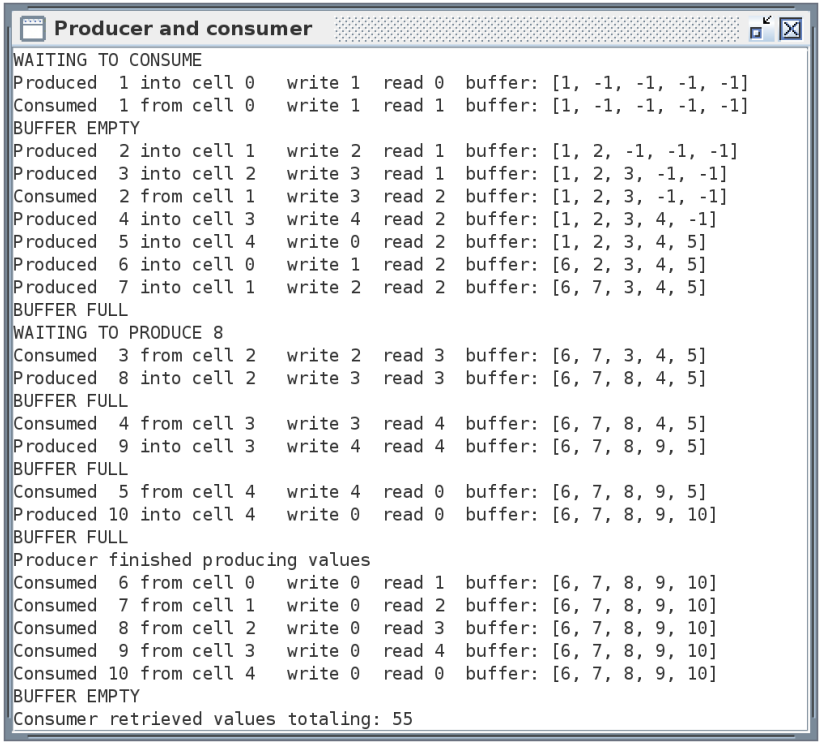
```
Produced  5 into cell 4   write 0  read 2  buffer: [1, 2, 3, 4, 5]
Produced  6 into cell 0   write 1  read 2  buffer: [6, 2, 3, 4, 5]
Produced  7 into cell 1   write 2  read 2  buffer: [6, 7, 3, 4, 5]
BUFFER FULL
WAITING TO PRODUCE 8
Consumed  3 from cell 2   write 2  read 3  buffer: [6, 7, 3, 4, 5]
Produced  8 into cell 2   write 3  read 3  buffer: [6, 7, 8, 4, 5]
```

`while (!writeable) wait();`: thread yang dibangunkan harus menguji lagi; thread lain mungkin sudah mengambil selnya, dan Java mengizinkan bangun palsu (*spurious wake-up*).

`wait()` **melepas kunci**, sehingga konsumen dapat memasuki `getSharedInt` selagi produsen menunggu. `notifyAll()`, bukan `notify()`, agar thread yang tepat tidak pernah dibiarkan tertidur.

Pada keluaran Kode 14.15, nilai-nilai dikonsumsi dari 1 sampai 10, masing-masing sekali, dan totalnya 55 pada setiap jalan. Buku: Subbab 14.9.

Pertukaran di Dalam Jendela



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.

Gambar 14.8: pesan-pesan yang sama, perputaran yang sama dari sel 4 ke sel 0, dan total yang sama, 55.

Buku: Subbab 14.9 dan Gambar 14.8.

Python: threading.Condition

Kode 14.16 `procon.py`, `set_shared_int`: `with self.cond:` mengambil kunci; `cond.wait()` melepasnya dan menunggu; `notify_all()` membangunkan para penunggu.

```
    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
```

Kode 14.16 adalah terjemahan baris demi baris. Java juga menawarkan gaya eksplisit yang sama: `lock.newCondition()` dengan `await()` dan `signalAll()`. Buku: Subbab 14.9.

Antrean Pemblokir yang Menunggu untuk Kita

Kode 14.17 `QueueDemo.java`, produsen: `put` menunggu selama `ArrayBlockingQueue` lima sel itu penuh.

```
        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();
            }
        });
```

Kode 14.18 `QueueDemo.java`, konsumen: `take` menunggu selama antrean kosong, dan sebuah **pil racun** mengakhiri perulangan.

```
            try {
                while (true) {
                    int value = buffer.take();  // menunggu selama antrean kosong
                    if (value == done) {
                        break;
                    }
                    sum += value;
                    count++;
                }
```

Kode 14.17 dan Kode 14.18 mencetak `Consumer took 10 values totaling: 55` *pada setiap jalan. Sesudah menangkap* `InterruptedException`*, pulihkan benderanya dengan* `Thread.currentThread().interrupt()`*. Buku: Subbab 14.10.*

queue.Queue di Python

Kode 14.19 `queue_demo.py`: `put` dan `get` menunggu pada `queue.Queue(maxsize=5)` yang penuh atau kosong; `None` adalah pil racunnya.

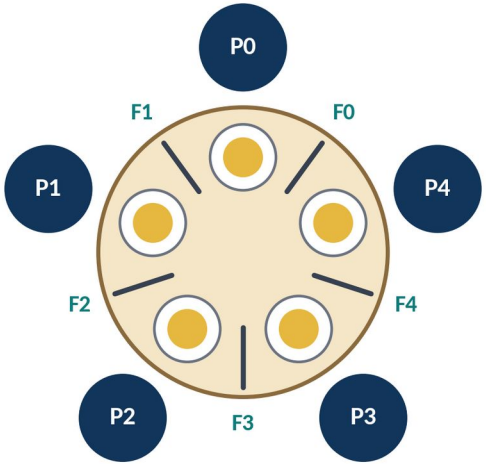
```
    def produce():
        for i in range(1, 11):
            buffer.put(i)      # menunggu selama antrean penuh
        buffer.put(DONE)

    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}")
```

WAWASAN KUNCI. Antrean pemblokir menggantikan *data bersama* (yang memerlukan kunci) dengan *pesan* (yang menjadi milik penerimanya saja). Inilah perkakas konkurensi yang paling berguna.

Di dalamnya, antrean pada Kode 14.19 adalah list yang dilindungi sebuah kunci dan dua kondisi, "tidak kosong" dan "tidak penuh". Buku: Subbab 14.10.

Para Filsuf yang Bersantap



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`.

Pada Gambar 14.9, untuk makan seorang filsuf memerlukan **kedua** garpu di sebelahnya. Disiplin yang paling jelas: ambil garpu kiri, lalu garpu kanan, makan, letakkan keduanya.

Diajukan oleh Dijkstra pada 1965 dan diberi bentuknya yang sekarang oleh Hoare. Setiap filsuf adalah thread dan setiap garpu adalah kunci. Buku: Subbab 14.11.1 dan Gambar 14.9.

Kiri, Lalu Kanan

Kode 14.20 `Philosopher.java`, `run()`: berpikir, mengambil garpu kiri, menunggu satu satuan waktu, mengambil garpu kanan, makan.

```
    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, "<0 ");
                    sleep(UNIT);
                    synchronized (forks[right]) {           // ambil garpu kanan
                        show(right, " ", me, "<0>");
                        sleep(UNIT);                        // makan
                        ate(me);
                        show(right, "^", me, "<0 ");        // letakkan kembali garpu kanan
                    }
                    show(left, "^", me, "_0_");             // letakkan kembali garpu kiri
                }
            }
```

Pada Kode 14.20, `sleep(UNIT)` sambil memegang garpu kiri adalah celah yang berbahaya: pekerjaan apa pun di antara memperoleh dua kunci. Kunci gambar selalu diambil di dalam kunci garpu. Buku: Subbab 14.11.1.

Meja Makan Terkunci

```
  <0  ^ _0_ ^ _0_   <0  ^ _0_   32
  <0>   _0_   <0    <0    <0    33
  _0_ ^ _0_   <0    <0    <0>   34
  <0    <0    <0    <0>   _0_   35
  <0    <0    <0    <0    <0    36
The diner is locked.
Meals eaten before the deadlock: [7, 5, 5, 7, 6]
```

`^` garpu di meja, `_0_` tanpa garpu, `<0` hanya garpu kiri, `<0>` sedang makan. Pada gambar ke-36 setiap filsuf memegang garpu kiri dan menunggu garpu kanan yang dipegang tetangganya.

Yang berubah-ubah: kapan hal itu terjadi (24 sampai 224 gambar dalam percobaan kami). Yang tidak: pada akhirnya kelima filsuf mengambil garpu kiri bersamaan, dan tidak ada yang makan lagi. Python berperilaku sama dengan garpu `threading.Lock`*. Buku: Subbab 14.11.1.*

Dua Petugas dan Dua Mesin

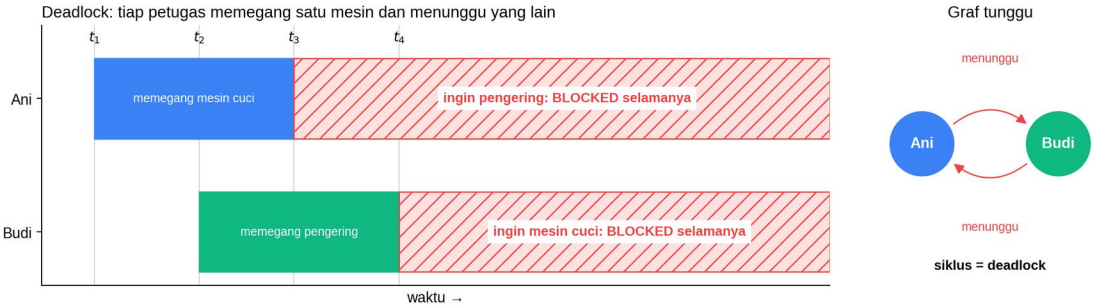
Kode 14.21 `DeadlockDemo.java`, `clerk`: mengambil mesin pertama, menunggu sampai petugas lain juga memegang satu, lalu meminta mesin kedua.

```
    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
            }
        }
    }
```

```
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
```

Pada Kode 14.21, Ani mengambil mesin cuci lebih dulu dan Budi mesin pengering lebih dulu; sebuah `CountDownLatch` membuat deadlock itu pasti. `findDeadlockedThreads()` melaporkan siklusnya; `jstack` melakukan hal yang sama dari luar. Buku: Subbab 14.11.2.

Siklus pada Graf Tunggu



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.

Pada Gambar 14.10, panah berjalan dari setiap thread yang terblokir ke pemilik kunci yang diinginkannya. **Siklus pada graf tunggu tepat sama dengan deadlock.**

Buku: Subbab 14.11.2 dan Gambar 14.10.

Empat Syarat Coffman

DEFINISI. Thread-thread berada dalam **deadlock** bila masing-masing menunggu sumber daya yang dipegang thread lain dalam kumpulannya. Deadlock memerlukan keempatnya: (1) **saling pengecualian**; (2) **pegang dan tunggu**; (3) **tanpa perampasan**; (4) **penantian melingkar**.

Patahkan **satu saja** syaratnya, dan deadlock menjadi mustahil. Pada contoh petugas, keempatnya berlaku.

Python tidak mempunyai pendeteksi bawaan: sesudah batas waktu, kedua thread masih hidup dan kedua kunci masih dipegang. `print` di Python dapat menggabungkan dua baris; kunci cetak (`say`) menjaga setiap baris tetap utuh.

Buku: Subbab 14.11.2.

Mencegah Deadlock

Pengurutan kunci: thread yang memegang ℓ_i hanya boleh meminta ℓ_j bila

$$i < j \tag{14.5}$$

Di sepanjang rantai penantian nomornya naik tegas, sehingga tidak ada siklus yang dapat tertutup.

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`

Tabel 14.3 menyebutkan syarat Coffman yang dipatahkan setiap obat; menurut Persamaan (14.5) hanya filsuf 4 yang mengambil garpu kanan lebih dulu. Buku: Subbab 14.12 dan Tabel 14.3.

Pengurutan dan Seorang Penengah

Kode 14.22 `DeadlockFixes.java`, perbaikan 1: selalu ambil garpu bernomor lebih kecil lebih dulu.

```
        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]);
        });
```

Kode 14.23 `DeadlockFixes.java`, perbaikan 3: sebuah `Semaphore` hanya mengizinkan paling banyak empat filsuf 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
```

Kode 14.22 mematahkan penantian melingkar dengan urutan; Kode 14.23 dengan tidak pernah membiarkan siklusnya tertutup. Versi naif makan 0 dari 15 kali. Buku: Subbab 14.12.

Coba, lalu Mundur

Kode 14.24 `DeadlockFixes.java`, perbaikan 2: `tryLock` dengan batas waktu; bila gagal, letakkan juga garpu kiri, lalu tunggu selama waktu acak.

```
            while (true) {
                left.lock();
                try {
                    pause(10);
                    if (right.tryLock(5, TimeUnit.MILLISECONDS)) {
                        try {
                            pause(10);          // makan
                            return;
                        } finally {
                            right.unlock();
                        }
                    }
                } finally {
                    left.unlock();              // bila gagal: letakkan juga garpu kiri
                }
                retries.incrementAndGet();
                pause(10);                      // mundur selama waktu acak, lalu coba lagi
            }
```

WAWASAN KUNCI. Pengurutan kunci adalah kuda beban: gratis saat berjalan, tanpa batas waktu, dan dapat diperiksa dengan membaca kode. Untuk objek-objek dari satu kelas, kunci nomor unik yang lebih kecil lebih dulu.

Jeda acak pada Kode 14.24 itu penting: dua thread yang gagal bersamaan dan mencoba lagi bersamaan dapat gagal bersamaan selamanya, dan itulah livelock. Buku: Subbab 14.12.

Livelock

> **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 `Livelock.java`: pemilik yang melihat pekerja lain masih aktif menyerahkan sumber dayanya, sampai `politeness` kali.

```
                // 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);
```

Pada Kode 14.25, dengan kesopanan tanpa batas sumber daya berpindah bolak-balik selama 100 ms dan tidak satu pekerjaan pun selesai; thread-threadnya tidak pernah `BLOCKED`, itulah sebabnya livelock sulit dikenali. Buku: Subbab 14.13.1.

Mematahkan Kesimetrian

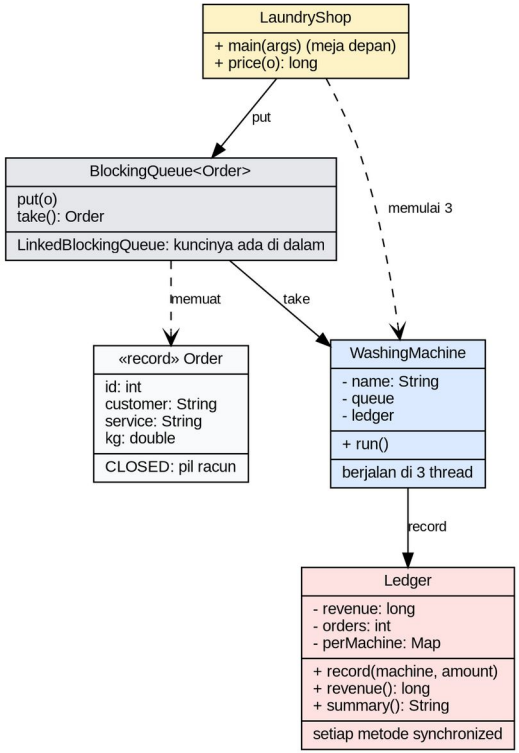


Gambar 14.11 Livelock dan obatnya. Atas: dua pekerja yang selalu sopan mengoper sumber daya bolak-balik dan tidak pernah bekerja. Bawah: setiap pekerja mengalah paling banyak sekali, sehingga setelah dua penyerahan keduanya bekerja.

Pada Gambar 14.11: batasi kesopanannya, beri prioritas, atau mundur selama waktu *acak*. **Starvation**: thread yang tidak pernah mendapat giliran; obati dengan **keadilan** (`new ReentrantLock(true)`, antrean FIFO).

Livelock adalah kegagalan rancangan, bukan kegagalan sinkronisasi: programnya sama sekali tidak mempunyai data race. Buku: Subbab 14.13.1 dan 14.13.2, serta Gambar 14.11.

Laundry yang Konkuren



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.

Pada Gambar 14.12: sebuah **antrean pemblokir** berisi pesanan, tiga **thread pekerja**, satu **buku kas yang tersinkronisasi**, dan **pil racun** untuk mematikan mesin.

Harganya sama dengan Bab 7: `WF` Rp7.000 dan `WDI` Rp10.000 per kilogram, minimal 3 kg, dibulatkan ke atas ke setengah kilogram berikutnya. Buku: Subbab 14.14 dan Gambar 14.12.

Satu Kasir

Kode 14.26 `LaundryShop.java`, `Ledger`: setiap metode `synchronized`, sehingga hanya satu mesin pada satu waktu 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;
        }
    }
```

Hapus `synchronized` dari `record` pada Kode 14.26, dan pembayaran dapat hilang persis seperti kenaikan pencacah; Soal Latihan 14.3 meminta Anda mencobanya. Buku: Subbab 14.14.

Mesin Adalah Thread Pekerja

Kode 14.27 `LaundryShop.java`, `WashingMachine.run()`: mengambil pesanan, mencucinya di luar kunci mana pun, lalu membayar.

```
        @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");
        }
```

*Pada Kode 14.27, pencucian terjadi **di luar kunci**: jagalah bagian kritis tetap pendek. `Order` adalah record imutabel, dan pil `CLOSED` dikenali melalui identitasnya. Buku: Subbab 14.14.*

Yang Berubah, dan Yang Tidak Pernah

```
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
```

Berubah: mesin mana mengambil pesanan mana, beban per mesin (4, 4, 4 atau 3, 5, 4), urutan mesin dimatikan, waktunya. **Tidak pernah:** 12 pesanan, masing-masing sekali, Rp533.500, sama dengan total satu thread.

Dengan m mesin dan waktu cuci t_i, tidak ada jadwal yang mengalahkan

$$T_{\text{wall}} \geq \max\left(\frac{1}{m}\sum_{i=1}^{k} t_i, \max_i t_i\right) \tag{14.6}$$

Persamaan (14.6) memberi sekitar 415 ms; aturan "ambil pesanan berikutnya bila luang" ditelusuri menjadi 498 ms, terukur sekitar 530 ms: "tiga mesin" tidak pernah benar-benar berarti "tiga kali lebih cepat", sejalan dengan Persamaan (14.1). Buku: Subbab 14.14.

Laundry di Python

Kode 14.28 `laundry_shop.py`, `WashingMachine.run()`: `get()` menunggu pesanan; `None` adalah pilnya; pencetakan melalui `say()`.

```
    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")
```

Mencuci adalah menunggu, sehingga GIL dilepas selama setiap cucian dan ketiga mesin pada Kode 14.28 bertumpang tindih sebaik mesin Java: sekitar 500 ms untuk 1.246 ms cucian. Buku: Subbab 14.14.

Profitina dalam Praktik

Permintaan adalah pelanggan di meja depan: selagi satu permintaan menunggu basis data, permintaan lain dilayani (FastAPI di atas Uvicorn).

Penjadwal adalah produsen, pemindai adalah konsumen, dipisahkan oleh sebuah antrean, dengan bentuk seperti Kode 14.27.

Model bahasa lokal adalah kasir tunggal: permintaan menunggu gilirannya; bila tidak ada model yang tersedia, aplikasi mengatakannya dengan jujur.

Pencacah yang tidak boleh kehilangan pembaruan (batas percobaan login, saldo koin PROFIT) disimpan di PostgreSQL, yang transaksi dan kunci barisnya berperan seperti `synchronized`.

Perangkat lunak mengukur dan menjelaskan; pengguna yang memutuskan. Buku: Subbab 14.15.

Kesalahan yang Sering Terjadi

1. `run()` **sebagai ganti** `start()`**. 2. Data bersama tanpa kunci. 3. Menjaga operasi tulis tetapi tidak operasi baca. 4.** `volatile` **untuk** `++`**.**

5. Memercayai GIL. 6. `if` **sebagai ganti** `while` **di sekitar** `wait()`**. 7.** `wait`**/**`notify` **tanpa memegang kunci.**

8. Kunci dalam urutan yang berbeda (Persamaan (14.5)). **9. Pekerjaan lambat di dalam kunci. 10. Menelan** `InterruptedException`**.**

11. Menunggu selamanya: pakailah batas waktu. **12. Mencetak dari thread Python tanpa kunci. 13. Memercayai uji yang lolos** (Persamaan (14.2)).

Setiap kesalahan memiliki obatnya di buku, di samping program yang memperlihatkannya. Buku: Subbab 14.16.

Ringkasan — Lima Hal

1. **Thread berbagi heap**; penjadwal memilih selang-selingnya, berbeda pada setiap jalan. Panggil `start()`, tunggu dengan `join()`, hentikan secara kooperatif (`interrupt()`, sebuah `Event`, bendera `volatile`).
2. **GIL**: thread Python membantu menunggu, bukan menghitung dalam Python murni; percepatan tunduk pada Persamaan (14.1).
3. **Race** menghilangkan pembaruan pada baca-ubah-tulis; obati dengan `synchronized`, kunci, atau atomik, yang sekaligus memberi visibilitas.
4. **Menunggu** harus di dalam perulangan `while` di sekitar `wait()`, atau lebih baik dengan `BlockingQueue` / `queue.Queue` beserta pil racun.
5. **Deadlock** memerlukan empat syarat; patahkan satu, biasanya dengan pengurutan kunci. **Livelock** adalah kesibukan tanpa kemajuan; **starvation** diobati dengan keadilan.

Buku: Subbab 14.17.

Pertemuan Berikutnya

Bab 15 — Socket, Jaringan, dan Proyek Tim. Program di komputer yang berbeda bercakap-cakap melalui TCP, protokol yang dirancang di atas kertas lebih dulu, dan HTTP.

Server memberi setiap klien thread-nya sendiri: thread pekerja, antrean, dan kunci hari ini bertemu dengan jaringan.

Pekerjaan rumah. Soal Latihan 14.1–14.11; mulailah dengan 14.2 (menghitung selang-seling dengan Persamaan (14.2)) dan 14.8 (dua dompet tanpa deadlock).

Penyelesaiannya ada di Subbab 14.19; cobalah setiap soal terlebih dahulu. Buku: Subbab 14.18 dan 14.19.

Kuliah 15 — Socket, Jaringan, dan Proyek Tim

Pemrograman Berorientasi Objek (EF234302)

Bab 15 buku ini dalam bentuk ringkas, untuk dipakai di kelas.

Buku: Irfan Subakti, *Pemrograman Berorientasi Objek dengan Java dan Python*.

Pengajar: Irfan Subakti — yifana@gmail.com

Program yang bercakap dengan program lain, yang mungkin ditulis orang lain dalam bahasa lain: pertama jaringannya, lalu orang-orangnya. Buku: Bab 15, paragraf pembuka.

Agenda

1. **Jaringan dari nol.** Alamat IP, port, lapisan, TCP dan UDP, klien dan server.
2. **Socket.** Server dan klien *echo* berbasis baris di Java dan Python; pembilasan dan pembingkaian; ketika pihak lain tidak ada.
3. **Protokol di atas kertas lebih dulu.** Protokol knock-knock sebagai tabel, diagram, dan objek; banyak klien sekaligus dengan kumpulan thread.
4. **Profitina Laundry.** Layanan status pesanan dengan JSON di atas TCP, Java dan Python saling bercakap; HTTP; keamanan dan UU PDP; pengujian.
5. **Proyek tim.** Topik, komponen dan antarmuka, pengujian silang, Git, penanganan kegagalan, dokumentasi, presentasi.

Halaman-halaman ini mengikuti urutan buku; soal latihan di akhir bab adalah pekerjaan rumahnya. Buku: Subbab 15.2 sampai 15.11.

Di Mana Kita Berhenti

Setiap program sejauh ini **hidup sendirian**: satu komputer, papan ketik, sebuah berkas, layar. Bab 11 memberi program kita jendela dan Bab 14 memberinya thread, tetapi tidak satu pun bercakap dengan program lain.

Perangkat lunak sungguhan senang bergaul: laptop kasir bertanya kepada server apakah ORD-0713 sudah siap; ponsel kurir melaporkan pengantaran; dasbor pemilik meminta pendapatan hari ini.

> **WAWASAN KUNCI.** Koneksi jaringan hanyalah dua aliran, satu untuk setiap arah, seperti aliran berkas pada Bab 5. Yang sulit: pihak lain adalah program terpisah, ia dapat menghilang kapan saja, dan kedua pihak harus sepakat tentang makna setiap bita.

Entah pihak-pihaknya program atau manusia, mereka berhasil bila menuliskan, sebelum mulai, apa tepatnya yang akan dikatakan dan boleh diharapkan setiap pihak. Buku: Subbab 15.1.

Host dan Alamat IP

DEFINISI. Alamat IP mengenali sambungan sebuah host ke jaringan. **IPv4**: 32 bit, ditulis sebagai empat bilangan 0–255, misalnya `192.168.1.10`. **IPv6**: 128 bit, delapan kelompok heksadesimal, misalnya `2001:db8::10`.

Loopback: `127.0.0.1` dan `::1`, "komputer ini sendiri"; setiap contoh hari ini memakainya. `localhost` adalah nama host yang diterjemahkan menjadi alamat loopback.

Alamat **privat** (`192.168.x.y`, `10.x.y.z`) hidup di dalam jaringan rumah atau kantor; server laundry dan laptop kasir mempunyai alamat seperti itu.

$2^{32} \approx 4{,}3$ *miliar alamat IPv4 terlalu sedikit untuk planet yang penuh ponsel; IPv6 adalah jawaban jangka panjangnya. Buku: Subbab 15.2.1.*

Port dan Empat-Serangkai

DEFINISI. Port adalah bilangan 16 bit yang dipakai sistem operasi untuk mengantarkan data jaringan ke program yang tepat. Server *mendengarkan* pada sebuah port; klien menyebut port itu ketika menyambung.

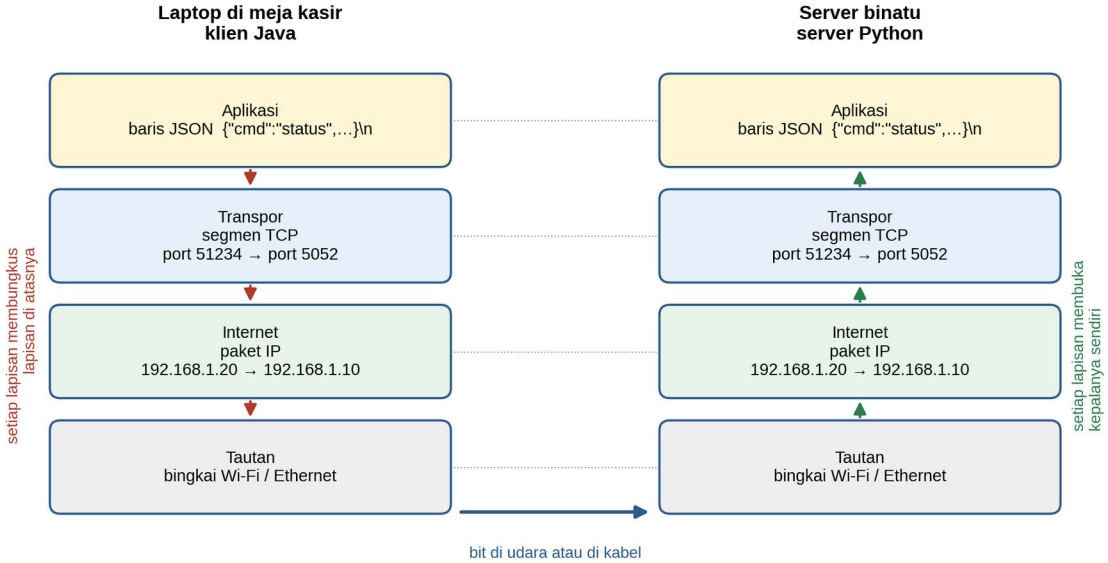
$$2^{16} = 65\,536, \text{port } 0, 1, \ldots, 65\,535. \tag{15.1}$$

Terkenal 0–1023 (80 HTTP, 443 HTTPS); terdaftar 1024–49151 (kita memakai 5050–5052); efemeral 49152–65535. **Port 0** meminta sistem memilihkan port yang bebas.

$$\text{koneksi} = (IP_c, p_c, IP_s, p_s) \tag{15.2}$$

Menurut Persamaan (15.1) ada 65.536 port per protokol; menurut Persamaan (15.2) satu port yang didengarkan dapat melayani ribuan klien, karena setiap klien mendapat p_c efemeralnya sendiri. Buku: Subbab 15.2.2.

Bagaimana Sebuah Pesan Berpindah



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.

Pada Gambar 15.1: **tautan** (satu lompatan), **internet** (IP, host ke host, tanpa janji), **transpor** (TCP atau UDP, program ke program, port), **aplikasi** (makna bita-bitanya). Setiap lapisan menambahkan header-nya sendiri dalam perjalanan turun.

Setiap lapisan tampak bercakap langsung dengan kembarannya di host lain, ilusi yang paling berguna dalam jaringan. Buku: Subbab 15.2.3 dan Gambar 15.1.

TCP dan UDP

DEFINISI. TCP menyediakan *koneksi*: *aliran bita* dua arah yang andal dan berurutan. **UDP** menyediakan *datagram*: pesan-pesan mandiri, masing-masing utuh atau tidak sama sekali, tanpa janji pengantaran maupun urutan.

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

Tabel 15.1: TCP adalah panggilan telepon sesudah jabat tangan tiga arah; UDP adalah setumpuk kartu pos. Laundry tidak boleh kehilangan "ORD-0713 sudah diambil", jadi kita memakai TCP. Buku: Subbab 15.2.4 dan Tabel 15.1.

Datagram untuk Diri Sendiri

Kode 15.1 `UdpHello.java`: tiga datagram dikirim ke penerima yang terikat pada port 0 di loopback, dan diterima satu datagram utuh per pemanggilan.

```
            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);
            }
```

```
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
```

Pada Kode 15.1, `getBytes(StandardCharsets.UTF_8)`: jaringan membawa bita, tidak pernah karakter. Tiga pengiriman menghasilkan tiga penerimaan tepat 19, 16, dan 18 bita: UDP menjaga batas pesan. `setSoTimeout(2000)` mencegah datagram yang hilang menggantung `receive`. Buku: Subbab 15.2.5.

Hal yang Sama di Python

Kode 15.2 `udp_hello.py`: `SOCK_DGRAM` berarti UDP; alamat ditulis sebagai tuple `(host, port)`.

```
    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')}")
```

Modul `socket` Python pada Kode 15.2 adalah pembungkus tipis Berkeley sockets (Unix, 1983); Java menyembunyikannya di balik satu kelas per tugas. Keluarannya persis sama dengan program Java. Buku: Subbab 15.2.5.

Klien dan Server

DEFINISI. Server berjalan lebih dahulu, mengikat socket ke port yang diketahui, lalu menunggu. **Klien** mengetahui host dan port server lalu *memulai* hubungan. Peran ini soal siapa yang memulai, bukan soal besarnya komputer.



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).

Pada Gambar 15.2, klien juga terikat ke port lokalnya sendiri, biasanya ditetapkan oleh sistem. Buku: Subbab 15.2.6 dan Gambar 15.2.

Socket Pendengar dan Socket Tersambung



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.

Pada Gambar 15.3, `accept` mengembalikan socket **baru** untuk klien itu; socket **pendengar** tetap terbuka untuk klien berikutnya. Socket pendengar tidak pernah membawa data; setiap socket tersambung mempunyai empat-serangkainya sendiri.

Inilah gagasan terpenting dalam menulis server. Buku: Subbab 15.2.6 dan Gambar 15.3.

Socket di Java dan Python

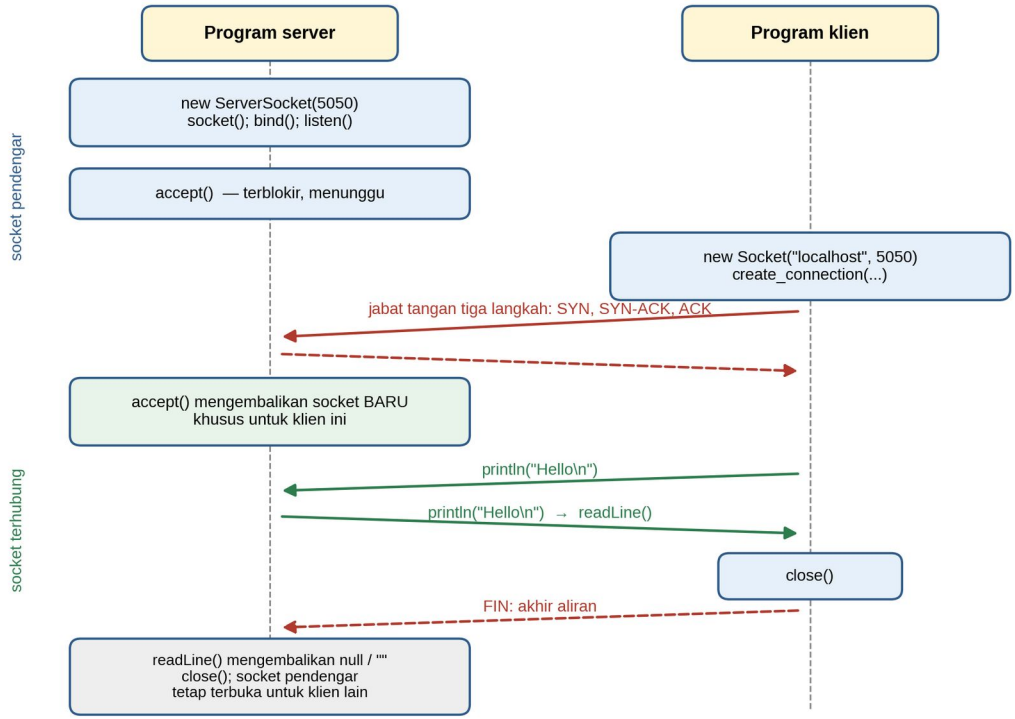
DEFINISI. Socket adalah satu titik ujung dari tautan komunikasi dua arah antara dua program di jaringan, terikat ke sebuah port agar lapisan transpor dapat mengenali program tujuan datanya.

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`

Tabel 15.2 mendaftar pemanggilan dalam urutan yang dipakai program. Buku: Subbab 15.3 dan Tabel 15.2.

Kehidupan Satu Koneksi



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.

Bungkus alirannya (`BufferedReader`, `PrintWriter`; `makefile`). **Selalu sebutkan pengodeannya**, UTF-8. **Bilas**: `PrintWriter` dengan *autoflush*, `writer.flush()` di Python.

Lupa membilas adalah alasan paling umum program jaringan pertama "macet". Gambar 15.4 diakhiri `close()` milik klien, yang dilihat server sebagai akhir masukan. Buku: Subbab 15.3 dan Gambar 15.4.

Server Echo di Java

Kode 15.3 `EchoServer.java`: mendengarkan di loopback, menerima satu klien, dan mengirim balik setiap baris.

```
        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");
            }
```

Pada Kode 15.3, `accept()` terblokir sampai klien datang; `readLine()` mengembalikan `null` ketika klien menutup; backlog 50 adalah banyaknya permintaan koneksi yang boleh mengantre. Buku: Subbab 15.4.1.

Server Echo di Python

Kode 15.4 `echo_server.py`: `bind`, `listen`, `accept`, dan perulangan `for` atas baris-baris klien.

```
        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")
```

Pada Kode 15.4, `SO_REUSEADDR` membuat server yang dinyalakan ulang dapat langsung mengikat port walaupun ada TIME_WAIT (`ServerSocket` Java memasangnya secara bawaan); `flush=True` membuat baris "listening" tampak bahkan melalui pipa. Buku: Subbab 15.4.1.

Klien Echo

Kode 15.5 `EchoClient.java`: mengirim setiap baris yang diketik, mencetak gemanya, dan mengubah tiga macam kegagalan menjadi kalimat.

```
            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?");
        } catch (IOException e) {
            System.out.println("I/O error: " + e.getMessage());
        }
```

Lima langkah setiap klien: 1. buka socket; 2. buka aliran; 3. bercakap menurut protokol; 4. tutup aliran; 5. tutup socket. Hanya langkah 3 yang berbeda-beda.

Pada Kode 15.5, try-with-resources mengerjakan langkah 4 dan 5; `ConnectException` *berarti tidak ada yang mendengarkan. Buku: Subbab 15.4.2.*

Klien Echo di Python

Kode 15.6 `echo_client.py`: `create_connection` mengerjakan langkah 1; `gaierror` dan `ConnectionRefusedError` melaporkan kegagalan.

```
            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}")
```

`create_connection` pada Kode 15.6 mencoba setiap alamat hasil terjemahan nama host (`localhost` dapat berupa `::1` sekaligus `127.0.0.1`). Buku: Subbab 15.4.2.

Dua Program, Dua Terminal

```
Terminal 1:  java EchoServer 5050
Terminal 2:  java EchoClient localhost 5050
```

```
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
```

```
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
```

Klien Java bekerja dengan server Python dan sebaliknya: **protokol**, bukan bahasa, yang menentukan siapa dapat bercakap dengan siapa.

Jalankan server lebih dulu. Tanpa server, kedua klien mencetak `Cannot connect to localhost port 5050: is the server running?`. *Buku: Subbab 15.4.3.*

Pembingkaian, dan Pihak Lain yang Hilang

DEFINISI. Pembingkaian adalah aturan untuk menemukan akhir sebuah pesan di dalam aliran bita: **pembatas** (baris baru), **awalan panjang**, atau **ukuran tetap**.

`Hello\n` dan `bye\n` dapat tiba sebagai satu potong atau tiga potong. Baris baru yang terlupa atau pembilasan yang terlupa adalah **deadlock yang melintasi dua program**.

Server mati: `ConnectException` seketika. Klien menghilang: `null` atau `""`, atau tidak ada tanda sama sekali, maka pakailah **batas waktu**. Server menghilang di tengah jalan: akhir aliran atau “connection reset”.

PERINGATAN. Setiap pemanggilan jaringan yang memblokir memerlukan jawaban atas pertanyaan “bagaimana jika pihak lain tidak pernah menjawab?”: batas waktu ditambah penanganan eksepsi.

Buku: Subbab 15.4.4 dan 15.4.5.

Protokol sebagai Tabel

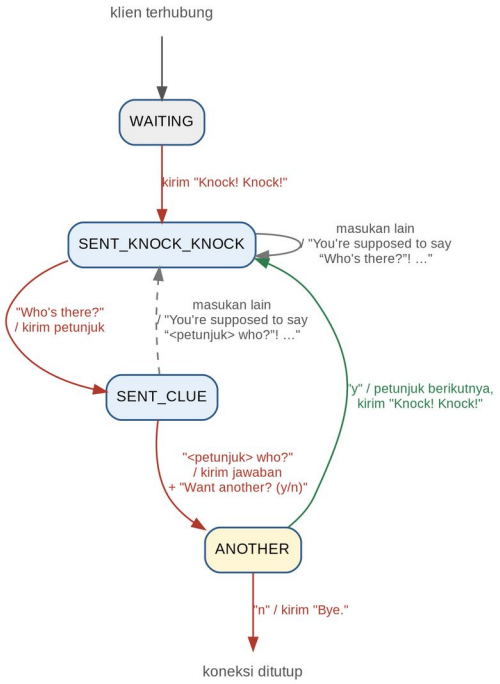
DEFINISI. Protokol adalah seperangkat lengkap aturan sebuah percakapan: pesan-pesannya, format persisnya, urutannya, maknanya, dan apa yang dilakukan setiap pihak terhadap pesan yang tidak diharapkannya.

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`

Setiap baris Tabel 15.3 menyatakan apa yang terjadi bila klien mengatakan hal lain; baris-baris itulah tempat program jaringan menghabiskan sebagian besar hidupnya yang sungguhan. Buku: Subbab 15.5 dan Tabel 15.3.

Protokol sebagai Diagram



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.

> **WAWASAN KUNCI.** Tuliskan protokolnya tiga kali sebelum menjalankannya: sebagai tabel, sebagai diagram (Gambar 15.5), dan sebagai objek tanpa I/O. Masing-masing menangkap kesalahan yang disembunyikan yang lain.

Tabel lebih baik untuk memeriksa bahwa setiap kasus tertangani, diagram untuk melihat bentuknya; simpan keduanya di repositori. Buku: Subbab 15.5 dan Gambar 15.5.

Protokol sebagai Objek

Kode 15.7 `KnockKnockProtocol.java`, `processInput`, paruh pertama: satu `case` untuk setiap keadaan, seperti dalam tabel.

```
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!";
```

Protokol pada Kode 15.7 ***tidak tahu apa-apa tentang socket****: satu objek per percakapan, keadaan berupa* `enum` *privat, dan satu metode dari baris terakhir klien ke baris berikutnya server. Buku: Subbab 15.5.*

Diuji Tanpa Jaringan

Kode 15.8 `KnockKnockProtocol.java`, `processInput`, paruh kedua: lelucon berikutnya, atau `Bye.`.

```
            case ANOTHER:
                if (input.equalsIgnoreCase("y")) {
                    joke = (joke + 1) % CLUES.length;
                    state = State.SENT_KNOCK_KNOCK;
                    return "Knock! Knock!";
                }
                state = State.WAITING;
                return "Bye.";
            default:
                throw new IllegalStateException("unknown state " + state);
        }
    }
```

```
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.
```

`Iron?` bukan `Iron who?`, sehingga Kode 15.8 kembali ke `SENT_KNOCK_KNOCK`, tepat seperti baris kelima Tabel 15.3. `main` kelas ini memainkan percakapan yang sudah ditulis naskahnya. Buku: Subbab 15.5.

Protokol di Python

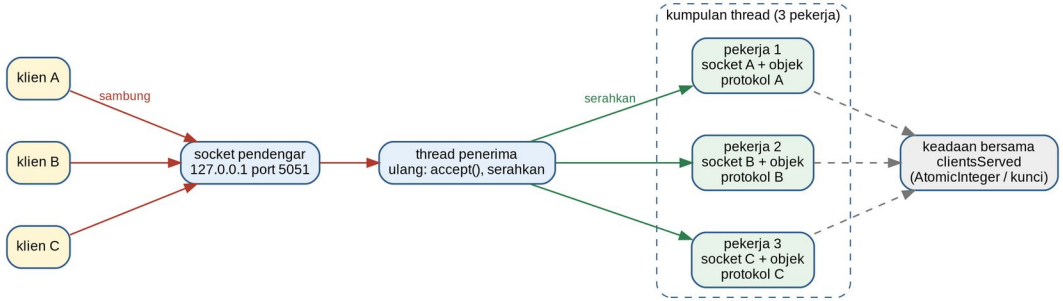
Kode 15.9 `knock_knock_protocol.py`: keadaan `enum.Enum` dan rangkaian pernyataan `if`.

```
    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!"
```

Mengapa kelas tersendiri? Protokol diuji dengan pemanggilan metode biasa; kode socket menjadi perulangan pendek yang berlaku untuk protokol berbasis baris *apa pun*; dua anggota tim bekerja paralel dan bertemu di `processInput`.

Kode 15.9 mencetak persis apa yang dicetak versi Java. Buku: Subbab 15.5.

Melayani Banyak Klien Sekaligus



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.

Satu thread per klien memakan memori stack: dengan n klien dan stack berukuran s,

$$M \approx n \cdot s \tag{15.3}$$

sehingga seribu klien yang menganggur dengan 1 MB memesan sekitar 1 GB.

*Persamaan (15.3) adalah alasan server memakai **kumpulan thread**, seperti pada Gambar 15.6; thread virtual Java 21 dan `asyncio` Python adalah alternatif modernnya. Buku: Subbab 15.6 dan 15.6.1, serta Gambar 15.6.*

Pintu Depan Hanya Menerima

Kode 15.10 `KnockKnockServer.java`, `acceptLoop`: menerima, menyerahkan socket ke kumpulan, lalu menerima klien berikutnya.

```
    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());
            }
        }
    }
```

Pada Kode 15.10, klien yang lambat tidak dapat memblokir pintu, karena penerima tidak pernah bercakap dengan klien sendiri. `start(port)` *mengembalikan port yang benar-benar terikat, yang penting untuk port 0. Buku: Subbab 15.6.2.*

Satu Percakapan per Pekerja

Kode 15.11 `KnockKnockServer.java`, `handle`: batas waktu diam dan objek protokol **baru** untuk setiap klien.

```
            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;
                }
            }
```

Field bersama memakai perkakas Bab 14: pencacah `AtomicInteger`, set socket terbuka yang konkuren, bendera `volatile`. **Bagilah sesedikit mungkin, dan lindungi apa yang harus dibagi.**

`stop()` menutup socket pendengar (`accept` yang terblokir melemparkan eksepsi), menunggu penerima, menutup setiap socket terbuka, dan mematikan kumpulan; Kode 15.11 mencatat setiap klien di `finally`. Buku: Subbab 15.6.2.

Server Python

Kode 15.12 `knock_knock_server.py`, `_accept_loop`: batas waktu `accept` 0,2 detik membuat perulangan dapat memeriksa event berhenti.

```
    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
```

Menutup socket pendengar dari thread lain tidak dapat diandalkan untuk membangunkan `accept()` di Python, maka ada batas waktu pada Kode 15.12; `shutdown(SHUT_RDWR)` membangunkan pekerja yang terblokir di `readline`. Buku: Subbab 15.6.2.

Membuktikan Server Itu Konkuren

Kode 15.13 `KnockKnockLoadTest.java`, `converse`: setiap klien yang sudah disapa menunggu di `CyclicBarrier` sampai ketiganya disapa.

```
        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());
            }
```

```
server stopped; clients served: 3
all clients were greeted before any finished: true
```

Server yang sekuensial tidak akan pernah menyapa klien kedua selagi klien pertama menunggu; barrier pada Kode 15.13 akan habis waktunya dan uji mencetak `false`. Server berjalan pada port efemeral di dalam uji itu sendiri. Buku: Subbab 15.6.4.

Protokol Status Pesanan

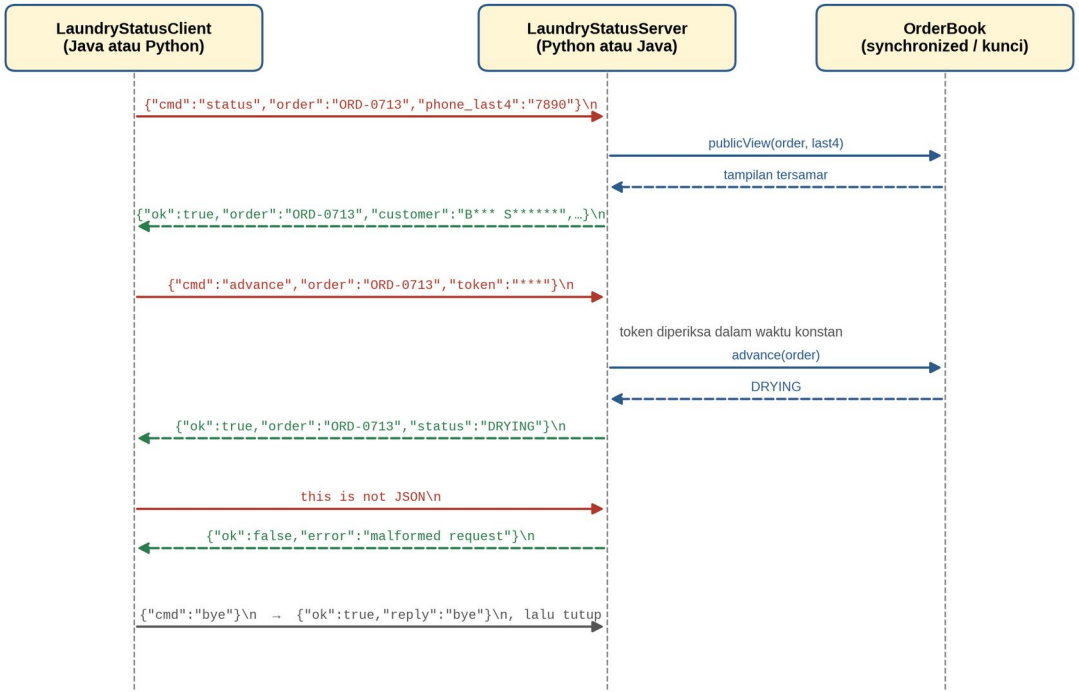
DEFINISI. JSON adalah format teks untuk data terstruktur: objek `{"key": value}`, larik, string, bilangan, `true`, `false`, `null`. Satu objek per baris disebut *JSON Lines*.

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

Pada Tabel 15.4, klien berbicara lebih dulu, setiap permintaan mendapat tepat satu tanggapan, dan setiap tanggapan memuat `ok`. Python mempunyai `json`; `Json.java` di buku adalah pengurai recursive-descent yang kecil. Buku: Subbab 15.7.1 dan 15.7.2, serta Tabel 15.4.

Percakapan dengan Layanan



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 pada Gambar 15.7: empat digit terakhir nomor telepon diwajibkan; tanggapan hanya menampilkan inisial; nomor pesanan yang salah dan nomor telepon yang salah mendapat galat yang *sama*; perintah staf memerlukan token dari variabel lingkungan.

Status mengikuti siklus hidup Bab 1, dari `RECEIVED` sampai `PICKED_UP`. Buku: Subbab 15.7.1 dan Gambar 15.7.

Seluruh Protokol sebagai Satu Fungsi

Kode 15.14 `LaundryStatusServer.java`, `handle`, paruh pertama: `ping` dan `status`, dengan setiap field diperiksa sebelum dipakai.

```
    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");
```

`handle` *pada Kode 15.14 memetakan map permintaan ke map tanggapan tanpa I/O, sehingga dapat diuji tanpa jaringan.* `{"cmd":"status","order":42}` *harus ditolak, bukan membuat pekerja runtuh. Buku: Subbab 15.7.3.*

Perintah Staf Memerlukan Token

Kode 15.15 `LaundryStatusServer.java`, `handle`, paruh kedua: `advance`, `bye`, dan `shutdown`.

```
        } 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");
        }
```

*Pada Kode 15.15, `isStaff` membandingkan dengan `MessageDigest.isEqual`, dalam waktu konstan. `OrderBook` sepenuhnya `synchronized`, dan `mask` mengubah "Budi Santoso" menjadi "B*** S*******". Baris yang lebih panjang dari 1024 karakter ditolak. Buku: Subbab 15.7.3.*

Server di Python

Kode 15.16 `laundry_status_server.py`, `handle`: aturan yang sama, kelas demi kelas.

```
    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")
```

Pada Kode 15.16, `threading.Lock` menggantikan `synchronized`, `hmac.compare_digest` adalah perbandingan waktu konstan, dan `readline(MAX_LINE + 1)` membatasi panjang baris secara langsung. Buku: Subbab 15.7.3.

Objek Klien Menyembunyikan Kabelnya

Kode 15.17 `LaundryStatusClient.java`: `status(...)` tampak seperti pemanggilan biasa.

```
    public Map<String, Object> status(String order, String phoneLast4) throws IOException {
        return send(Json.object("cmd", "status", "order", order, "phone_last4", phoneLast4));
    }
```

Kode 15.18 `LaundryStatusClient.java`: `send` menulis satu baris JSON dan membaca satu baris, seluruh protokol dari sisi klien.

```
    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);
    }
```

*Kode 15.17 adalah sebuah **proxy**: kelas ini mewakili layanan di dalam klien. Kode 15.18 melemparkan eksepsi ketika server menutup koneksi. Konstruktornya menyambung dengan batas 2 detik dan membaca dengan batas 5 detik. Buku: Subbab 15.7.4.*

Skenarionya, Dicocokkan dengan Tabel

```
> {"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"}
```

Digit telepon yang benar memperlihatkan nama yang disamarkan; digit yang salah dan pesanan yang tidak dikenal mendapat penolakan yang sama; sampah mendapat `malformed request` dan koneksi tetap hidup; token tebakan ditolak, token yang asli memindahkan `WASHING` ke `DRYING`. Buku: Subbab 15.7.4.

Objek Klien di Python

Kode 15.19 `laundry_status_client.py`, `send` dan `send_raw`; `__enter__` dan `__exit__` membuatnya dapat dipakai dalam pernyataan `with`.

```
    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()
```

Java bercakap dengan Python, Python bercakap dengan Java. `InteropDemo.java` menjalankan server **Python** sebagai proses terpisah pada port 0 dan menjalankan skenario Java; `interop_demo.py` melakukan kebalikannya.

```
> {"cmd":"shutdown","token":"***"}
< {"ok":true,"reply":"shutting down"}
Python server exited: true, exit code 0
```

Dengan Kode 15.19, kedua keluaran identik kecuali field `server` pada `ping`: itulah yang diberikan sebuah spesifikasi. Token untuk jalan itu berasal dari `SecureRandom` dan dikirim melalui lingkungan, tidak pernah melalui baris perintah. Buku: Subbab 15.7.4 dan 15.7.5.

Biaya Sebuah Percakapan

Satu pesan berukuran S bita, latensi L, lebar pita B:

$$T = L + \frac{S}{B} \tag{15.4}$$

k pasangan permintaan–tanggapan, waktu server t_s:

$$T(k) \approx k \cdot (RTT + t_s), RTT = 2L + \frac{S_{\text{req}} + S_{\text{resp}}}{B}. \tag{15.5}$$

CONTOH. Berjarak 20 ms dengan 10 Mbit/s: S/B untuk 180 bita $\approx 0,14$ ms dibanding $2L = 40$ ms. Menurut Persamaan (15.5), sepuluh permintaan memakan sekitar 400 ms; satu permintaan untuk sepuluh pesanan sekitar 40 ms.

Pada Persamaan (15.4), latensi mendominasi pesan kecil: utamakan sedikit pesan yang kaya daripada banyak pesan kecil. Buku: Subbab 15.7.6.

HTTP: Protokol Web

Baris-baris teks di atas TCP, dibakukan: baris permintaan (`GET /orders/ORD-0714 HTTP/1.1`), header, satu baris kosong, badan; tanggapannya diawali baris status (`HTTP/1.1 200 OK`).

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

Pada Tabel 15.5, 2xx berarti berhasil, 4xx masalah pada permintaan, 5xx masalah di server. Pembingkaiannya: header berakhir pada baris kosong; `Content-Length` menghitung bita badan. Buku: Subbab 15.8.1 dan Tabel 15.5.

HTTP di Java

Kode 15.20 `HttpStatusDemo.java`: `HttpClient` mengajukan tiga pertanyaan kepada `HttpServer` di dalam proses yang sama.

```
        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));
            System.out.println("GET " + path + " -> " + response.statusCode() + " "
                    + response.headers().firstValue("content-type").orElse("?"));
            System.out.println("    body: " + response.body());
```

```
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"}
```

Pada Kode 15.20, 404 adalah tanggapan, bukan eksepsi. `NO_PROXY` *menyambung secara langsung. Program yang sama juga menulis permintaan dengan tangan pada* `Socket` *biasa dan mendapat* `HTTP/1.1 200 OK`*: HTTP hanyalah teks di atas TCP. Buku: Subbab 15.8.2.*

HTTP di Python

Kode 15.21 `http_status_demo.py`, `do_GET`: pustaka mengurai permintaan lalu memanggil metode kita.

```
    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")
```

Kode 15.21 adalah pola template method dari Bab 7; `ThreadingHTTPServer` melayani setiap permintaan di thread tersendiri; `urllib` membangkitkan `HTTPError` untuk 4xx dan 5xx. Bagian belakang Profitina berbicara HTTP melalui FastAPI. Buku: Subbab 15.8.2.

Tujuh Kebiasaan untuk Kode Jaringan

1. Dengarkan hanya di tempat yang perlu (`127.0.0.1` selama pengembangan). **2. Jangan percaya apa pun** dari pihak lain. **3. Batasi segalanya**: panjang baris, lama diam, jumlah pekerja.

4. Rahasia di luar kode dan log. 5. Enkripsi apa pun yang meninggalkan mesin: **TLS**. **6. Sedikit bicara ketika menolak.**

7. Perlambat penebak. Empat digit, r percobaan per detik:

$$t_{\text{guess}} \approx \frac{10^4 + 1}{2 \cdot r}. \tag{15.6}$$

Pada $r = 1000$, Persamaan (15.6) memberi sekitar lima detik; lima percobaan per menit memberi sekitar 17 jam: itulah pembatasan laju. Buku: Subbab 15.9.1.

Apa yang Diminta UU PDP dari 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*

PERINGATAN. Jangan pernah memakai data pelanggan sungguhan dalam proyek, uji, tangkapan layar, atau demonstrasi. Karanglah nama dan nomornya.

Tabel 15.6: log mencatat `status -> ok`, bukan pesanan siapa; `publicView` mengembalikan inisial, tidak pernah nomor telepon. Pelindungan data adalah keputusan-keputusan kecil yang diambil sewaktu merancang protokol. Buku: Subbab 15.9.2 dan Tabel 15.6.

Menguji Program Jaringan

Uji protokolnya **tanpa** jaringan; jalankan server sungguhan **pada port 0** di dalam uji; server **baru** untuk setiap uji; uji **kegagalan** dengan sengaja.

Kode 15.22 `LaundryStatusCheck.java`, `check`: server baru dengan data baru untuk setiap pemeriksaan, selalu dihentikan di `finally`.

```
    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();
        }
```

```
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
```

Kode 15.22 menjalankan sepuluh pemeriksaan, termasuk dua puluh klien sekaligus dan lima klien staf yang tidak kehilangan satu pembaruan pun; rangkaian `unittest` Python melakukan hal yang sama. Buku: Subbab 15.10.

Proyek Tim: Sebuah Topik

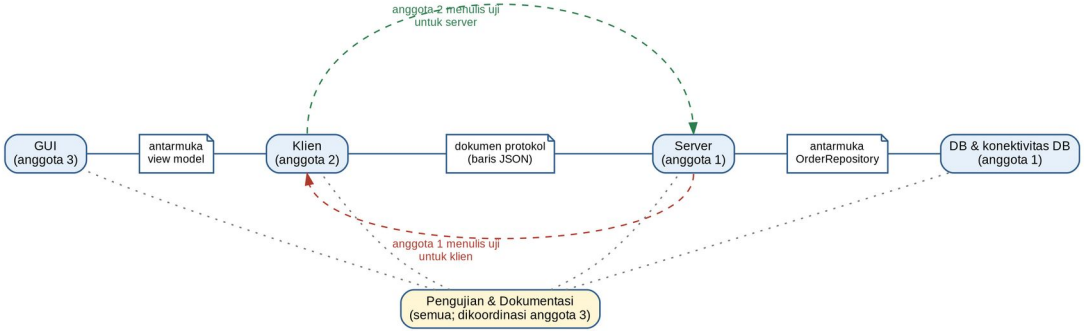
Topiknya harus *memerlukan* klien dan server: permainan dengan obrolan, jejaring sosial kecil, alat kolaborasi, portal interaktif, gagasan rintisan.

Tiga uji. *Jaringan*: perlukah jaringan? *Ukuran*: dapatkah intinya diperlihatkan sesudah dua pertiga waktu? *Kegagalan*: dapatkah Anda menyebut kegagalan jaringan yang berarti bagi pengguna?

Tetapkan topik, judul kerja, dan satu paragraf gagasan pada pertemuan pertama: siapa penggunanya, untuk apa, dan mengapa memerlukan jaringan.

Buku: Subbab 15.11 dan 15.11.1.

Lima Komponen, Tiga Antarmuka



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.

Pada Gambar 15.8: server, klien, basis data, GUI, pengujian dan dokumentasi, masing-masing dengan pemilik. Di antaranya: **dokumen protokol**, **antarmuka repositori**, **view model**.

Kepemilikan berarti "bertanggung jawab atasnya dan meninjau setiap perubahan", bukan "orang lain tidak boleh menyentuhnya". Buku: Subbab 15.11.2 dan Gambar 15.8.

Spesifikasi, Rancangan, Implementasi

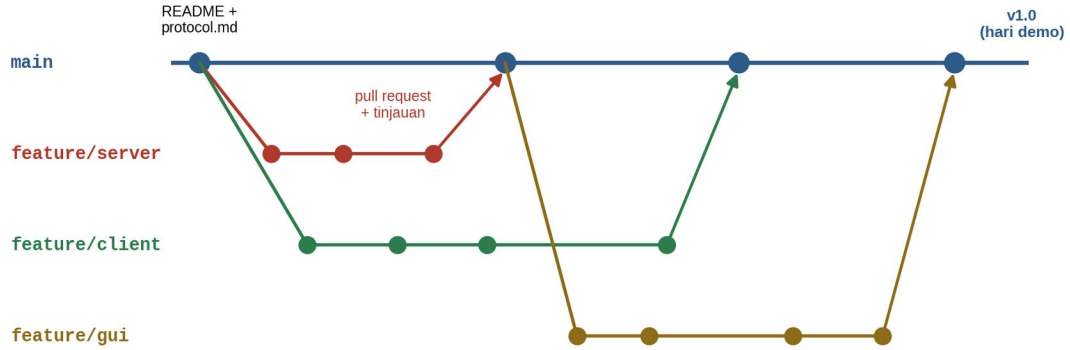
Spesifikasi: fitur dalam satu kalimat beserta ukuran berhasilnya. **Rancangan**: komponen, antarmuka, UML, diagram keadaan. **Implementasi**: kode dan uji menurut rancangan.

Pekan 1 antarmuka bersama-sama; **pekan 2** paralel dengan **stub**; **pekan 3** stub diganti, uji silang; **pekan 4** kegagalan, dokumen, gladi.

Pengujian silang: bila A memakai B, pemilik A merancang uji untuk B. Uji unit, berbasis stub, multithread, kegagalan, dan terdistribusi.

Orang yang menulis sebuah komponen menguji apa yang dimaksudkannya; pemakainya menguji apa yang diandaikannya; bug hidup di celah di antara keduanya. Buku: Subbab 15.11.3 dan 15.11.4.

Bekerja Bersama dengan Git



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*.

Pada Gambar 15.9: `main` selalu terkompilasi dan lolos ujinya; setiap perubahan hidup di cabang fitur dan kembali melalui **pull request** yang ditinjau. Lakukan commit kecil dan sering; jauhkan rahasia dengan `.gitignore`.

Perintah pekan pertama: `git clone`, `git switch -c`, `git add`, `git commit -m`, `git push`, `git pull`, `git merge main`. Setiap repositori diawali README. Buku: Subbab 15.11.5 dan 15.11.6, serta Gambar 15.9.

Kegagalan, Dokumen, Presentasi

Server mati: klien mengatakannya dengan jelas, menyimpan masukan pengguna, mencoba lagi dengan jeda yang makin panjang. **Klien mati**: server terus berjalan dan membersihkan diri di `finally`.

Empat dokumen: panduan pengguna, kriteria kinerja dalam angka yang dapat diperiksa, panduan instalasi, panduan pemecahan masalah.

Presentasikan masalahnya dalam satu kalimat, peragakan langsung di beberapa komputer, matikan server dengan sengaja, tunjukkan ujinya, biarkan setiap anggota berbicara, dan berlatihlah dua kali.

Tidak ada yang lebih meyakinkan daripada kegagalan yang ditangani dengan baik. Buku: Subbab 15.11.7 sampai 15.11.9.

Contoh Rancangan Proyek: Laundry Live Board

Staf mencatat pesanan dan memindahkannya dari tahap ke tahap; setiap layar diperbarui dalam satu detik; pelanggan memeriksa satu pesanan dengan nomor dan digit teleponnya. Ayu: server dan basis data; Bagus: klien dan protokol; Citra: GUI, uji, dan dokumen.

Tabel 15.7 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

Tabel 15.7: satu hasil per pekan, masing-masing diperiksa oleh orang lain selain pembuatnya. Sebuah daftar periksa untuk pekan sebelum presentasi menutup subbab itu. Buku: Subbab 15.11.10 dan 15.11.11, serta Tabel 15.8.

Profitina dalam Praktik

app.profitina.com adalah server Python (FastAPI di atas Uvicorn); setiap peramban, ponsel, dan tugas terjadwal adalah klien yang berbicara HTTP dan JSON.

`GET /api/me/dashboard` mengembalikan semuanya dalam satu jawaban, bukan selusin permintaan: Persamaan (15.5) yang diterapkan pada sebuah produk.

Umpan komunitas memakai **Server-Sent Events**; Cloudflare Tunnel menyediakan TLS; sebuah rilis dianggap berhasil hanya bila `/api/health` menjawab.

UU PDP diwujudkan menjadi fitur: hashing Argon2id, persetujuan, pembatasan laju login (Persamaan (15.6)), ekspor data, dan penghapusan akun. Aplikasi mengukur dan menyarankan; pengguna yang memutuskan.

Buku: Subbab 15.12.

Kesalahan yang Sering Terjadi

1. Lupa membilas. 2. Lupa baris baru. 3. Mengira satu `recv` sama dengan satu pesan. 4. Melayani satu klien pada satu waktu.

5. Keadaan bersama tanpa perlindungan di antara thread klien. 6. Menunggu selamanya. 7. "Address already in use": hentikan server lama, `SO_REUSEADDR`, port 0 dalam uji.

8. Pengodean yang tidak cocok. 9. Mengikat ke semua alamat selama pengembangan. 10. Rahasia di kode atau log.

11. Data pribadi di pesan galat dan log. 12. Menulis kode sebelum menyepakati protokol.

Buku: Subbab 15.13.

Ringkasan — Lima Hal

1. **Alamat dan port.** Sebuah koneksi adalah empat-serangkainya (Persamaan (15.2)); TCP adalah aliran bita yang andal, UDP datagram-datagram mandiri.
2. **Socket.** Socket pendengar menghasilkan satu socket tersambung per klien; bungkus alirannya, sebutkan UTF-8, bilas, dan bingkai dengan baris baru.
3. **Protokol di atas kertas lebih dulu**: tabel, diagram, dan objek tanpa I/O yang diuji tersendiri.
4. **Banyak klien**: penerima dan kumpulan thread; bagi sesedikit mungkin dan lindungi yang dibagi. Baris JSON dan sebuah proxy membuat Java dan Python saling bercakap; perjalanan pulang-pergi mendominasi biaya.
5. **Manusia.** Antarmuka tertulis, pengujian silang, Git dengan tinjauan, penanganan kegagalan yang disengaja, UU PDP sejak rancangan, dan demonstrasi yang sudah dilatih.

Buku: Subbab 15.14.

Pertemuan Berikutnya

Bab 16 — Evaluasi dan Pembahasan IV. Soal-soal yang dikerjakan tentang Bab 13 sampai 15 dan tentang seluruh buku, masing-masing diselesaikan lengkap dalam Java dan Python.

Iterator dan imutabilitas, thread dan deadlock, socket dan protokol: setiap perkakas dari tiga bab terakhir dipakai lagi, bersama-sama.

Pekerjaan rumah. Soal Latihan 15.1–15.13; mulailah dengan 15.1 (empat-serangkai dengan Persamaan (15.2)) dan 15.3 (membuat klien echo macet, lalu menyembuhkannya dengan batas waktu).

Penyelesaiannya ada di Subbab 15.16; cobalah setiap soal terlebih dahulu. Buku: Subbab 15.15 dan 15.16.

Kuliah 16 — Evaluasi dan Pembahasan IV: Iterator, Konkurensi, Jaringan, dan Keseluruhan Buku

Pemrograman Berorientasi Objek (EF234302)

Bab 16 buku ini dalam bentuk ringkas, untuk dipakai di kelas.

Buku: Irfan Subakti, *Pemrograman Berorientasi Objek dengan Java dan Python*.

Pengajar: Irfan Subakti — yifana@gmail.com

Hari ini tidak ada fitur bahasa baru: empat belas soal yang dikerjakan, delapan dari satu tugas proyek tim dan enam soal baru, masing-masing diselesaikan dalam Java dan Python serta diperiksa oleh ujinya sendiri. Buku: Bab 16, paragraf pembuka.

Agenda

1. **Sebuah metode** sepuluh langkah untuk proyek satu sistem utuh, dan tugas proyeknya dinyatakan ulang.
2. **Perencanaan dan protokol** (16.1–16.2): kontrak, penyimpan lelucon, protokol sebagai mesin keadaan.
3. **Membangun dan menguji** (16.3–16.8): server multiklien, uji silang dengan *stub*, dua puluh klien sekaligus, kegagalan, jendela yang tidak pernah membeku, dokumentasi yang diukur.
4. **Enam soal baru** (16.9–16.14): iterator, objek nilai, *race*, lini cuci, protokol buatan sendiri, dan proyek puncak.
5. **Pemantapan:** perbandingan soal, seluruh buku dalam satu tabel, ke mana setelah ini, kesalahan yang sering terjadi, ringkasan, dan kata penutup.

Setiap soal memiliki bentuk yang sama: soalnya, dengan bahasa kita sendiri, rancangan, penyelesaian Java dan Python, baris demi baris, penyelesaian keliru yang menggoda, dan uji diri. Buku: Subbab 16.1.

Titik Tolak Kita

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

Tabel 16.1 menamai perkakas Bab 13 sampai 15; menamai perkakas adalah separuh dari memilihnya. Sebagian besar program hari ini mencetak baris yang sama pada setiap jalan; beberapa yang mengukur waktu atau pembaruan yang hilang menyatakannya. Buku: Subbab 16.1 dan Tabel 16.1.

Metode untuk Satu Sistem Utuh

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`, protokol jaringan, dan format berkas adalah kontrak.

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

Tabel 16.2: tujuan kecil, cerita pengguna, bagian dengan pemilik, kontrak lebih dulu, protokol sebagai mesin keadaan, uji silang dengan stub, kerangka berjalan, konkurensi dan kegagalan dengan sengaja, ukur lalu dokumentasikan, tulis laporan sambil jalan. Buku: Subbab 16.2 dan Tabel 16.2.

Tugas Proyek

Bangun aplikasi klien–server dengan GUI yang berkomunikasi melalui socket, dalam tim beranggotakan paling banyak tiga orang: topik, pembagian kerja, spesifikasi dan antarmuka, protokol dan diagram alir, uji silang, uji *multithread*, penanganan server mati dan klien mati, dokumentasi.

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

*Topik kita adalah **Knock-Knock Club**: cukup kecil untuk dicetak utuh, dengan setiap bagian yang diminta tugas itu. Tabel 16.3 memetakan setiap tuntutan ke satu soal. Buku: Subbab 16.3 dan Tabel 16.3.*

Soal 16.1 — Kontrak Pertama

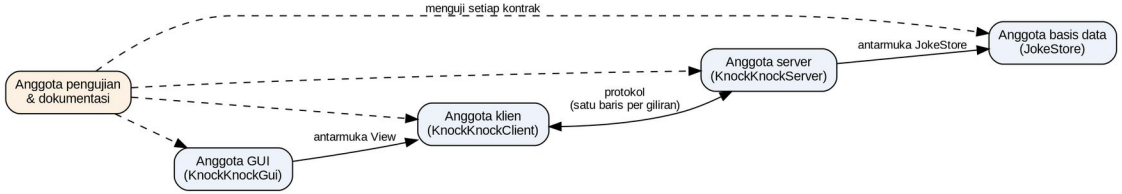
> **SOAL.** Pilih topiknya, bagi pekerjaannya, tulis spesifikasi setiap bagian, dan dokumentasikan antarmuka antara penyimpanan dan bagian lain begitu cermat sehingga anggota lain dapat mengimplementasikannya; tunjukkan dengan sebuah uji bahwa dua implementasi menepati janji yang sama.

Tujuan: pengguna menyambung, mendengar lelucon, menjawab di tempat yang tepat, dan boleh meminta lelucon lain. *Bukan tujuan:* akun, skor, lelucon karangan pengguna.

Kontrak penyimpanan: sedikitnya satu lelucon, bernomor **mulai dari 0** sampai `size() - 1`; indeks lain melemparkan `IndexOutOfBoundsException`. Menuliskan aturan penomoran tampak berlebihan. Justru itulah intinya.

Buku: Subbab 16.4.1.

Lima Bagian dan Kontraknya



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.

Pada Gambar 16.1: basis data (anggota 1), server dan klien (anggota 2), GUI serta pengujian dan dokumentasi (anggota 3).

Kontrak: `JokeStore`, protokol knock-knock, `converse`, `View`.

Kepemilikan berarti “saya yang membuatnya berjalan dan menjawab pertanyaan tentangnya”, bukan “orang lain tidak boleh menyentuhnya”. Buku: Subbab 16.4.1 dan Gambar 16.1.

Kontrak sebagai Antarmuka

Kode 16.1 `JokeStore.java`: kontrak penyimpanan beserta record `Joke` yang imutabel; Javadoc milik `get` *adalah* spesifikasinya.

```java
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();

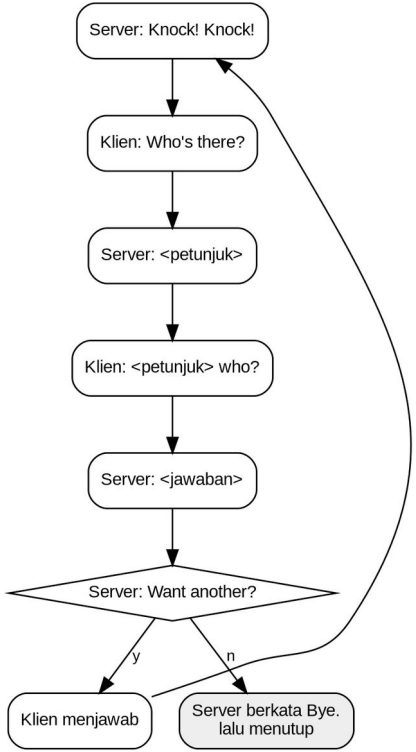
    /**
     * Mengembalikan lelucon bernomor index, dihitung mulai dari 0.
     *
     * @throws IndexOutOfBoundsException jika index tidak berada dalam 0 .. size()-1
     */
    Joke get(int index);
```

```
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
```

Pada Kode 16.1, `JokeStore.of` mengembalikan stub di memori sejak jam pertama. Penyelesaian keliru yang menggoda menomori mulai dari 1: uji kontrak menangkapnya, `get(0)` melemparkan eksepsi dan `get(4)` tidak. Kesalahan yang lebih dalam adalah sepakat hanya "secara lisan". Buku: Subbab 16.4.1.

Soal 16.2 — Protokol, Digambar

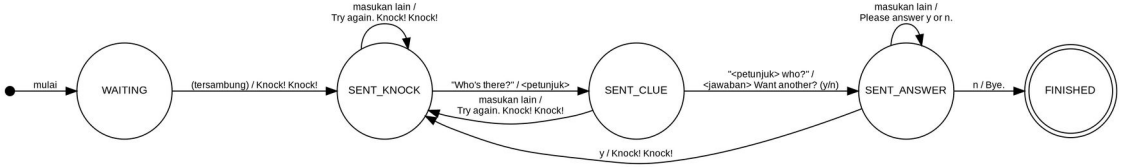
> **SOAL.** Jelaskan protokolnya dengan diagram alir: “Knock, knock”; “Who’s there?”; sebuah petunjuk; “*petunjuk* who?”; kalimat pamungkas; “Want more?”. Implementasikan sisi server sebagai kelas yang dapat diuji tanpa jaringan sama sekali.



Gambar 16.2 Protokol *knock-knock* sebagai diagram alir: percakapan bagi manusia, hanya dengan jawaban-jawaban yang diharapkan.

Gambar 16.2 adalah gambar yang baik untuk jalur bahagia, tetapi bagaimana jika klien berkata "Who is it?", mengembalikan petunjuk yang salah, atau menjawab "mungkin"? Buku: Subbab 16.4.2 dan Gambar 16.2.

Protokol sebagai Mesin Keadaan



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* mempunyai keadaan-keadaan, sebuah keadaan awal, dan fungsi transisi

$$\delta(s, x) = (y, s') \tag{16.1}$$

dari keadaan s dan baris klien x ke balasan y dan keadaan berikutnya s'.

Persamaan (16.1) tepat sama dengan tanda tangan `processInput`*. Pada Gambar 16.3 setiap keadaan memiliki panah untuk "hal lain". Pembingkaian: satu baris per giliran, sehingga kalimat pamungkas dan "Want another? (y/n)" dikirim bersama. Buku: Subbab 16.4.2 dan Gambar 16.3.*

Tabel Transisi

Tabel 16.4 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`

Tabel 16.4 adalah spesifikasi yang harus dicocokkan implementasinya baris demi baris; setiap keadaan memiliki baris untuk "hal lain", yang tidak diperlihatkan diagram alir. Buku: Subbab 16.4.2 dan Tabel 16.5.

Transisi dalam Kode

Kode 16.2 `KnockKnockProtocol.java`, `processInput`, paruh pertama: `WAITING`, `SENT_KNOCK`, dan `SENT_CLUE`.

```
    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;
                }
                state = State.SENT_KNOCK;
                return "You're supposed to say \"" + joke.clue() + " who?\"! Try again. " + KNOCK;
```

Pada Kode 16.2, dua field `state` dan `current` adalah seluruh ingatan percakapan. `"Who's there?".equalsIgnoreCase(input)` menaruh konstanta di depan, sehingga masukan `null` tidak dapat melemparkan eksepsi. Buku: Subbab 16.4.2.

Lelucon Lain, atau Selamat Tinggal

Kode 16.3 `KnockKnockProtocol.java`, `processInput`, paruh kedua: `(current + 1) % jokes.size()` memutar kembali ke lelucon pertama.

```
            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");
        }
    }
```

```
PASS finished after n
PASS second joke told
PASS wrong greeting
PASS wrong clue
PASS not finished
PASS stateless reply is fooled
```

Cabang `default` pada Kode 16.3 hanya tercapai sesudah `Bye.`: memanggil protokol pada saat itu adalah kesalahan pemrograman. Python memakai `Enum` dan rangkaian pernyataan `if`. Buku: Subbab 16.4.2.

Server Tanpa Ingatan

Kode 16.4 `KnockKnockProtocol.java`, `statelessReply` yang keliru tetapi menggoda: ia hanya melihat baris terakhir.

```
    /** 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;
    }
```

Kode 16.4 menjawab “Lettuce in, it’s cold out here!” untuk “Cabbage who?”, dan tidak dapat menceritakan lelucon kedua.

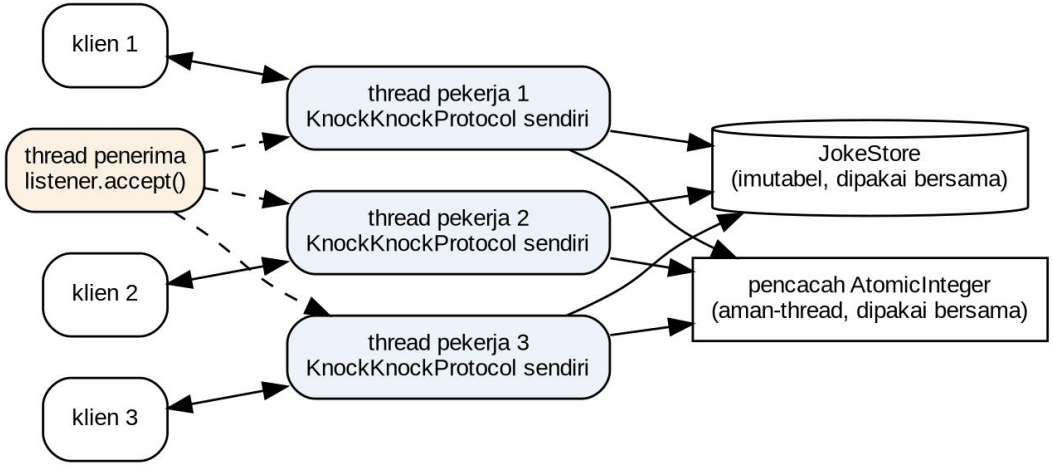
Bila balasan yang benar bergantung pada apa yang dikatakan sebelumnya, objek memerlukan **keadaan**.

> **UJI DIRI.** Tambahkan aturan: di `SENT_KNOCK`, `quit` langsung mengakhiri dengan `Bye.`. *Jawaban:* baris (`SENT_KNOCK`, `quit`, `Bye.`, `FINISHED`), dan tiga baris di dalam `case SENT_KNOCK`; klien tidak perlu diubah.

Buku: Subbab 16.4.2.

Soal 16.3 — Banyak Klien Sekaligus

SOAL. Bangun server dan kliennya. Server harus melayani beberapa klien pada waktu yang sama; tunjukkan bahwa klien kedua dilayani selagi klien pertama masih di tengah percakapannya.



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.

Pada Gambar 16.4, setiap klien mempunyai thread pekerja dan objek protokolnya sendiri; penyimpan lelucon (imutabel) dan pencacah (`AtomicInteger`) adalah satu-satunya objek bersama. Buku: Subbab 16.5.1 dan Gambar 16.4.

Terima, lalu Serahkan

Kode 16.5 `KnockKnockServer.java`, `acceptLoop`: satu thread per 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
        }
    }
```

Penyelesaian keliru yang menggoda, `serve(client)` di dalam perulangan, bekerja sempurna dengan satu klien. Dengan dua klien, klien kedua tersambung tetapi tidak ada yang melayaninya sampai percakapan pertama selesai.

Pada Kode 16.5, pengguna yang pergi membuat kopi di tengah lelucon akan memblokir seluruh klub. Buku: Subbab 16.5.1.

Satu Percakapan, Dapat Diuji dengan String

Kode 16.6 `KnockKnockServer.java`, `converse`: server berbicara lebih dulu; `null` dari `readLine()` berarti klien sudah pergi.

```
    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;
    }
```

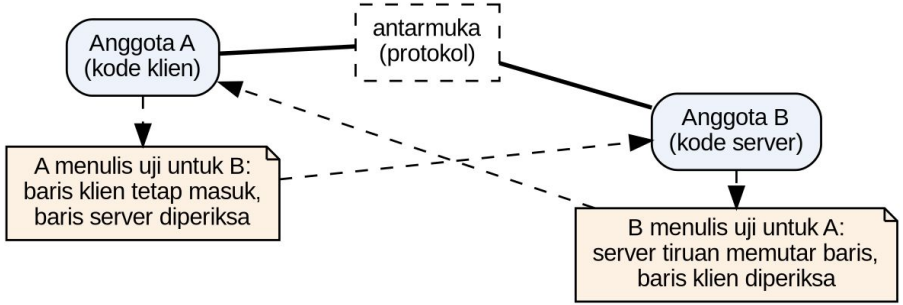
```
PASS ends with Bye.
PASS 13 lines
PASS B served while A talks
one-at-a-time server, client B got: (nothing within 500 ms)
PASS B starved by the wrong server
All 4 checks passed.
```

`converse` pada Kode 16.6 menerima reader dan writer, bukan socket, sehingga Soal 16.4 dapat mengujinya dengan string. Uji diri: sepuluh ribu klien dengan `newCachedThreadPool`? Pakai `newFixedThreadPool(200)`, atau thread virtual. Buku: Subbab 16.5.1.

Soal 16.4 — Setiap Pihak Menguji Pihak Lain

SOAL. Anggota A memakai fungsi anggota B, maka A merancang kasus uji untuk B, dan B untuk A. Terapkan pada klien dan server, dengan *stub*, sehingga uji-ujinya tidak memerlukan jaringan yang berjalan.

DEFINISI. *Stub* adalah pengganti minimal bagi komponen yang menjadi tumpuan kode yang diuji: jawaban tetap, tanpa logika sendiri.



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.

Pada Gambar 16.5, orang yang memakai sebuah kontrak paling tahu apa yang harus dilakukannya. Buku: Subbab 16.5.2 dan Gambar 16.5.

Stub yang Memutar Ulang Baris

Kode 16.7 `ProtocolContractTest.java`: `replay` adalah *stub*-nya; `serverSays` adalah sudut pandang anggota klien terhadap server.

```
/** 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;
}
```

Kode 16.8 `ProtocolContractTest.java`, `weakTest` yang keliru tetapi menggoda: ia hanya memeriksa bahwa klien mengatakan *sesuatu*.

```
/** 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();
}
```

Kode 16.8 meloloskan mutan yang berkata "Who is there?"; uji yang kuat membandingkan baris-baris persisnya dan menggagalkannya: mutation testing. Dengan Kode 16.7, `serverSays("Who's there?", "Cabbage who?")` berakhir dengan `(client left)`. Buku: Subbab 16.5.2.

Soal 16.5 — Dua Puluh Klien Bersamaan

> **SOAL.** Jalankan banyak klien pada saat yang sama terhadap satu server yang berjalan; periksa bahwa setiap klien menerima percakapan yang tepat dan server mencatat semuanya. Tunjukkan server yang lolos uji satu klien tetapi gagal pada uji ini.

Orakel: transkrip yang diharapkan dihitung dengan cara kedua yang independen, tanpa jaringan. **Gerbang start**: `CountDownLatch(1)` melepas semua klien bersamaan. **Batas**: 30 giliran, batas waktu, dan tenggat dua detik untuk hitungannya.

Buku: Subbab 16.5.3.

Gerbang Start

Kode 16.9 `ConcurrentClientsTest.java`, `correctConversations`: dua puluh tugas menunggu di gerbang, mulai bersamaan, dan dibandingkan dengan orakel.

```
    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;
```

Pada Kode 16.9, `startGate.countDown()` melepas semuanya dalam hitungan mikrodetik; setiap `Future<Boolean>` memegang putusan satu klien. Memeriksa berulang dengan tenggat adalah cara menguji sesuatu yang pada akhirnya menjadi benar. Buku: Subbab 16.5.3.

Satu Objek Protokol untuk Semua

Kode 16.10 `ConcurrentClientsTest.java`, server keliru yang menggoda: satu objek protokol dipakai bersama oleh setiap koneksi.

```
    /** 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());
```

```
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
```

Keadaan per koneksi milik koneksi itu. Yang dipakai bersama harus imutabel (penyimpannya) atau dilindungi (pencacahnya).

Uji diri: apakah `processInput` *yang* `synchronized` *menyelamatkan Kode 16.10? Tidak: pemanggilan tidak lagi tumpang tindih, tetapi ketukan A lalu salam B tetap merusak kedua percakapan. Buku: Subbab 16.5.3.*

Soal 16.6 — Ketika Satu Pihak Mati

SOAL. Ketika server mati, klien harus mengetahuinya; ketika klien mati, server harus mengetahuinya. Tangani galat koneksi sebagai eksepsi, dan tunjukkan bahwa server terus melayani klien lain.

Tabel 16.5 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

*Tabel 16.5: setiap pemanggilan yang menahan mendapat **batas waktu**; **akhir aliran adalah hasil yang normal**, `null` di Java dan `""` di Python. Buku: Subbab 16.6.1 dan Tabel 16.6.*

Setiap Kegagalan Menjadi Satu Kalimat

Kode 16.11 `FailureDetection.java`, `safeTalk`: menyambung dengan batas, membaca dengan batas, dan menangkap eksepsi dari yang khusus ke yang umum.

```
    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();
```

Pada Kode 16.11, `ConnectException` dan `SocketTimeoutException` adalah subkelas `IOException`; klausa umum yang ditulis lebih dulu membuat keduanya tidak terjangkau, sebuah galat kompilasi. Transkrip yang tidak berakhir dengan `S: Bye.` berarti server menutup terlalu dini. Buku: Subbab 16.6.1.

Runtuh, atau Satu Kalimat

```
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]
```

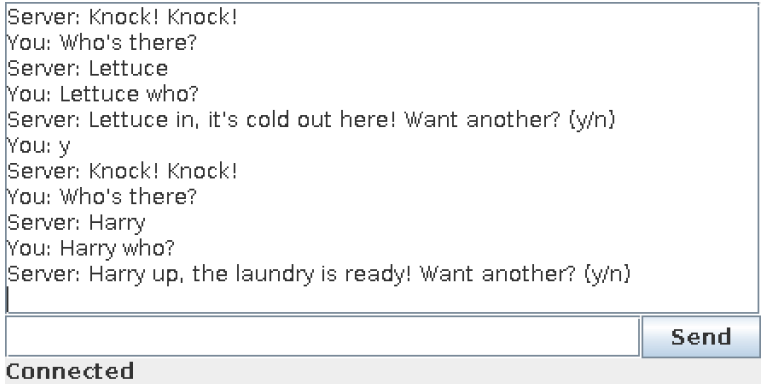
Kode 16.12 `FailureDetection.java`, `naiveTalk` yang keliru tetapi menggoda: `line.equals(...)` pada baris `null`.

```
    /** 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();
```

Kode 16.12 melemparkan `NullPointerException`; versi Python berputar pada `""` sampai `BrokenPipeError`. Uji diri: kabel yang dicabut tidak mengirim apa pun; beri server batas waktu baca, atau heartbeat. Buku: Subbab 16.6.1.

Soal 16.7 — Jendela yang Tidak Pernah Membeku

SOAL. Beri klien sebuah GUI yang menampilkan percakapan, mengirim setiap balasan, dan menampilkan status koneksi. Jendela harus tetap tanggap selagi jaringan menunggu, dan logikanya harus dapat diuji tanpa membuka jendela.



Gambar 16.6 Jendela Knock-Knock Club (Swing) selama percakapan. Baris status berbunyi “Connected”; kolom masukan aktif karena server sedang menunggu jawaban.

Gambar 16.6: thread yang menggambar jendela tidak boleh menunggu apa pun. MVC memberi strukturnya: `View` yang hanya menampilkan, dan controller yang bekerja di thread `network`-nya sendiri. Buku: Subbab 16.6.2 dan Gambar 16.6.

Thread Jaringan Mengirim ke Thread UI

Kode 16.13 `KnockKnockGui.java`, `connect`: percakapan berjalan di thread daemon bernama `network`.

```
        Thread connect(int port) {
            Thread network = new Thread(() -> talk(port), "network");
            network.setDaemon(true);
            network.start();
            return network;
        }
```

Kode 16.14 `KnockKnockGui.java`, `talk`: setiap perubahan tampilan melalui `ui.execute`; penantian jawaban pengguna terjadi di thread jaringan.

```
                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);
```

Pada Kode 16.13, `connect` langsung kembali. Pada Kode 16.14, `ui` adalah `SwingUtilities::invokeLater` di jendela dan `Runnable::run` dalam uji; Tkinter menguras sebuah `queue.Queue` setiap 50 ms. Buku: Subbab 16.6.2.

Diuji Tanpa Jendela

```
[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
```

Penyelesaian keliru yang menggoda menaruh kode socket di *listener* tombol **Send**: selagi ia menunggu, tidak ada yang digambar ulang. Pengukurannya memperlihatkan “repaint” tertahan 500 ms.

Uji diri: menghitung `String last` *sebelum* `ui.execute` *hanya menyerahkan nilai imutabel kepada thread lain, alasan yang sama yang membuat penyimpan lelucon imutabel. Buku: Subbab 16.6.2.*

Soal 16.8 — Janji yang Diukur

Kriteria kinerja memuat sebuah angka dan sebuah syarat. Throughput dan hukum Little:

$$X = \frac{n}{t} \tag{16.2}$$

$$L = X \cdot W \tag{16.3}$$

```
 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
```

Persamaan (16.2) menghitung percakapan per detik; Persamaan (16.3): 900/s dengan sekitar 5 ms berarti sekitar 4,5 yang sedang berlangsung. Uji diri: 1000/s dengan 20 berlangsung memberi $W = 0,02$ *s. Kriteria yang keliru adalah satu jalan yang beruntung ditulis sebagai fakta. Buku: Subbab 16.6.3.*

Dokumentasi dan Laporan

Tabel 16.6 Dokumentasi produk Knock-Knock Club, satu baris per dokumen.

Dokumen	Isi untuk Knock-Knock Club
Spesifikasi	tujuan, cerita pengguna pada Soal 16.1, Tabel 16.4
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 16.7
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)

Laporan: pendahuluan, spesifikasi, rancangan, implementasi, pengujian, evaluasi, kesimpulan, dan **kontribusi**: bagian yang dimiliki setiap anggota dan persentase yang disepakati.

Tabel 16.6 mendaftar dokumen-dokumennya; tabel pemecahan masalah di halaman berikut melengkapinya. Buku: Subbab 16.6.3 dan Tabel 16.7.

Pemecahan Masalah dengan Kata-Kata Pengguna

Tabel 16.7 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

Tabel 16.7 mendaftar gejala sebagaimana dilihat pengguna, beserta penyebab dan penawarnya; butir-butirnya berasal dari skenario kegagalan Soal 16.6. Buku: Subbab 16.6.3 dan Tabel 16.8.

Soal 16.9 — Jadwal dengan Iteratornya Sendiri

> **SOAL.** Tulis `PickupSchedule`, sebuah larik yang selalu terurut menurut jam (stabil untuk jam yang sama), dapat dipakai dalam perulangan for-each, dengan tampilan *malas* atas penjemputan di antara dua jam dan iterator yang *gagal cepat* bila jadwal berubah di tengah perulangan.

Rancangan. Field `slots`, `size`, dan `modCount`; iteratornya adalah kelas dalam `Cursor` dengan `next` sendiri dan salinan `modCount`; penambahan menggeser penjemputan yang lebih lambat ke kanan, seperti pada *insertion sort*.

Buku: Subbab 16.7.1.

Cursor Baru untuk Setiap Perulangan

Kode 16.15 `PickupSchedule.java`, `Cursor`: kelas dalam dengan posisinya sendiri dan pemeriksaan gagal-cepat.

```
    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++];
        }
    }
```

`iterator()` mengembalikan `new Cursor()` setiap kali, itulah yang membuat perulangan bersarang menghitung 25 pasangan. Kode 16.15 melemparkan `ConcurrentModificationException` ketika perulangan menghapus A4. Buku: Subbab 16.7.1.

Koleksi Bukanlah Iterator

Kode 16.16 `PickupSchedule.java`, `SelfIteratingSchedule` yang keliru tetapi menggoda: `iterator()` mengembalikan `this`.

```
    /** 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
        }
```

```
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
```

Perulangan bersarang atas n elemen mengunjungi

$$P_{\text{own}} = n^2, P_{\text{shared}} = n - 1 \tag{16.4}$$

pasangan: 25 dan 4 untuk $n=5$.

Persamaan (16.4) menjelaskan keluaran Kode 16.16. Uji diri: menghapus A1, elemen kedua dari belakang, mengakhiri perulangan tanpa suara: gagal-cepat hanyalah "upaya terbaik". Buku: Subbab 16.7.1.

Jadwal di Python

Kode 16.17 `pickup_schedule.py`: `__iter__` mengembalikan `_Cursor` baru; `between` adalah generator.

```
    def __iter__(self):
        return _Cursor(self)                    # posisi BARU untuk setiap gelung

    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
```

Kode 16.18 `pickup_schedule.py`, `_Cursor.__next__`: `StopIteration` mengakhiri perulangan.

```
    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
```

Objek generator dari Kode 16.17 adalah iterator: `len(list(view))` *bernilai 3 pertama kali dan 0 kali kedua;* `s.between(8, 12)` *membuat generator baru pada setiap pemanggilan. Kode 16.18 membangkitkan* `RuntimeError` *untuk jadwal yang berubah. Buku: Subbab 16.7.1.*

Soal 16.10 — Uang sebagai Nilai

SOAL. Tulis `Money`: jumlah dalam satuan bulat dan kode mata uang tiga huruf. Objek ini tidak pernah berubah; penjumlahan atau penetapan harga menghasilkan objek baru; jumlah yang sama bernilai sama dan berlaku sebagai kunci *hash* yang sama; mata uang yang berbeda ditolak.

DEFINISI. *Kontrak hash*: jika `a.equals(b)`, maka `a.hashCode() == b.hashCode()`. *Hash map* hanya mencari di ember milik kode *hash* kuncinya.

Rancangan. Kelas `final`, field `private final`, konstruktor privat di balik *factory* `of` yang memvalidasi, jumlah dalam `long`, tidak pernah `double`.

Buku: Subbab 16.7.2.

equals dan hashCode dari Field yang Sama

Kode 16.19 `Money.java`: `equals` membandingkan kedua field; `hashCode` dibangun dari field yang persis sama.

```
    @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
    }
```

```
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
```

Pada Kode 16.19, `a == b` salah (dua objek) dan `a.equals(b)` benar (satu nilai). Dua baris terakhir adalah varian yang keliru: tanpa `hashCode` sebuah set memuat dua elemen yang "sama"; kunci yang diubah menjadi hantu. Buku: Subbab 16.7.2.

Uang di Python

Kode 16.20 `money.py`: dataclass beku menuliskan `__eq__` dan `__hash__`; `__post_init__` memvalidasi.

```python
@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}")
```

Python menolak kesalahan pertama mentah-mentah: `__eq__` tanpa `__hash__` membuat kelas tidak dapat di-*hash*. `round` membulatkan setengah ke bilangan genap; tuliskan aturan pembulatan dalam spesifikasi.

Uji diri untuk Kode 16.20: IDR 5000 tidak sama dengan USD 5000; `rupiah(0)` diperbolehkan; `rupiah(-1)` melemparkan eksepsi, sehingga tidak pernah ada `Money` yang tidak sah. Buku: Subbab 16.7.2.

Soal 16.11 — Race yang Diukur

Empat mesin, empat thread, satu pencacah bersama. Total yang diharapkan dan pembaruan yang hilang:

$$N = T \cdot k \tag{16.5}$$

$$L_{\text{lost}} = N - c\,. \tag{16.6}$$

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.

Dengan $T = 4$ dan $k = 1.000.000$, Persamaan (16.5) memberi 4.000.000; Persamaan (16.6) mengukur kerusakannya. Gambar 16.7: kedua mesin membaca 41, keduanya menulis 42. Buku: Subbab 16.8.1 dan Gambar 16.7.

Melebarkan Jendela dengan Sengaja

Kode 16.21 `WashCounter.java`, `SlowUnsafeCounter`: celah antara baca dan tulis dibuka 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
        }
```

```
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
```

Kode 16.21 mengubah "gagal sebulan sekali" menjadi "gagal setiap kali": 50 dari 200. Pencacah tak aman yang `volatile` kehilangan 7.834 sampai 60.543 dari empat juta; angka-angka ini berubah-ubah antarjalan. Buku: Subbab 16.8.1.

Tiga Penawar, dan Satu Kejutan

Metode `synchronized`; objek kunci privat; `AtomicLong.incrementAndGet()`, sebuah *compare-and-swap*. Ketiganya mencapai tepat 4.000.000.

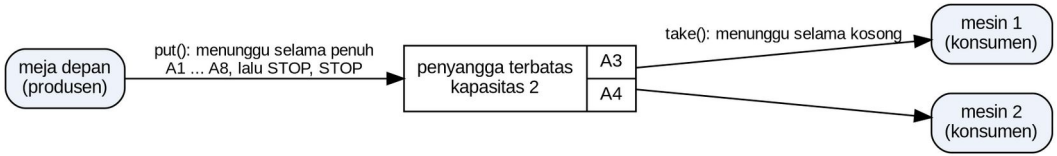
Di CPython, pencacah tak aman yang biasa *tidak* kehilangan apa pun: itu rincian implementasi, bukan janji. Pencacah lambat tetap membuktikan kodenya salah; `with self._lock:` membuatnya tepat.

> **PERINGATAN.** Kesimpulan yang menggoda, "`+=` atomik di Python", itu keliru. Bila dua thread memakai satu variabel dan salah satunya menulis, lindungilah, entah pernah ada uji yang gagal atau tidak.

Uji diri: dengan 4 thread yang berjalan seirama, setiap putaran menambah 1 alih-alih 4, sehingga 50 putaran memberi tepat 50. Buku: Subbab 16.8.1.

Soal 16.12 — Lini Cuci

SOAL. Meja depan menaruh pesanan di rak berkapasitas dua; dua mesin mengambilnya. Tulis raknya dengan `wait`/`notifyAll` (Python: `Condition`), lalu dengan antrean pemblokir; hentikan mesin-mesin dengan rapi; periksa bahwa setiap pesanan dicuci tepat sekali.



Gambar 16.8 Lini cuci: satu produsen, penyangga terbatas berkapasitas 2, dan dua konsumen. Dua pil `STOP`, satu per mesin, mengakhiri hari kerja.

Pada Gambar 16.8, satu pil `STOP` per mesin mengakhiri hari kerja. Buku: Subbab 16.8.2 dan Gambar 16.8.

Monitor Buatan Tangan

Kode 16.22 `WashLine.java`, `OrderBuffer`: `put` dan `take` menunggu di dalam perulangan `while` dan membangunkan semua dengan `notifyAll`.

```
    @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
    }

    @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;
    }
```

Pada Kode 16.22, `wait()` melepas kunci, tidur, lalu memperoleh kembali kuncinya; `while` memeriksa ulang sesudah bangun palsu atau bangun yang basi; `notify()` dapat membangunkan thread yang salah jenis. Dalam program sungguhan pakailah `ArrayBlockingQueue`. Buku: Subbab 16.8.2.

if Sebagai Ganti while

Kode 16.23 `WashLine.java`, `IfBuffer.take` yang keliru tetapi menggoda: syaratnya hanya diperiksa sekali.

```
        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 16.24 `wash_line.py`, `OrderBuffer.take`: cara yang benar dengan `threading.Condition`.

```
    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
```

```
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]
```

Dengan Kode 16.23, kedua mesin bangun untuk satu pesanan; yang kedua mengambil dari deque yang kosong. Kode 16.24 menunggu di dalam `while`. Uji diri: satu pil untuk dua mesin membuat satu mesin menunggu selamanya. Buku: Subbab 16.8.2.

Soal 16.13 — LAUNDRY/1

SOAL. Rancang `LAUNDRY/1`: server menyapa `LAUNDRY/1 READY`; klien mengirim `SERVICES`, `PRICE <service> <kg>`, atau `QUIT`; setiap permintaan mendapat tepat satu baris yang diawali `OK` atau `ERR`. Ujilah dengan sepuluh klien sekaligus.

Tabel 16.8 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 ...`

Pada Tabel 16.8, setiap permintaan berdiri sendiri, sehingga server tidak memerlukan keadaan percakapan: seluruh protokolnya adalah fungsi murni. Buku: Subbab 16.9.1 dan Tabel 16.9.

Protokol sebagai Fungsi Murni

Kode 16.25 `LaundryProtocol.java`, `reply`, kasus `PRICE`: satu cabang `ERR` untuk setiap baris tabel.

```
            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];
                }
                if (kg <= 0 || kg > 50) {
                    return "ERR weight must be in (0, 50] kg";
                }
                return "OK " + Math.round(rate * kg);
            default:
```

Kode 16.25: $7000 \times 3{,}5 = 24.500$ *dan* $14.000 \times 2 = 28.000$. *Sebagian besar protokol dapat diuji tanpa socket; uji jaringan memeriksa pembingkaian dan konkurensi. Buku: Subbab 16.9.1.*

Fungsi yang Sama di Python

Kode 16.26 `laundry_protocol.py`, `reply`: `split()` tanpa argumen memecah pada deretan spasi.

```
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}"
```

Kode 16.26 mengembalikan baris-baris yang persis sama dengan versi Java. Buku: Subbab 16.9.1.

Permintaan yang Tak Pernah Berangkat

```
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)
```

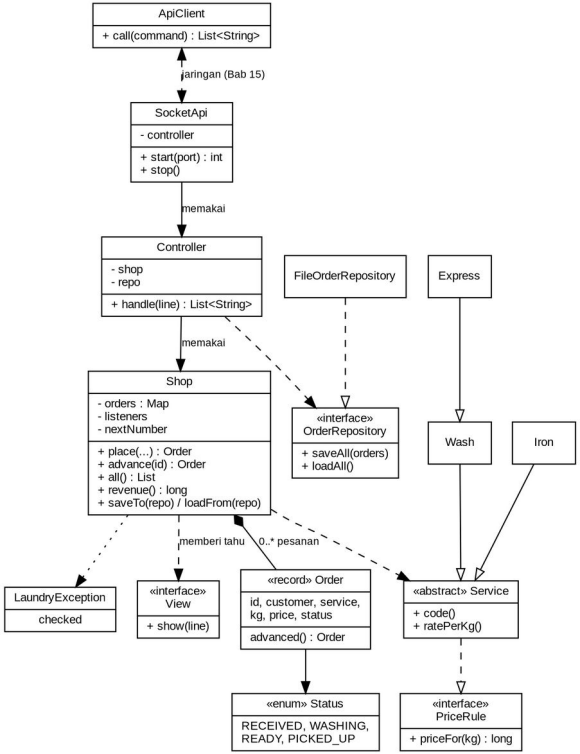
Kode 16.27 `LaundryProtocol.java`, `NoFlushClient` yang keliru tetapi menggoda: `PrintWriter` tanpa *autoflush*.

```
    /** 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));
        }
    }
```

Sepuluh klien menjumlah $7000 \times 55 = 385.000$. *Kode 16.27 menyimpan permintaan di penyangganya; hanya batas waktu yang menyelamatkannya dari menggantung. Uji diri:* `QUOTE WASH 3.5` *memberi* `OK 26950`*, dan hanya* `reply` *yang berubah. Buku: Subbab 16.9.1.*

Soal 16.14 — Proyek Puncak

SOAL. Sistem binatu yang kecil tetapi lengkap: pesanan dengan layanan dan berat, sebuah identitas dan harga, status dari diterima sampai diambil, laporan, berkas yang bertahan setelah dinyalakan ulang, beberapa kasir sekaligus melalui API socket; kelas, antarmuka, pewarisan, koleksi, eksepsi terperiksa, berkas, MVC, dalam kedua bahasa, beserta uji.



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.

Pada Gambar 16.9: `SocketApi`, `Controller`, `Shop` (model, dengan pengamat), dan balok-balok terkecil. `LIST` memiliki banyak baris balasan, sehingga setiap balasan diakhiri baris yang hanya berisi titik, aturan SMTP. Buku: Subbab 16.10.1 dan Gambar 16.9.

Cuci Kilat Adalah Cucian

Kode 16.28 `LaundryCore.java`, `Service`: *template method* `priceFor` yang dibangun dari dua metode abstrak.

```
    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 16.29 `LaundryCore.java`, `Express extends Wash`: dua kali tarif cuci, melalui `super`.

```
    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
        }
    }
```

Kode 16.28 adalah template method dari Bab 7; Kode 16.29 otomatis mengikuti setiap perubahan tarif cuci. Uji diri: dry cleaning adalah satu kelas baru dan satu entri map; `PLACE Ayu DRYCLEAN 1.5` *memberi* `OK L1 30000`*. Buku: Subbab 16.10.1.*

Satu Tempat Pesanan Berubah

Kode 16.30 `LaundryCore.java`, `Shop.advance`: `synchronized`, dan pesanan yang imutabel *diganti*, tidak pernah diubah.

```
    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;
    }
```

```
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
```

Tanpa `synchronized` pada Kode 16.30, dua thread dapat membaca `nextNumber` yang sama: pembaruan yang hilang dari Soal 16.11 di tempat yang baru. Lima klien menaruh dua puluh pesanan dengan dua puluh id yang berbeda. Buku: Subbab 16.10.1.

Satu Sesi di Meja Kasir

```
> 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
```

Setiap penolakan adalah baris `ERR`, tidak pernah thread koneksi yang mati. Klien keliru yang menggoda membaca satu baris per balasan; sesudah `LIST` ia tertinggal selamanya, dan `REVENUE` "mengembalikan" pesanan L2. Buku: Subbab 16.10.1.

Keempat Belas Soal Dibandingkan

Tabel 16.9 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

Bacalah kolom terakhir Tabel 16.9 dari atas ke bawah: hampir setiap penyelesaian keliru lolos dengan satu klien, satu thread, atau satu baris, dan gagal hanya di bawah syarat yang menjadi pokok soalnya. Buku: Subbab 16.11 dan Tabel 16.11.

Seluruh Buku dalam Satu Tabel

Tabel 16.10 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

Tabel 16.10 (Lanjutan) *Seluruh buku dalam satu tabel: pertanyaan setiap bab dan tempat jawabannya dipakai dalam bab ini.*

Bab	Pertanyaan yang dijawabnya	Dipakai dalam bab ini oleh
15	Bagaimana program berkomunikasi melalui jaringan?	Soal 16.3 sampai 16.8, 16.13, 16.14

Pada Tabel 16.10, pertanyaannya berganti watak di tengah jalan: dari "bagaimana menulis ini?" menjadi "bagaimana bagian-bagian ini bekerja sama, ketika beberapa hal terjadi sekaligus?". Buku: Subbab 16.12 dan Tabel 16.12.

Ke Mana Setelah Ini

Pola rancangan dengan namanya: Iterator, Observer, MVC, Strategy, Template method, Factory method, Repository, Producer–consumer, State. Penyelesaian bagi masalah, bukan tujuan.

Perkakas build: Maven (`pom.xml`, `mvn test`) dan Gradle; lingkungan virtual Python, pip, `pyproject.toml`, pytest; Git dan integrasi berkelanjutan sejak hari pertama.

Framework memanggil kode Anda: Spring Boot dan JavaFX; FastAPI, Django, Flask. Mempelajari salah satunya berarti mempelajari di mana ia menaruh potongan-potongan yang sudah Anda kenal.

Jalur bacaan: Bloch, Goetz dkk., Ramalho, Gamma dkk., Refactoring karya Fowler, The Mythical Man-Month karya Brooks. Buku: Subbab 16.13.

Profitina dalam Praktik

Kontrak: setiap sinyal pemindai berada di balik satu antarmuka, diperiksa oleh uji kontrak yang sama; lebih dari seribu uji menjadi gerbang setiap rilis.

Banyak klien: Server-Sent Events mendorong kiriman ke setiap peramban yang terbuka; `GET /api/me/dashboard` adalah satu permintaan, satu balasan lengkap.

Kegagalan ditangani dengan jujur: kiriman yang gagal mengembalikan teksnya; sebuah rilis dianggap berhasil hanya bila `/api/health` menjawab; tanpa model lokal, aplikasi mengatakannya.

Data pribadi di satu tempat, sesuai UU PDP 27/2022. Produk menyarankan; penggunanya yang memutuskan.

Buku: Subbab 16.14.

Kesalahan yang Sering Terjadi

Kontrak hanya secara lisan; protokol tanpa jawaban untuk setiap masukan; tanpa aturan pembingkaian; melayani klien satu per satu; memakai bersama keadaan per koneksi.

Uji yang tidak mungkin gagal; pemanggilan yang menahan tanpa batas waktu; melupakan akhir aliran; kode jaringan di thread UI; klaim kinerja dari satu kali jalan.

`iterator()` yang mengembalikan `this`; `equals` tanpa `hashCode`, kunci yang mutabel; "ujinya lolos, jadi tidak ada race"; `if` di sekitar `wait`; lupa membilas.

Setiap kesalahan menunjuk ke soal yang memperlihatkannya. Buku: Subbab 16.15.

Ringkasan — Lima Hal

1. **Kontrak lebih dulu**: antarmuka dan protokol ditulis sebelum kode, masing-masing dengan uji; protokol adalah mesin keadaan, Persamaan (16.1).
2. **Server**: satu thread dan satu keadaan per koneksi; uji silang dengan *stub*; banyak klien melalui gerbang start dibandingkan dengan orakel.
3. **Ketangguhan**: batas waktu, akhir aliran sebagai hasil normal, jendela yang tidak pernah menunggu, kriteria yang diukur dengan Persamaan (16.2) dan Persamaan (16.3).
4. **Iterator dan nilai**: iterator baru untuk setiap perulangan (Persamaan (16.4)); nilai imutabel dengan `hashCode` yang sejalan dengan `equals`.
5. **Thread dan protokol**: race menghilangkan pembaruan (Persamaan (16.6)); `while` di sekitar `wait`; protokol berupa fungsi murni yang membilas; proyek puncak menyatukan seluruh buku.

Buku: Subbab 16.16.

Kata Penutup

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 selagi dunia lambat.

> **WAWASAN KUNCI.** Tanyakan kepada setiap bagian sistem apa yang diketahuinya, apa yang dikerjakannya, siapa yang dipercayainya, dan apa yang dijanjikannya; tuliskan janji itu; ujilah sebelum memercayainya. Kebiasaan itu bukan milik satu bahasa saja.

Bangunlah sesuatu yang Anda pedulikan, bangunlah bersama orang lain, lalu bangunlah hal berikutnya sedikit lebih baik. Buku: Subbab 16.16, "Kata penutup".

Langkah Selanjutnya

Soal Latihan 16.1–16.12: `Countdown`, menghapus selama perulangan, membagi uang, race dengan tangan, deadlock antara dua mesin, `TOTAL`, *heartbeat*, penyimpan SQLite, pemecahan masalah, pencacah `READY`, rancangan uji, tata letak Maven.

Mulailah dengan 16.4 (hasil terkecil yang mungkin dari dua thread yang berlomba) dan 16.7 (*heartbeat* untuk protokol knock-knock).

Lalu proyek tim: pilih topiknya pada pertemuan pertama, dan tuliskan kontraknya sebelum kodenya.

Penyelesaian lengkap, dengan program Java dan Python, ada di Subbab 16.18. Buku: Subbab 16.17 dan 16.18.