Pemrograman Berorientasi Objek

Edisi Pertama

Irfan Subakti

JALUR PYTHON

Profitina
profitina.com

DAFTAR ISI

Kata Pengantar **7**

Pengantar OOP: Dari Prosedur menuju Objek **9**

1.1 Selamat Datang di Pemrograman Berorientasi Objek 9
1.2 Mengapa Program Prosedural Melampaui Batasnya Sendiri 10
1.3 Objek: Menyatukan Data dan Perilaku 10
1.4 Prosedural vs. Berorientasi Objek: Masalah Sama, Dua Cara Berpikir 11
1.5 Sekilas Empat Pilar 12
1.6 Menyiapkan Lingkungan Pengembangan Python Anda 13
1.7 Contoh Kerja: Class Python Pertama Anda 14
1.8 Menjalankan Program Anda 16
1.9 OOP dalam Praktik: Profitina 16
1.10 Ringkasan Bab dan Poin Penting 17
1.11 Soal Latihan 18
Lampiran 1.A — Log Verifikasi Eksekusi 18
Referensi 19

Anatomi Sebuah Class: Atribut, Constructor & Enkapsulasi **20**

2.1 Melanjutkan dari Bab 1 20
2.2 Atribut: Data Objek, Secara Formal 21
2.3 Constructor: Bagaimana Objek Dilahirkan 22
2.4 Satu Constructor, Argumen Default — Jawaban Python untuk Overloading 22
2.5 Property: Akses Baca Terkendali 23
2.6 Kontrol Akses: Empat Level Java vs. Konvensi Python 24
2.7 Contoh Kerja: Memperluas Book dengan Tahun Terbit 25
2.8 Menjalankan Program Anda 27
2.9 OOP dalam Praktik: Profitina 27
2.10 Ringkasan Bab dan Poin Penting 28
2.11 Soal Latihan 29
Lampiran 2.A — Log Verifikasi Eksekusi 29
Referensi 29

Tipe Bawaan, Alur Kontrol, String & Alat Debugging **31**

3.1 Melanjutkan dari Bab 2 31
3.2 Tipe Bawaan Python 31
3.3 Alur Kontrol: if/elif/else dan match 32
3.4 Loop: for dan while 33
3.5 String: Perbandingan Nilai dan Identitas 34
3.6 Alat Debugging: Mengetahui Apa yang Sebenarnya Dilakukan Program Anda 35
3.7 Contoh Kerja: Perhitungan Denda dan Status Perpanjangan untuk Book 36
3.8 Menjalankan Program Anda 38
3.9 OOP dalam Praktik: Profitina 38
3.10 Ringkasan Bab dan Poin Penting 39
3.11 Pertanyaan Tinjauan 39
Lampiran 3.A — Log Verifikasi Eksekusi 40
Referensi 40

Soal Latihan: Kelas, Field, Enkapsulasi & Dasar Bahasa **42**

4.1 Rekap: Kuliah 1–3 Secara Singkat 42
4.2 Cara Menggunakan Bab Latihan Ini 42

4.3 Soal Latihan....43
4.4 Format Kuis 1: Open-Book, Individu, Berwaktu....45
4.5 OOP dalam Praktik: Profitina....46
4.6 Ringkasan Bab....46
Lampiran 4.A — Checklist Self-Check (Tidak Dinilai)....47
Referensi....48

List & Kisah Membangun String....49
5.1 Dari Satu Buku ke Satu Rak Buku....49
5.2 List Python: Satu Wadah, Selalu Bisa Tumbuh....49
5.3 Operasi Inti List....50
5.4 Kisah Membangun String....50
5.5 Memperluas Pustaka: Kelas Library....51
5.6 Menjalankan Program....52
5.7 OOP dalam Praktik: Profitina....52
5.8 Ringkasan Bab dan Poin-Poin Utama....53
5.9 Pertanyaan Tinjauan....53
Lampiran 5.A — Log Verifikasi Eksekusi....54
Referensi....54

Penanganan Exception & File I/O....55
6.1 Dari Objek dalam Memori ke Program yang Bisa Gagal....55
6.2 Model Exception Python: Satu Kategori, Tanpa Penegakan Compiler....55
6.3 try, except, else, dan finally....56
6.4 Exception Kustom: BookNotFoundError....57
6.5 File I/O dengan Pernyataan with....57
6.6 Memperluas Pustaka: Persistensi dan Penghapusan yang Lebih Ketat....58
6.7 Menjalankan Program Anda....59
6.8 OOP dalam Praktik: Profitina....59
6.9 Ringkasan Bab dan Poin-Poin Utama....60
6.10 Pertanyaan Review....60
Lampiran 6.A — Log Verifikasi Eksekusi....60
Referensi....61

Interface & Kelas Abstrak....62
7.1 Rekap: Pustaka Sejauh Ini....62
7.2 Jawaban Python untuk Kelas Abstrak: ABC dan @abstractmethod....62
7.3 Jawaban Python untuk Interface: Protocol dan Typing Struktural....63
7.4 Membandingkan dengan Java, dan Memilih Antara ABC dan Protocol....64
7.5 Memperluas Pustaka: LibraryItem, Renewable, dan Tipe DVD Baru....64
7.6 Polimorfisme dalam Aksi: Program Driver....66
7.7 Menjalankan Program Anda....67
7.8 OOP dalam Praktik: Profitina....67
7.9 Ringkasan Bab dan Poin-Poin Utama....68
7.10 Pertanyaan Review....68
Lampiran 7.A — Log Verifikasi Eksekusi....69
Referensi....69

Soal Latihan: UTS — Meninjau Ulang Kuliah 1 Sampai 7....71
8.1 Rekap: Kuliah 1–7 Secara Singkat....71
8.2 Cara Menggunakan Bab Latihan Ini....71
8.3 Soal Latihan....72

8.4 Format UTS: Open-Book, Individu, Take-Home 74
8.5 OOP dalam Praktik: Profitina 75
8.6 Ringkasan Bab 75
Lampiran 8.A — Checklist Self-Check (Tidak Dinilai) 76
Referensi 77

Pewarisan & Polimorfisme 78
9.1 Rekap: Pustaka Sejauh Ini 78
9.2 Pewarisan, Secara Formal: Subclassing, super(), dan Kontrak Override 78
9.3 Anak Tangga Ketiga: ReferenceBook dan Pewarisan Multi-Tingkat 79
9.4 Polimorfisme, Secara Formal: Tipe Statis vs. Tipe Runtime 80
9.5 Setiap Objek Diam-Diam Mewarisi dari object: __eq__ dan __hash__ 81
9.6 Menggeneralisasi Library: Satu Koleksi, LibraryItem Apa Pun 82
9.7 Polimorfisme dalam Aksi: Program Driver 83
9.8 Mengkompilasi dan Menjalankan Program Anda 83
9.9 OOP dalam Praktik: Profitina 84
9.10 Ringkasan Bab dan Poin-Poin Utama 84
9.11 Pertanyaan Review 85
Lampiran 9.A — Log Verifikasi Eksekusi 85
Referensi 86

Generik & Kerangka Kerja Koleksi 87
10.1 Rekap: Pustaka Sejauh Ini 87
10.2 Kelas Generik yang Bisa Dipakai Ulang: Pair[A, B] 87
10.3 Fungsi Generik Berparameter Tipe Terbatas: total_fines[T: LibraryItem] 89
10.4 Satu Koleksi, Dua Indeks: _by_isbn Berdampingan dengan _items 89
10.5 Urutan Alami: @total_ordering dan __lt__ 91
10.6 set Berisi Tipe-Tipe yang Berbeda -- dan Mengapa Demo Mengurutkannya 91
10.7 Menyatukan Semuanya: Program Driver 92
10.8 Mengkompilasi dan Menjalankan Program Anda 93
10.9 OOP dalam Praktik: Profitina 93
10.10 Ringkasan Bab dan Poin-Poin Utama 94
10.11 Pertanyaan Review 94
Lampiran 10.A — Log Verifikasi Eksekusi 95
Referensi 96

Pemrograman GUI: Event & MVC 98
11.1 Rekap: Pustaka Sejauh Ini 98
11.2 Dari Konsol ke Jendela: Pemrograman Berbasis Event 98
11.3 Pola Model-View-Controller 99
11.4 View: LibraryView 100
11.5 Controller: Memasang Event 101
11.6 Titik Masuk Aplikasi: library_app.py 103
11.7 Menyatukan Semuanya: Memverifikasi GUI Sungguh Berjalan 103
11.8 Mengkompilasi dan Menjalankan Program Anda 104
11.9 OOP dalam Praktik: Profitina 105
11.10 Ringkasan Bab dan Poin-Poin Utama 106
11.11 Pertanyaan Review 106
Lampiran 11.A — Log Verifikasi Eksekusi 107
Referensi 108

Soal Latihan: Kuis 2 — Meninjau Ulang Kuliah 1 Sampai 11 109

12.1 Rekap: Kuliah 1–11 Secara Singkat 109
12.2 Cara Menggunakan Bab Latihan Ini 109
12.3 Soal Latihan 110
12.4 Format Kuis 2: Open-Book, Individu, Timed 113
12.5 OOP dalam Praktik: Profitina 114
12.6 Ringkasan Bab 115
Lampiran 12.A — Checklist Self-Check (Tidak Dinilai) 115
Referensi 116

Konkurensi: Thread, Race Condition, dan Sinkronisasi 117
13.1 Rekap: Pustaka Sejauh Ini 117
13.2 Mengapa Konkurensi: Thread, State Bersama, dan GIL 117
13.3 Race Condition: Bug Nyata yang Bisa Direproduksi 118
13.4 Sinkronisasi: Mutual Exclusion dengan threading.Lock 119
13.5 Thread Pool: ThreadPoolExecutor 120
13.6 Konkurensi dan Library: BorrowCounter sebagai Studi Kasus 121
13.7 Program Driver: Tiga Demo, Berturut-turut 122
13.8 Mengompilasi dan Menjalankan Program Anda 122
13.9 OOP dalam Praktik: Profitina 122
13.10 Ringkasan Bab dan Poin-Poin Utama 123
13.11 Pertanyaan Review 123
Lampiran 13.A — Log Verifikasi Eksekusi 124
Referensi 125

Networking: Socket, Klien, dan Server 126
14.1 Rekap: Pustaka Sejauh Ini 126
14.2 Model Client-Server 126
14.3 Membuka Socket yang Mendengarkan 127
14.4 Menangani Satu Klien: Protokol Berbasis Baris 128
14.5 Satu Thread Per Klien: Networking Bertemu Bab 13 128
14.6 Sisi Klien: library_client.py 129
14.7 Mengompilasi dan Menjalankan: Server dan Klien, Dua Program Terpisah 130
14.8 OOP dalam Praktik: Profitina 130
14.9 Ringkasan Bab dan Poin-Poin Utama 131
14.10 Pertanyaan Review 131
Lampiran 14.A — Log Verifikasi Eksekusi 131
Referensi 132

Design Pattern: Menamai Apa yang Sudah Dilakukan Pustaka 133
15.1 Rekap: Pustaka Sejauh Ini 133
15.2 Mengapa Design Pattern? 133
15.3 Factory Method: Memusatkan "Kelas Mana yang Dibangun" 134
15.4 Observer: Mengumumkan Perubahan Tanpa Tahu Siapa yang Mendengarkan 135
15.5 Singleton: Kisah Peringatan 136
15.6 Design Pattern dan Library: Sebuah Studi Kasus 139
15.7 Menjalankan Program 139
15.8 OOP dalam Praktik: Profitina 139
15.9 Ringkasan Bab dan Poin-Poin Utama 140
15.10 Pertanyaan Review 140
Lampiran 15.A — Log Verifikasi Eksekusi 141
Referensi 142

UAS: Proyek Capstone Client-Server Berbasis GUI dan Socket 143
16.1 Rekap: Kuliah 13–15 secara Ringkas 143
16.2 Cara Menggunakan Bab Latihan Ini 144
16.3 Proyek Capstone Final 144
16.4 Format UAS: Proyek Tim, Open-Book 147
16.5 OOP dalam Praktik: Profitina 148
16.6 Ringkasan Bab 148
Lampiran 16.A — Daftar Periksa Mandiri (Tidak Dinilai) 149
Referensi 150

Kata Pengantar

Buku ini tumbuh dari mata kuliah Pemrograman Berorientasi Objek yang saya ampu di Departemen Teknik Informatika, Institut Teknologi Sepuluh Nopember (ITS) Surabaya, dan disusun sebagaimana kuliahnya disusun: 16 bab, masing-masing satu sesi yang bisa Anda ikuti, kerjakan, lalu pakai. Ini adalah edisi Python; edisi pendampingnya membahas 16 bab yang sama dalam bahasa pemrograman lain.

Sebelum menulis satu class pun, Anda perlu melihat sendiri masalah yang membuat pemrograman berorientasi objek diciptakan. Buku ini membangun kasus itu dengan sistem perpustakaan kecil — Pustaka — yang versi proseduralnya mulai kewalahan seiring bertumbuh, lalu membangunnya ulang, konsep demi konsep, pertama dengan Java lalu dengan Python.

Pemrograman Berorientasi Objek (OOP) hadir sebagai dua edisi paralel — satu dengan Java, satu dengan Python — yang mengajarkan roadmap 16-langkah yang persis sama, hanya berbeda pada idiom bahasanya, sehingga pembaca yang sudah menguasai satu edisi bisa memakai edisi lainnya sebagai perbandingan langsung bagaimana gagasan yang sama tampil di tiap bahasa. Kuliah ini dibuka dengan menunjukkan mengapa program manajemen perpustakaan prosedural (Pustaka) mulai kewalahan seiring bertumbuh, lalu membangunnya ulang sebagai class dengan field, constructor, dan enkapsulasi; menambahkan array/list dan pembentukan string; membahas exception handling dan file I/O; memperkenalkan interface dan abstract class; lalu masuk ke inheritance, polymorphism, generics dan collections framework, pemrograman GUI dengan event dan MVC, konkurensi (thread, race condition, sinkronisasi), dan jaringan dengan socket. Buku ini ditutup dengan menamai, di bab Design Patterns, apa yang sebenarnya sudah dilakukan kode Pustaka sejak lama — dan berakhir dengan proyek puncak client-server GUI-socket.

Setibanya Anda di halaman terakhir, Anda semestinya mampu:

- Mengenali kapan dan mengapa program prosedural perlu menjadi berorientasi objek — bukan sekadar cara menulis sintaks class.
- Merancang class di sekitar field, constructor, dan enkapsulasi, dalam idiom Java maupun Python.
- Memakai inheritance, polymorphism, interface/abstract class, dan generics untuk membangun kode yang fleksibel dan dapat dipakai ulang.
- Menangani exception dan file I/O secara andal, serta membangun antarmuka grafis dengan struktur event-driven MVC.
- Menulis kode konkuren secara aman — thread, race condition, sinkronisasi — serta kode jaringan dengan socket, client, dan server.
- Mengenali design pattern umum sebagai nama untuk struktur yang sudah dituju sendiri oleh kode Anda.

Ketika buku ini perlu menunjuk sistem nyata, ia menunjuk Profitina — profitina.com dan app.profitina.com — platform yang saya rancang, bangun, dan jalankan sendiri. Pilihan itu praktis, bukan promosi: saya bisa menunjukkan keputusan yang sesungguhnya, trade-off yang sesungguhnya, dan akibat yang sesungguhnya, karena sayalah yang harus menanggungnya. Contoh-contoh itu bukan analogi yang dicocok-cocokkan; di mana ia muncul, ia adalah persoalan yang sama, dilihat dari sudut pandang pengajaran.

Untuk siapa buku ini ditulis: mahasiswa yang sudah mengambil kuliah Dasar Pemrograman atau Struktur Data dan bisa menulis kode prosedural, dan kini butuh desain berorientasi objek — dalam Java, Python, atau keduanya.

Sepatah kata tentang metode. Setiap contoh kode dalam buku ini dikompilasi dan dijalankan sebelum dituliskan, dan lampirannya memuat catatan verifikasi sebagai buktinya. Ketika sebuah angka muncul, angka itu berasal dari pengukuran, bukan dari ingatan. Bila Anda menemukan kekeliruan, atau bagian yang bisa lebih jelas, saya sungguh ingin mendengarnya — begitulah edisi berikutnya menjadi lebih baik.

Surabaya, September 2026

Irfan Subakti
yifana@gmail.com

Profitina
profitina.com

BAB 1

Pengantar OOP: Dari Prosedur menuju Objek

Sebelum menulis satu class pun, Anda perlu gambaran jelas tentang masalah yang ingin dipecahkan oleh pemrograman berorientasi objek — dan mengapa "tambah saja lebih banyak fungsi" pada akhirnya berhenti bekerja.

Tujuan Pembelajaran — setelah bab ini Anda mampu:

(1) Menjelaskan, dengan contoh konkret, mengapa program prosedural berskala besar menjadi sulit diubah dengan aman. (2) Mendefinisikan objek sebagai gabungan data (atribut) dan perilaku (method), serta menjelaskan enkapsulasi sebagai "menyembunyikan data di balik perilaku," ala Python. (3) Menyebutkan empat pilar OOP (enkapsulasi, pewarisan, polimorfisme, abstraksi) dan pratinjau di mana masing-masing diajarkan di mata kuliah ini. (4) Memasang dan memverifikasi interpreter Python yang berfungsi. (5) Membaca dan menjelaskan setiap baris class Python minimal beserta script driver-nya, lalu menjalankannya sendiri.

Satu Toolchain, Tiga Sistem Operasi — Windows 11, macOS, Ubuntu

OS	Instalasi sekali saja (Python 3.12+)	Jalankan (identik di mana pun)
Windows 11	winget install Python.Python.3.12 (atau installer python.org)	python main.py (py main.py juga bisa)
macOS	brew install python@3.12 (atau installer python.org)	python3 main.py
Ubuntu/Linux	sudo apt install python3 python3-pip	python3 main.py

Semua contoh di buku ini berjalan identik di Windows 11, macOS, dan Ubuntu/Linux; perintah hanya berbeda saat instalasi. Pilih baris Anda sekali dan ikuti terus di laptop Anda sendiri.

1.1 Selamat Datang di Pemrograman Berorientasi Objek

Selamat datang di mata kuliah Pemrograman Berorientasi Objek (OOP). Jika Anda sudah mengambil mata kuliah Dasar Pemrograman atau Struktur Data, Anda sudah tahu cara menulis fungsi, loop, list, dan kondisional — perangkat prosedural. Mata kuliah ini tidak membuang perangkat itu. Sebaliknya, mata kuliah ini mengajarkan cara berbeda mengorganisasi perangkat yang sama: alih-alih menyebar data ke berbagai list lepas dan mengoper data tersebut antar-fungsi yang berdiri sendiri, Anda akan belajar menyatukan data dan fungsi yang mengoperasikannya ke dalam satu unit bernama objek. Satu ide tunggal itu — data dan perilaku, disatukan, menyembunyikan detail internalnya sendiri — adalah benih tempat setiap konsep lain di mata kuliah ini tumbuh: enkapsulasi, pewarisan, polimorfisme, abstract base class, duck typing, dan akhirnya design pattern yang dipakai di seluruh perangkat lunak produksi nyata, termasuk platform yang namanya tertera di sampul buku ini.

Mata kuliah ini mengajarkan setiap konsep dua kali: sekali dalam Java (bahasa OOP bertipe-statis, standar industri) dan sekali dalam Python (edisi ini — bahasa bertipe-dinamis yang fitur OOP-nya lebih ringan namun secara konsep identik). Ini adalah edisi Python. Edisi Java pendamping memiliki peta jalan 16-lecture yang persis sama dan contoh kerja yang persis sama, diterjemahkan ke Java yang idiomatis — sehingga bahasa apa pun yang kelak dibutuhkan tim, pemberi kerja, atau proyek pribadi Anda, Anda sudah memahami ide dasarnya dan hanya perlu mempelajari sintaks baru, bukan konsep baru (Gambar 1.1).

Gambar 1.1 — Peta jalan 16-lecture OOP. Jalur Java dan Python sama-sama mengunjungi 16 pemberhentian yang persis sama; hanya idiom kode yang berbeda. Kita di sini: Lecture 1.

1.2 Mengapa Program Prosedural Melampaui Batasnya Sendiri

Bayangkan Anda membangun sistem perpustakaan kecil untuk ruang baca kampus — sebut saja Pustaka. Dalam gaya prosedural murni, Anda mungkin menyimpan koleksi sebagai beberapa list paralel: satu list judul, satu list penulis, satu list ISBN, dan satu list boolean yang mencatat apakah tiap buku sedang dipinjam. Setiap operasi — meminjam buku, mengembalikannya, memperpanjang, mencetak katalog — adalah fungsi bebas yang menerima keempat list tersebut sebagai parameter beserta indeks yang menunjukkan buku mana yang disentuh.

Ini berhasil, pada awalnya. Masalah muncul seiring program membesar. Tidak ada apa pun dalam bahasa yang mencegah fungsi mana pun, di mana pun dalam program seribu baris, untuk langsung menjangkau list on_loan dan mengubah satu entri menjadi True tanpa pernah memeriksa apakah buku itu sebenarnya tersedia, atau lupa memperbarui times_renewed saat peminjaman diperpanjang, atau memperbarui satu list tapi tidak tiga list paralel saudaranya setelah bug diperbaiki hanya di satu fungsi. Tidak ada satu tempat pun dalam kode yang menjamin "data sebuah buku hanya bisa berubah dengan cara yang masuk akal untuk buku nyata" — tanggung jawab itu tersebar di setiap fungsi yang kebetulan menyentuh list-list tersebut, dan setiap fungsi baru yang ditambahkan ke program adalah satu tempat lagi di mana jaminan itu bisa diam-diam rusak.

> **"Yang penting hati-hati" tidak bisa diandalkan pada skala besar**
>
> Reaksi pertama yang umum adalah: "programmer-nya saja yang perlu hati-hati agar tidak menyalahgunakan list." Ini berhasil untuk tugas mahasiswa 200 baris yang ditulis satu orang dalam satu waktu duduk. Ini nyaris selalu gagal untuk program yang dirawat lebih dari satu orang, disentuh lebih dari sekali, atau lebih besar dari beberapa ratus baris — yang menggambarkan hampir semua perangkat lunak nyata yang dipakai di dunia kerja. OOP tidak membuat kesalahan ceroboh menjadi mustahil, tetapi ia memberi bahasa itu sendiri konvensi dan alat — property, name-mangling, type hint yang diperiksa alat seperti mypy — yang membuat banyak kelas kesalahan jauh lebih mudah ditangkap, bukan sekadar meminta programmer mengingat untuk tidak melakukannya.

1.3 Objek: Menyatukan Data dan Perilaku

Gerakan inti pemrograman berorientasi objek adalah berhenti memperlakukan data sebuah buku (judul, penulis, ISBN, status pinjam) dan perilaku sebuah buku (pinjam, kembalikan, perpanjang)

sebagai hal terpisah yang tersebar di seluruh program. Sebaliknya, sebuah class mendefinisikan cetak biru — Book — yang menyatukan atribut data dan method perilaku menjadi satu unit, dan sebuah objek adalah satu instans konkret dari cetak biru tersebut, persis seperti satu buku fisik di rak adalah satu instans konkret dari ide umum "sebuah buku".

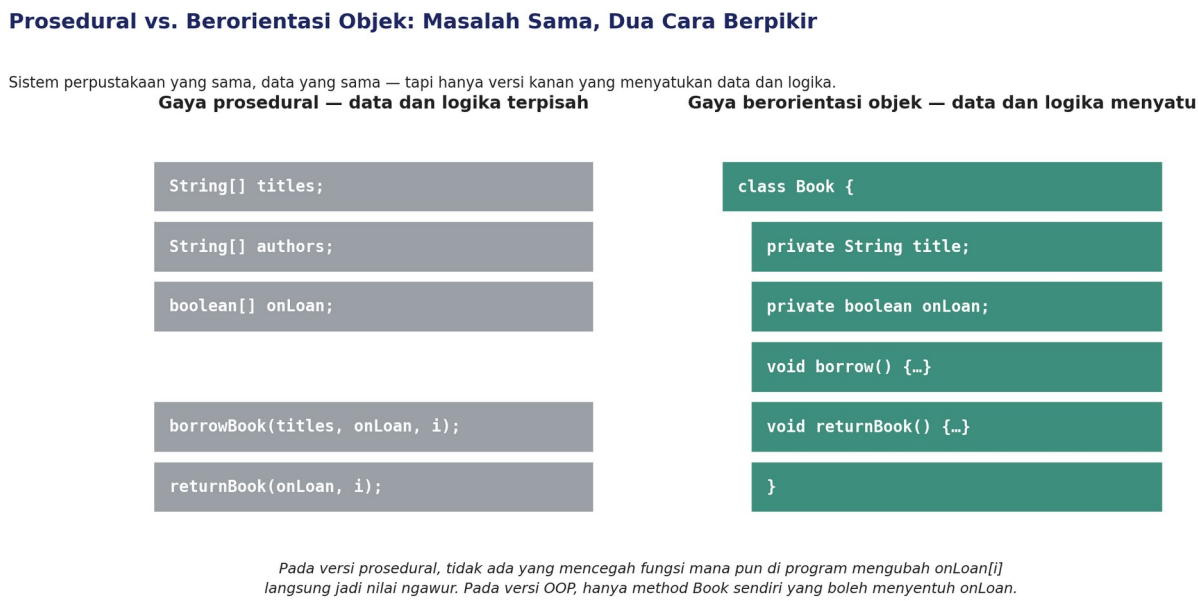
Gambar 1.2 — Buku nyata dan objek Book menyimpan informasi yang sama; objeknya tambahan hanya memperlihatkan operasi (borrow, return_book, renew) yang masuk akal dilakukan padanya.

Enkapsulasi, pratinjau — ala Python

Java menandai sebuah field sebagai private dan compiler-nya menegakkan aturan itu. Python tidak punya kata kunci private yang ditegakkan compiler; sebagai gantinya, satu garis bawah di depan (seperti _title, _on_loan pada objek Book di Gambar 1.2) adalah KONVENSI penamaan yang berarti "ini milik internal class itu sendiri — kode di luar seharusnya tidak menyentuhnya secara langsung," dan decorator @property tambahan bisa mengekspos tampilan terkendali, hanya-baca. Python mempercayai programmer untuk menghormati konvensi itu, alih-alih membuat interpreter menolak meng-compile pelanggarannya. Penyatuan-plus-penyembunyian ini tetap disebut enkapsulasi, dan ia adalah pilar pertama dari empat pilar OOP yang dibahas mata kuliah ini: enkapsulasi (Bab 2), pewarisan (Bab 9), polimorfisme (juga Bab 9), dan abstraksi lewat abstract base class serta protocol (Bab 7).

1.4 Prosedural vs. Berorientasi Objek: Masalah Sama, Dua Cara Berpikir

Gambar 1.3 menempatkan kedua gaya berdampingan untuk masalah perpustakaan Pustaka yang persis sama. Versi prosedural di kiri menyimpan empat list paralel dan sekumpulan fungsi bebas yang harus selalu diberi list yang benar dan indeks yang benar setiap kali. Versi berorientasi objek di kanan mendefinisikan satu class Book; setiap objek Book memiliki atribut title dan _on_loan-nya sendiri, dan menurut konvensi satu-satunya cara mengubah _on_loan adalah memanggil method yang didefinisikan dan bisa ditolak oleh Book itu sendiri.



Gambar 1.3 — Versi prosedural bisa dirusak dari mana pun dalam program. Versi berorientasi objek menyimpan data dan logika yang mengubahnya di satu tempat, secara konvensi dan desain.

Tidak ada satu gaya pun yang bisa melakukan sesuatu yang secara fundamental tidak bisa dilakukan gaya lainnya — OOP bukan jenis komputasi baru, melainkan cara berbeda mengorganisasi komputasi yang sama demi kejelasan, keamanan, dan ketahanan terhadap perubahan. Seiring program tumbuh melewati beberapa ratus baris dan melewati satu penulis saja, perbedaan organisasional itu menjadi pembeda antara basis kode yang bisa Anda perluas dengan aman selama bertahun-tahun dan basis kode di mana setiap perubahan berisiko merusak sesuatu yang jauh yang tidak pernah Anda sadari bergantung pada data lama yang tidak terlindungi.

1.5 Sekilas Empat Pilar

Mata kuliah ini disusun mengelilingi empat ide yang terus-menerus muncul dalam kode Java dan Python profesional. Bagian ini memberi pratinjau masing-masing dalam satu kalimat; masing-masing akan mendapat bab penuh nanti.

- Enkapsulasi (Bab 2) — menyatukan data dengan method yang mengoperasikannya, dan menyembunyikan data (lewat konvensi garis bawah dan @property) di balik method tersebut sehingga hanya bisa berubah dengan cara yang valid.
- Abstraksi lewat abstract base class & protocol (Bab 7) — mendeskripsikan APA yang bisa dilakukan sebuah objek ("ini bisa dipinjam") tanpa berkomitmen pada BAGAIMANA persisnya, sehingga jenis objek berbeda bisa dipakai secara bergantian di mana pun perilaku itu diharapkan.
- Pewarisan (Bab 9) — membiarkan satu class memakai ulang dan mengkhususkan atribut serta method class lain, sehingga ReferenceBook dan Textbook bisa berbagi hampir semua yang sudah didefinisikan Book umum.
- Polimorfisme (Bab 9) — memanggil nama method yang sama pada objek dari tipe konkret berbeda dan mendapatkan perilaku yang sesuai untuk tiap tipe, tanpa pemanggil perlu tahu tipe apa yang sedang dihadapinya. "Duck typing" dinamis milik Python membuat ini terasa sangat alami.

Mengapa urutan ini?

Setiap basis kode Java atau Python profesional yang akan Anda baca bersandar pada keempat pilar secara bersamaan, sehingga urutan sebuah mata kuliah memperkenalkannya adalah pilihan pengajaran, bukan fakta tentang bahasa. Mata kuliah ini mengajarkan enkapsulasi lebih dulu karena ketiga pilar lainnya, dalam arti nyata, dibangun di atasnya: Anda tidak bisa membicarakan secara bermakna tentang mengkhususkan perilaku objek (pewarisan/polimorfisme) atau mendeskripsikan perilaku secara abstrak (abstract base class) sebelum lebih dulu menerima bahwa data sebuah objek seharusnya disembunyikan di balik method-nya sendiri.

1.6 Menyiapkan Lingkungan Pengembangan Python Anda

Setiap contoh dalam buku ini ditulis untuk rilis Python 3 yang terkini. Per Agustus 2026, Python 3.14 adalah rilis stabil terbaru (pertama dirilis Oktober 2025), dengan dukungan aktif berlanjut hingga 2027 dan dukungan keamanan-saja hingga 2030; Python 3.13 juga masih dalam dukungan aktif. Kode buku ini sendiri ditulis dan dijalankan pada Python 3.11, dan semua yang ditampilkan berjalan tanpa perubahan pada 3.12, 3.13, dan 3.14, karena tidak ada materi mata kuliah ini yang bergantung pada fitur yang dihapus atau berubah di antara rilis-rilis tersebut. Jangan mengikuti tutorial yang ditulis untuk Python 2 — Python 2 mencapai akhir masa dukungannya pada Januari 2020 dan tidak kompatibel dengan sintaks yang dipakai mata kuliah ini (misalnya, print() sebagai fungsi, bukan pernyataan).

Alat	Rekomendasi	Alasan
Interpreter	CPython 3.12, 3.13, atau 3.14 (python.org)	Implementasi rujukan; gratis, dan yang diasumsikan hampir semua tutorial dan pustaka.
Editor — utama	Visual Studio Code (gratis) + ekstensi Python	Editor Python paling banyak dipakai di industri saat ini; debugging, linting, dan dukungan virtual-environment yang unggul.
Editor — alternatif	PyCharm Community Edition (gratis)	IDE khusus Python yang lengkap dengan alat refactoring kuat, alternatif gratis yang sepenuhnya mumpuni dibanding VS Code.
Alat paket/venv (bab lanjutan)	venv (bawaan) + pip	Diperkenalkan saat sebuah bab membutuhkan dependensi di luar pustaka standar (mis. pytest atau toolkit GUI di Lecture 11).

Anda tidak perlu memilih "satu editor yang benar"

Setiap listing kode dalam buku ini adalah source .py biasa yang bisa dijalankan dengan interpreter CPython standar — ia akan berjalan identik apa pun editor (atau tanpa editor sama sekali, dari terminal) yang Anda pakai untuk menulisnya. Pilih alat apa pun yang diwajibkan kelas, pemberi kerja, atau lab Anda; tidak ada apa pun di mata kuliah ini yang bergantung pada file proyek khas suatu editor.

Setelah Python terpasang, verifikasi dari terminal sebelum membuka editor apa pun — ini menangkap instalasi yang hilang atau salah konfigurasi sebelum bisa membingungkan Anda di dalam alat yang lebih kompleks:

```
$ python3 --version
Python 3.11.15
```

1.7 Contoh Kerja: Class Python Pertama Anda

Contoh kerja mata kuliah ini mengikuti kerangka lima-fase — Understand, Abstract, Design, Analyze, Implement — untuk setiap masalah yang tidak trivial, dimulai sekarang dengan yang paling sederhana: memodelkan satu buku di perpustakaan Pustaka.

Fase 1 — Understand (Pahami)

```
Masalah: pustakawan perlu mencatat, untuk tiap buku, judul, penulis,
dan ISBN-nya, plus apakah buku itu sedang dipinjam, serta mendukung
peminjaman, pengembalian, dan perpanjangan (maks 2x per peminjaman).
```

Fase 2 — Abstract (Abstraksi)

```
Lucuti semua hal spesifik tentang "perpustakaan" dan "buku", lalu
kenali bentuk umumnya: sejumlah DATA yang tidak boleh menjadi tidak
masuk akal secara internal (mis. "sedang dipinjam" tapi "diperpanjang
-3 kali"), plus sekumpulan kecil PERILAKU yang menjadi satu-satunya
cara sah mengubah data tersebut.
```

Fase 3 — Design (Rancang)

```
Atribut (garis bawah di depan = 'internal, jangan disentuh langsung'):
  _title, _author, _isbn : str    (tidak berubah setelah dibuat)
  _on_loan               : bool   (mulai False)
  _times_renewed          : int    (mulai 0, reset tiap peminjaman baru)
Method (cara konvensional kode luar memengaruhi objek ini):
  borrow()      -> bool   False jika sudah dipinjam
  return_book() -> None
  renew()       -> bool   False jika tak dipinjam atau sudah
                          diperpanjang dua kali
  __str__()     -> str    ringkasan mudah-dibaca, untuk dicetak
```

Fase 4 — Analyze (Analisis)

```
Argumen kebenaran: _on_loan hanya bisa menjadi True di dalam borrow(),
dan hanya saat sebelumnya False; hanya bisa menjadi False di dalam
return_book(). _times_renewed hanya bisa bertambah di dalam renew(),
dan maksimal 2, serta di-reset ke 0 setiap kali borrow() berhasil.
Selama kode pemanggil menghormati konvensi garis bawah (Bagian 1.3),
tidak ada kode di tempat lain dalam program yang mengubah atribut
ini kecuali lewat keempat method ini.
```

Fase 5 — Implement (Implementasikan)

```
class Book:
    def __init__(self, title: str, author: str, isbn: str) -> None:
        self._title = title
        self._author = author
        self._isbn = isbn
        self._on_loan = False
        self._times_renewed = 0

    def borrow(self) -> bool:
        if self._on_loan:
            return False
        self._on_loan = True
        self._times_renewed = 0
        return True

    def return_book(self) -> None:
        self._on_loan = False

    def renew(self) -> bool:
        if not self._on_loan or self._times_renewed >= 2:
            return False
        self._times_renewed += 1
        return True

    def __str__(self) -> str:
        status = (
            f"ON LOAN (renewed {self._times_renewed}x)"
            if self._on_loan else "on the shelf"
        )
        return (
            f'"{self._title}" by {self._author} '
            f'[ISBN {self._isbn}] - {status}'
        )
```

Sebuah class dengan sendirinya tidak melakukan apa-apa sampai sesuatu membuat objek darinya — Python menyebut ini instansiasi, dilakukan cukup dengan memanggil nama class seperti memanggil fungsi. Script driver berikut (modul terpisah yang berisi fungsi main()) membuat satu objek Book dan menjalankan perilakunya; ini kutipan yang dipersingkat untuk halaman ini — file lengkapnya, yang juga membuat buku kedua serta mendemonstrasikan renew() dan return_book(), disertakan sebagai Code/Python/Chapter01/library_demo.py bersama buku ini, dan keluaran lengkapnya itulah yang diverifikasi Lampiran 1.A:

```
from pustaka import Book

def main() -> None:
    clean_code = Book("Clean Code",
        "Robert C. Martin", "978-0132350884")

    print(clean_code)
    clean_code.borrow()
    print(clean_code)
    again = clean_code.borrow()
    print(f"Second borrow allowed? {again}")

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

Telusuri sendiri sebelum melanjutkan membaca

Apa yang dicetak program di atas pada baris keempatnya? Telusuri: print pertama menampilkan buku yang baru dibuat (di rak); borrow() lalu mengubah _on_loan menjadi True; print kedua mencerminkan itu; dan pemanggilan kedua ke borrow() — pada buku yang sudah dipinjam — harus mengembalikan False, karena argumen kebenaran Fase 4 menjamin _on_loan tidak bisa menjadi True untuk kedua kalinya tanpa return_book() di antaranya. Jika penelusuran Anda tidak sesuai dengan "Second borrow allowed? False", baca ulang satu pernyataan if di dalam borrow().

1.8 Menjalankan Program Anda

Python (biasanya) adalah bahasa yang diinterpretasikan: interpreter CPython membaca source .py Anda dan langsung menjalankannya, tanpa langkah compile-lalu-jalankan terpisah seperti yang dibutuhkan Java. Menjalankan python3 library_demo.py sekaligus memuat pustaka.py (lewat pernyataan import) dan menjalankan kode library_demo.py sendiri dalam satu langkah:

```
$ python3 library_demo.py
```

Sebelum benar-benar menjalankan sebuah script, praktik yang baik adalah setidaknya memeriksanya untuk kesalahan sintaks dengan python3 -m py_compile, yang mem-parsing file (dan menulis file bytecode ter-cache) tanpa menjalankannya — ini adalah padanan Python yang paling dekat dengan langkah compile javac terpisah milik Java, dan log verifikasi Lampiran mata kuliah ini memakainya sebagai pemeriksaan pertama sebelum benar-benar menjalankan tiap contoh:

```
$ python3 -m py_compile pustaka.py library_demo.py
$ echo $?
0
```

Jika Anda memakai editor seperti VS Code atau PyCharm, editor menjalankan python3 your_file.py untuk Anda di balik tombol "Run" — memahami apa yang sebenarnya terjadi di balik tombol itu akan membuat pesan error editor jauh lebih tidak misterius sepanjang mata kuliah ini.

1.9 OOP dalam Praktik: Profitina

> **Tentang kotak ini**
>
> Di seluruh buku ini, kotak seperti ini menghubungkan materi mata kuliah dengan Profitina, platform AI-visibility (Generative Engine Optimization) Indonesia yang nyata dan aktif, dibangun oleh penulis. Kotak ini menjelaskan DI MANA dan MENGAPA sebuah konsep dipakai dalam sistem nyata yang berjalan — bukan logika bisnis atau source code lengkap Profitina yang proprietary, yang tetap rahasia. Lihat dokumen pendamping "Profitina: A Non-Confidential Technical Overview" untuk gambaran lengkapnya.

Produk Profitina sendiri (app.profitina.com) ditulis dalam Python — bahasa yang persis sama dengan kode buku ini, memakai idiom enkapsulasi yang persis sama yang diperkenalkan bab ini. Backend Profitina mendefinisikan class seperti objek hasil-pindai yang menyatukan skor visibilitas sebuah bisnis bersama method yang diizinkan memperbarui skor tersebut, mengikuti konvensi garis-bawah-plus-property yang sama seperti ditunjukkan Bagian 1.7, alih-alih mengekspos skor itu sebagai atribut telanjang yang bisa ditimpa bagian mana pun dari basis kode. Model AI yang dijalankan sendiri (self-hosted) oleh Profitina (Qwen2.5 untuk analisis panjang, Qwen3:1.7b untuk klasifikasi cepat, keduanya lewat Ollama, sepenuhnya di server milik Profitina sendiri) sendiri dibungkus di balik class dan method milik Profitina — bagian lain aplikasi tidak pernah menjangkau melewati objek-objek itu untuk mengutak-atik internal model secara langsung. Ini persis disiplin class-Book pada Bagian 1.7, hanya diperbesar skalanya dari contoh pengajaran menjadi produk nyata yang dipakai bisnis sungguhan hari ini.

Anda akan melihat pola yang sama ini di setiap bab berikutnya: ide Python (atau Java) apa pun yang diperkenalkan sebuah bab, mata kuliah ini akan menunjukkan persis di mana ide setara muncul di dalam basis kode Profitina yang nyata, aktif, dan produksi — selalu pada level "begini bentuk ide itu dipakai," tidak pernah pada level detail implementasi proprietary.

1.10 Ringkasan Bab dan Poin Penting

- Program prosedural yang menyebar data ke berbagai list paralel tidak punya satu tempat pun yang memastikan data tetap masuk akal — fungsi mana pun di mana pun bisa merusaknya.
- Sebuah objek menyatukan data (atribut) dengan perilaku (method) yang diizinkan mengubah data tersebut; enkapsulasi dalam Python berarti menyembunyikan atribut di balik satu garis bawah di depan dan mengekspos akses terkendali lewat @property, karena Python tidak punya kata kunci private yang ditegakkan compiler.
- Empat pilar mata kuliah ini — enkapsulasi, abstraksi, pewarisan, polimorfisme — masing-masing akan mendapat bab tersendiri, dalam urutan itu, karena setiap pilar berikutnya dibangun di atas disiplin yang ditetapkan pilar sebelumnya.
- Python adalah bahasa yang diinterpretasikan: CPython menjalankan source .py secara langsung. Pakai Python 3.12, 3.13, atau 3.14 (semuanya terkini per 2026), dan VS Code atau PyCharm sebagai editor Anda.
- Kerangka Lima-Fase (Understand, Abstract, Design, Analyze, Implement) yang dipakai untuk Book di bab ini akan dipakai untuk setiap contoh kerja tidak-trivial di mata kuliah ini.

"Saya yang menciptakan istilah 'object-oriented', dan saya bisa pastikan saya tidak sedang memikirkan C++." — sintaks bahasa yang akan Anda pelajari di mata kuliah ini adalah nomor dua dibanding model mental yang diperkenalkan bab ini: program sebagai objek-objek yang saling berkomunikasi, masing-masing bertanggung jawab melindungi datanya sendiri.

1.11 Soal Latihan

Soal-soal ini untuk belajar mandiri dan diskusi kelas; tidak dinilai. Kerjakan sebelum Lecture 2.

1. Dengan kata-kata Anda sendiri, jelaskan mengapa "programmer-nya saja yang perlu hati-hati" tidak dianggap jawaban memuaskan untuk masalah korupsi data yang dijelaskan di Bagian 1.2.
2. Atribut _times_renewed pada Book memakai satu garis bawah di depan. Tulis satu kalimat menjelaskan konvensi apa yang disinyalkan ini kepada programmer lain, dan mengapa Python mempercayai programmer untuk menghormatinya alih-alih membuat interpreter menegakkannya.
3. Modifikasi (di kertas, atau langsung di editor Anda) class Book untuk menambahkan method extend_due_date(self, days: int) yang hanya boleh berhasil jika buku sedang dipinjam. Di bagian class mana aturan ini harus ditegakkan, dan mengapa?
4. Gambar 1.3 menunjukkan versi prosedural dengan empat list paralel. Misalkan field baru, due_date, perlu ditambahkan. Daftarkan setiap tempat pada versi prosedural yang harus berubah, dibandingkan setiap tempat pada versi berorientasi objek yang harus berubah.
5. Pilar mana dari empat pilar (enkapsulasi, abstraksi, pewarisan, polimorfisme) yang didemonstrasikan contoh Book bab ini? Tiga pilar mana yang sengaja BELUM didemonstrasikan, dan mengapa (petunjuk: baca ulang Bagian 1.5)?

Lampiran 1.A — Log Verifikasi Eksekusi

File lengkap pustaka.py dan library_demo.py (Code/Python/Chapter01/, disertakan bersama buku ini — superset dari kutipan yang dipersingkat pada Bagian 1.7, juga mendemonstrasikan buku kedua serta renew() dan return_book()) diperiksa sintaksnya dan dijalankan pada CPython 3.11.15 sebelum bab ini ditulis, untuk memastikan kodenya benar, bukan sekadar "terlihat benar". Keluaran direproduksi verbatim di bawah.

```
$ python3 -m py_compile pustaka.py library_demo.py
(tidak ada error)

$ python3 library_demo.py
--- Pustaka: starting state ---
"Clean Code" by Robert C. Martin [ISBN 978-0132350884] - on the shelf
"The Pragmatic Programmer" by Hunt & Thomas [ISBN 978-0135957059] - on the shelf

--- A student borrows Clean Code ---
Borrow succeeded? True
"Clean Code" by Robert C. Martin [ISBN 978-0132350884] - ON LOAN (renewed 0x)

--- Someone else tries to borrow the SAME copy ---
Borrow succeeded? False (already on loan, so the object refuses)

--- The student renews, then returns it ---
"Clean Code" by Robert C. Martin [ISBN 978-0132350884] - ON LOAN (renewed 1x)
"Clean Code" by Robert C. Martin [ISBN 978-0132350884] - on the shelf
```

Referensi

[1] A. Kay. Surel ke milis pengembang Squeak, 2003 (banyak dikutip sebagai "I made up the term object-oriented, and I can tell you I did not have C++ in mind").

[2] Python Software Foundation. "Status of Python Versions." https://devguide.python.org/versions/ (diakses Agustus 2026) — mengonfirmasi Python 3.14 sebagai rilis stabil terbaru (Oktober 2025), 3.13 masih dalam dukungan aktif.

[3] R. C. Martin. Clean Code: A Handbook of Agile Software Craftsmanship. Prentice Hall, 2008 (sumber judul contoh yang dipakai bab ini).

[4] Microsoft. "Visual Studio Code — Python." https://code.visualstudio.com/docs/languages/python (diakses Agustus 2026).

[5] JetBrains. "PyCharm." https://www.jetbrains.com/pycharm/ (Community Edition, diakses Agustus 2026).

BAB 2

Anatomi Sebuah Class: Atribut, Constructor & Enkapsulasi

Bab 1 memberi tahu Anda bahwa objek menyatukan data dengan perilaku. Bab ini membuka tudungnya: persis bagian mana dari sebuah class yang mendeklarasikan data itu, persis bagian mana yang membangun objek baru, dan persis seberapa banyak data itu yang boleh dilihat dunia luar.

Tujuan Pembelajaran — setelah bab ini Anda mampu:

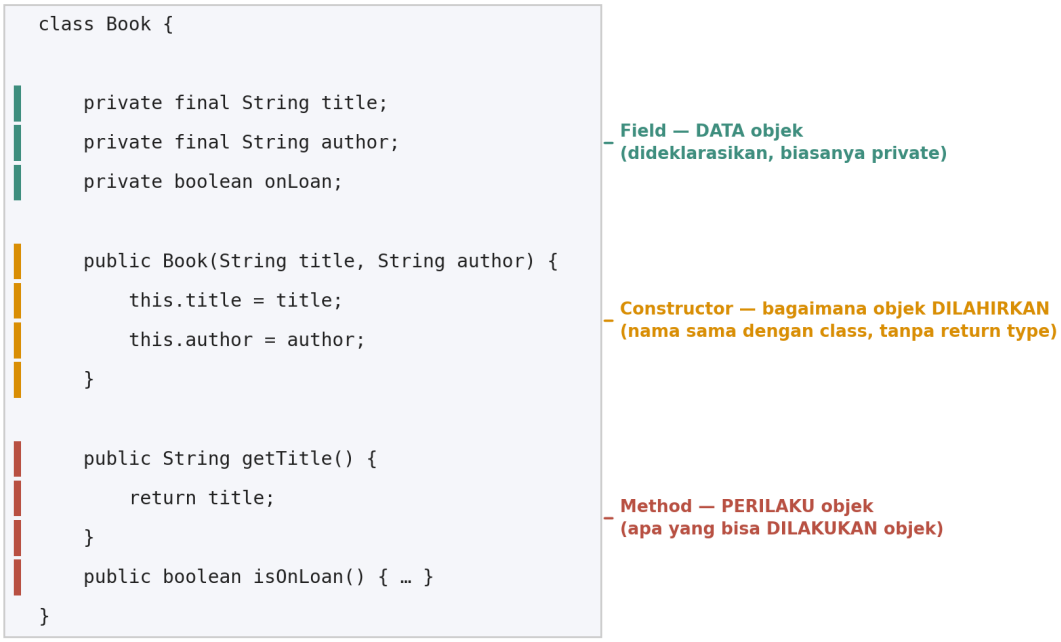
(1) Mengidentifikasi tiga jenis anggota sebuah class — atribut, constructor, method — dan menyatakan fungsi masing-masing. (2) Menulis __init__, dan menjelaskan apa yang dirujuk "self" di dalamnya. (3) Memakai nilai parameter default di __init__ agar satu constructor mencakup kasus yang di Java membutuhkan constructor overloading. (4) Menulis getter @property yang mengekspos atribut objek untuk dibaca tanpa mengizinkan kode luar merusaknya. (5) Menyebutkan empat access modifier Java, dan menjelaskan apa yang dipakai Python sebagai gantinya, alih-alih kontrol akses yang ditegakkan compiler. (6) Memperluas class Book dari Bab 1 dengan atribut baru, constructor yang lebih kaya, dan cakupan property lengkap, lalu memeriksa sintaksnya, menjalankan, dan memverifikasinya sendiri.

2.1 Melanjutkan dari Bab 1

Bab 1 memperkenalkan satu ide tunggal yang menjadi fondasi setiap bab berikutnya: objek menyatukan data (atribut) dengan perilaku (method) yang diizinkan mengubah data itu, dan menyembunyikan atribut di balik perilaku tersebut. Anda melihat satu class, Book, dengan tiga atribut utama dan empat method, dan Anda menjalankannya. Bab ini tidak memperkenalkan ide baru sebanyak ia membedah class Book yang sama itu bagian demi bagian dan mengamati dari dekat masing-masing dari tiga jenis anggotanya — atribut, constructor, dan method — dimulai dari jenis anggota yang terus-menerus dipakai Bab 1 namun tidak pernah dijelaskan: constructor.

Anatomi Sebuah Class: Field, Constructor, dan Method

Setiap class Java atau Python yang Anda tulis punya (paling banyak) tiga jenis anggota ini, dalam urutan konvensional ini.



Gambar 2.1 — Setiap class yang Anda tulis punya (paling banyak) tiga jenis anggota ini, dalam urutan konvensional ini: field/atribut, lalu constructor, lalu method. (Ditampilkan di sini dengan sintaks Java sebagai ilustrasi lintas-bahasa; bentuk setara Python sendiri muncul di seluruh bab ini.)

2.2 Atribut: Data Objek, Secara Formal

Atribut adalah nama yang diikat ke sebuah nilai pada self di dalam method sebuah class (paling umum di dalam __init__) — di sanalah data sebuah objek hidup. Book pada Bab 1 menetapkan lima atribut di dalam __init__: _title, _author, _isbn, _on_loan, dan _times_renewed. Setiap objek yang dibangun dari class Book mendapat salinan independennya sendiri dari kelima atribut ini; dua objek Book yang berbeda tidak pernah berbagi nilai _title yang sama, meskipun keduanya ditetapkan oleh baris source code yang persis sama di dalam __init__ (Gambar 2.1).

Perhatikan bahwa _title, _author, dan _isbn tidak pernah ditetapkan ulang di mana pun di luar __init__ pada class Book mata kuliah ini — menurut konvensi, ketiganya dimaksudkan untuk bersifat efektif-immutable setelah objek dibangun, maksud yang sama yang diekspresikan Java dengan kata kunci final. Python tidak punya kata kunci final dan tidak punya compiler yang menegakkan "ini tidak bisa ditetapkan ulang"; garis bawah di depan ditambah tidak adanya @x.setter yang sepadan adalah seluruh mekanismenya, dan ia bergantung pada programmer lain menghormati konvensi itu, bukan interpreter yang menolak meng-compile pelanggarannya. _on_loan dan _times_renewed sengaja dirancang untuk berubah, karena seluruh tujuan mereka adalah melacak status buku seiring dipinjam, diperpanjang, dan dikembalikan.

private + final milik Java, berdampingan dengan konvensi Python

Java menandai title, author, dan isbn sebagai private final: private ditegakkan compiler (kode di luar class bahkan tidak bisa mencoba membaca atau menulis field itu langsung), dan final adalah jaminan kedua yang independen dan ditegakkan compiler bahwa field itu tidak bisa ditetapkan ulang setelah konstruksi. _garis_bawah_di_depan Python menyinyalkan MAKSUD yang sama seperti private, tapi interpreter tidak menegakkan apa pun darinya — sama sekali tidak ada padanan Python untuk final. Ini bukan kekurangan yang khas Python; ini mencerminkan filosofi desain yang lebih luas ("kita semua orang dewasa yang menyetujui di sini") yang mempercayai disiplin programmer alih-alih penegakan compiler, dibahas lebih lanjut di Bagian 2.6.

2.3 Constructor: Bagaimana Objek Dilahirkan

Sebuah class dengan sendirinya hanyalah cetak biru; tidak ada yang eksis sampai kode memanggil nama class seperti fungsi, dan pemanggilan itu selalu menjalankan __init__. Tugas constructor tunggal: mengambil nilai mentah apa pun yang diberikan pemanggil dan memakainya untuk membawa satu objek baru ke keberadaan dalam keadaan valid, sepenuhnya terinisialisasi — tidak pernah objek setengah-jadi dengan beberapa atribut ditetapkan dan yang lain sekadar tidak ada.

Dalam Python, __init__ adalah method dengan dua sifat tidak lazim: parameter pertamanya selalu self, merujuk pada "objek yang sedang dibangun sekarang," dan ia mengembalikan None secara implisit (sebuah __init__ yang mengembalikan apa pun selain itu memicu TypeError). Setiap parameter lain diberikan pemanggil. Di dalam __init__, self.title = title secara tidak ambigu berarti "atribut milik objek ini mendapat nilai dari parameter" — tidak ada tabrakan penamaan yang perlu dipecahkan seperti this. milik Java, karena self. selalu wajib dipakai untuk menjangkau atribut; title telanjang di dalam __init__ hanya bisa merujuk pada parameter, tidak pernah pada sesuatu di objek.

Lupa self. adalah versi Python dari bug klasik pertama

Jika Anda menulis title = title di dalam __init__ alih-alih self._title = title, Python tidak memicu error — ia membuat (atau menetapkan ulang) sebuah variabel lokal biasa bernama title yang lenyap begitu __init__ selesai, dan self sama sekali tidak pernah mendapat atribut _title. Baris berikutnya yang mencoba membaca self._title memicu AttributeError, tapi seringkali tidak sampai method lain dipanggil jauh kemudian, yang bisa membuat akar masalahnya sulit dilacak balik ke self. yang hilang. Kesalahan setara di Java (Bagian 2.3 edisi Java) gagal sama diam-diamnya pada titik penetapan, tapi karena alasan struktural yang berlawanan — Java punya tabrakan penamaan yang justru dicegah sepenuhnya oleh keharusan self. milik Python.

2.4 Satu Constructor, Argumen Default — Jawaban Python untuk Overloading

Pemanggil nyata tidak selalu punya informasi yang sama tersedia. Terkadang Anda tahu tahun terbit sebuah buku; terkadang tahunnya sekadar tidak tercatat di mana pun yang bisa Anda akses. Java menangani ini dengan constructor overloading — beberapa constructor, dibedakan berdasarkan daftar parameternya (lihat Bagian 2.4 dan Gambar 2.2 edisi Java untuk gambaran lengkapnya). Signature method Python tidak mendukung overloading sejati — hanya ada tepat satu __init__ per

class — tapi nilai parameter default mencakup kasus penggunaan yang persis sama dengan kode yang lebih sedikit.

def __init__(self, title, author, isbn, publication_year=0): memberi publication_year nilai default 0, berarti "tidak tercatat." Memanggil Book(t, a, i) dan Book(t, a, i, 2008) keduanya bekerja, dan keduanya menjalankan badan __init__ yang persis sama — tidak ada constructor kedua yang perlu disinkronkan, tidak ada delegasi ala this(...) untuk dipelajari, dan tidak ada risiko kedua "versi" itu tidak-sinkron seperti yang bisa terjadi pada kedua constructor overload Java jika seseorang mengedit satu dan lupa yang lain. Ini salah satu tempat di mana fleksibilitas dinamis Python benar-benar menyederhanakan pola yang di Java membutuhkan fitur bahasa khusus (overloading) untuk diekspresikan.

> **Constructor overloading, sebagai perbandingan (Java)**
>
> Java akan menulis ini sebagai DUA constructor: Book(title, author, isbn), yang badan satu-barisnya this(title, author, isbn, 0);, dan Book(title, author, isbn, publicationYear), yang melakukan penetapan lima-atribut sebenarnya. Yang pertama mendelegasikan ke yang kedua via constructor chaining, sehingga logika penetapan eksis di tepat satu tempat. __init__ tunggal Python dengan argumen default mencapai jaminan yang sama — satu tempat penetapan terjadi — lewat fitur parameter-default bahasa itu, alih-alih pemanggilan chaining.

2.5 Property: Akses Baca Terkendali

Menyembunyikan atribut di balik garis bawah di depan (Bagian 2.2) memecahkan sisi-tulis enkapsulasi — tidak ada apa pun di luar class yang merusak atribut itu, secara konvensi. Tapi ini juga, sebagai efek samping yang tak terhindarkan, menghalangi dunia luar untuk sekadar membaca nilai atribut saat ini, yang sangat sering menjadi sesuatu yang secara sah dibutuhkan kode luar (halaman katalog perlu menampilkan judul buku; fitur pencarian perlu memeriksa penulisnya). @property adalah method kecil, di-decorate agar bisa DIBACA persis seperti atribut biasa (book.title, bukan book.title()), yang seluruh tugasnya adalah mengembalikan nilai satu atribut saat ini.

Getter property sengaja dibuat sesederhana mungkin: selain self, ia tidak menerima parameter, dan badannya adalah satu pernyataan return. Karena @property membiarkan kode pemanggil menulis book.title alih-alih book.get_title(), kode Python yang memakai property terlihat persis seperti kode yang memakai atribut public biasa — pemanggil tidak bisa membedakan, hanya dari titik pemanggilannya, apakah title adalah atribut mentah atau property terhitung, yang justru itulah intinya: Anda bisa mulai dengan atribut public biasa dan mengubahnya menjadi @property kemudian, tanpa mengubah satu baris pun kode pemanggil.

@property Python	Getter Java	Mengembalikan
title	getTitle()	judul buku (str / String)
author	getAuthor()	penulis buku (str / String)
isbn	getIsbn()	ISBN buku (str / String)
publication_year	getPublicationYear()	tahunnya, atau 0 jika tak tercatat (int)
is_on_loan	isOnLoan()	apakah buku sedang dipinjam (bool / boolean)

@property Python	Getter Java	Mengembalikan
times_renewed	getTimesRenewed()	berapa kali peminjaman ini diperpanjang (int)

Property yang mengembalikan str atau int tetap aman, bahkan tanpa "menyalinnya"

Kekhawatiran wajar: jika title sekadar mengembalikan self._title langsung, bukankah pemanggil yang menerimanya bisa balik merusak state internal Book lewat referensi yang dikembalikan itu? Untuk str, int, dan bool secara spesifik, tidak — tipe-tipe ini immutable di Python, artinya tidak ada operasi pada str atau int yang mengubah karakter atau digit yang sudah dipegangnya. Pemanggil menerima nilai yang bisa mereka baca, salin, atau buang, tapi tidak pernah sebuah pegangan yang membiarkan mereka menjangkau balik ke dalam Book dan mengubah atribut internalnya. (Pertanyaan ini menjadi jauh lebih menarik begitu sebuah property mengembalikan tipe mutable, seperti list peminjaman — masalah yang dibahas langsung mata kuliah ini di Bab 10, Generics & Collections.)

2.6 Kontrol Akses: Empat Level Java vs. Konvensi Python

Bagian 2.2 dan Bab 1 sama-sama sudah memakai konvensi garis-bawah-di-depan berulang kali tanpa mengkatalogkan secara lengkap apa yang ditawarkan Java sebagai gantinya. Java mendefinisikan empat level visibilitas yang ditegakkan compiler yang bisa diberikan pada field, constructor, atau method; Python menawarkan tepat satu konvensi penamaan dan mempercayai programmer untuk selebihnya (Gambar 2.3).

Access Modifier Sekilas: Empat Level Java vs. Konvensi Python

Compiler Java menegakkan visibilitas; Python menyinyalkan maksud yang sama lewat konvensi penamaan dan mempercayai programmer.

Modifier Java	Terlihat dari	Padanan Python
`private`	Hanya class itu sendiri	`_satu_garis_bawah` (konvensi: internal)
`(package-private)`	Hanya package yang sama	— (Python tidak punya kontrol akses tingkat package)
`protected`	Package + subclass	— (jarang dibutuhkan sampai Bab 9, pewarisan)
`public`	Di mana pun	tanpa garis bawah (default)

private milik Java ditegakkan compiler — kode di luar class bahkan tidak bisa mencoba meng-compile pelanggarannya. _garis_bawah_di_depan Python adalah janji antar-programmer, bukan tembok yang dibangun interpreter untuk Anda.

Gambar 2.3 — Empat level akses Java, dari yang paling sempit (private) hingga paling luas (public), berdampingan dengan padanan konvensi-penamaan Python untuk masing-masing.

private milik Java adalah level paling sempit: hanya terlihat di dalam badan class itu sendiri, dan ditegakkan compiler. package-private (default Java, saat tidak ada kata kunci sama sekali yang ditulis) membuka visibilitas ke setiap class lain dalam package yang sama. protected memperluas itu ke subclass juga, di mana pun mereka berada — level yang kegunaan penuhnya baru terlihat begitu Bab 9 memperkenalkan pewarisan. public, level terluas, adalah yang dipakai Java untuk setiap getter dan setiap method perilaku. Python sama sekali tidak punya padanan package-private atau protected di tingkat bahasa — satu garis bawah di depan mencakup semua yang dianggap Python "internal," dan

garis bawah ganda di depan (name-mangling, mis. self.__secret) ada terutama untuk menghindari tabrakan nama tak sengaja di subclass, bukan untuk menegakkan privasi sejati; ia tetap bisa dijangkau dari luar class jika Anda tahu nama yang sudah di-mangle.

> **Tanpa garis bawah di mana pun mengalahkan seluruh inti Bab 1**
>
> Secara teknis legal menamai setiap atribut tanpa garis bawah di depan dan menghapus setiap property — tidak ada apa pun di Python yang mencegah ini, dan tidak ada error yang pernah dipicu. Melakukannya membuang semua yang diperjuangkan Bab 1: dengan setiap atribut bernama public biasa, kode di mana pun dalam program bisa menulis book.on_loan = True secara langsung, sepenuhnya melewati pemeriksaan borrow() tentang apakah buku sudah dipinjam, memunculkan kembali persis risiko korupsi-data yang dibangun perbandingan prosedural-vs-OOP Bab 1 (Gambar 1.3) untuk didemonstrasikan. Enkapsulasi bukan karakter garis bawah itu sendiri; ia adalah atribut berprefiks-garis-bawah yang sengaja dipasangkan dengan sekumpulan kecil method dan property public yang terkurasi.

2.7 Contoh Kerja: Memperluas Book dengan Tahun Terbit

Contoh kerja bab ini mengambil class Book dari Bab 1 dan memperluasnya: atribut _publication_year baru (lewat argumen constructor default), dan @property untuk setiap atribut.

Fase 1 — Understand (Pahami)

```
Kebutuhan baru: sebagian program (fitur pencarian katalog di masa
depan) perlu akses baca ke judul, penulis, isbn, status pinjam, dan
jumlah perpanjangan sebuah buku -- dan perlu mencatat tahun terbit
buku bila diketahui, tanpa mewajibkannya bila tidak diketahui.
```

Fase 2 — Abstract (Abstraksi)

```
Bentuk umum: satu DATA baru (publication_year), plus kanal BACA
terkendali (@property) untuk setiap data yang sudah ada, plus cara
membangun objek entah data baru itu tersedia atau tidak saat
konstruksi.
```

Fase 3 — Design (Rancang)

```
Atribut baru:
  _publication_year : int  (0 berarti 'tidak tercatat')
Constructor (SATU __init__, argumen default mencakup kedua kasus):
  __init__(self, title, author, isbn, publication_year=0)
Property (hanya-baca; SATU-SATUNYA kanal baca untuk atribut
bergaris-bawah):
  title, author, isbn, publication_year, is_on_loan,
  times_renewed   -- semua hanya menerima self, satu return
```

Fase 4 — Analyze (Analisis)

```
Argumen kebenaran: _publication_year ditetapkan tepat sekali, di
dalam __init__, persis seperti _title, _author, dan _isbn -- tidak
ada method lain di class Book mata kuliah ini yang menetapkan
ulang. Setiap property mengembalikan tipe immutable (str, int, atau
bool), jadi tidak ada pemanggil yang membaca nilai property yang
bisa menjangkau balik dan mengubah state internal Book lewatnya.
Hanya ada SATU constructor, jadi tidak ada risiko dua constructor
tidak-sinkron seperti yang bisa terjadi pada dua constructor
overload Java.
```

Fase 5 — Implement (Implementasikan)

```python
class Book:
    def __init__(self, title: str, author: str, isbn: str,
                 publication_year: int = 0) -> None:
        self._title = title
        self._author = author
        self._isbn = isbn
        self._publication_year = publication_year
        self._on_loan = False
        self._times_renewed = 0

    # borrow(), return_book(), renew() -- tak berubah dari Bab 1

    @property
    def title(self) -> str:
        return self._title

    @property
    def author(self) -> str:
        return self._author

    @property
    def isbn(self) -> str:
        return self._isbn

    @property
    def publication_year(self) -> int:
        return self._publication_year

    @property
    def is_on_loan(self) -> bool:
        return self._on_loan

    @property
    def times_renewed(self) -> int:
        return self._times_renewed
```

Script driver di bawah membuat dua objek Book, satu memberikan tahun terbit yang diketahui dan satu mengandalkan nilai default (tak tercatat, default ke 0), lalu menjalankan beberapa property. Ia ditampilkan di sini secara lengkap, persis seperti tampil di Code/Python/Chapter02/library_demo.py, dan keluaran lengkapnya persis yang diverifikasi Lampiran 2.A:

```
from pustaka import Book

def main() -> None:
    clean_code = Book("Clean Code", "Robert C. Martin",
                      "978-0132350884", 2008)
    pragmatic = Book("The Pragmatic Programmer", "Hunt & Thomas",
                     "978-0135957059")

    print(clean_code)
    print(pragmatic)

    print(f"Clean Code author: {clean_code.author}")
    print(f"Pragmatic Programmer year: {pragmatic.publication_year} "
          f"(0 = unrecorded)")

    clean_code.borrow()
    print(clean_code)

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

Telusuri sendiri sebelum melanjutkan membaca

pragmatic dibangun tanpa argumen keempat, jadi publication_year jatuh ke nilai defaultnya. Apa yang dikembalikan pragmatic.publication_year, dan apa yang dicetak baris __str__() untuk tahun pragmatic, mengingat itu? Telusuri: karena tidak ada argumen keempat yang diberikan, publication_year default persis ke 0, dan __str__() (Fase 5 Bagian 2.7, diperluas lebih lanjut di file source lengkap) ditulis untuk menghilangkan sepenuhnya kurung tahun setiap kali _publication_year bernilai 0, alih-alih mencetak angka 0 yang menyesatkan.

2.8 Menjalankan Program Anda

Tidak ada yang berubah dari proses menjalankan yang diperkenalkan Bab 1 Bagian 1.8: python3 -m py_compile memeriksa kedua file untuk kesalahan sintaks, dan python3 library_demo.py sekaligus memuat pustaka.py (lewat pernyataan import) dan menjalankan kode library_demo.py sendiri.

```
$ python3 -m py_compile pustaka.py library_demo.py
$ python3 library_demo.py
```

Jika Anda memperluas pustaka.py sendiri sambil mengerjakan Soal Latihan bab ini, menjalankan python3 -m py_compile lebih dulu sepadan dengan detik ekstra yang dibutuhkannya — ia menangkap salah ketik atau kesalahan indentasi sebelum Anda harus mengurainya dari traceback yang dihasilkan menjalankan script yang (masih rusak) itu.

2.9 OOP dalam Praktik: Profitina

> **Tentang kotak ini**
>
> Seperti di setiap bab, kotak ini menghubungkan materi mata kuliah dengan Profitina, platform AI-visibility (Generative Engine Optimization) Indonesia yang nyata dan aktif, dibangun oleh penulis — menjelaskan DI MANA dan MENGAPA sebuah konsep dipakai dalam sistem nyata yang berjalan, bukan logika bisnis atau source code lengkap Profitina yang proprietary, yang tetap rahasia. Lihat dokumen pendamping "Profitina: A Non-Confidential Technical Overview" untuk gambaran lengkapnya.

Backend Profitina mendefinisikan class yang method __init__-nya mencerminkan persis pola yang diajarkan bab ini: sebuah class hasil-pindai, misalnya, dibangun sekali per pemindaian AI-visibility yang selesai, dengan setiap atribut yang dibutuhkannya (skor bisnis, timestamp pemindaian, model AI mana yang dikonsultasikan) diberikan saat konstruksi lalu diperlakukan sebagai tetap secara efektif seumur hidup objek itu — disiplin yang sama yang didemonstrasikan atribut _publication_year bab ini dalam skala kecil. Ketika kode berikutnya perlu membaca skor hasil-pindai untuk ditampilkan di dashboard, ia melakukannya lewat @property, tidak pernah dengan menjangkau langsung ke atribut berprefiks-garis-bawah objek, persis alasan yang dijelaskan Bagian 2.5: kanal baca terkendali yang tidak pernah bisa menjadi kanal tulis tak-terkendali.

Ini juga tempat kode produk Profitina sendiri menunjukkan persis konvensi yang diajarkan bab ini pada skala produksi: setiap class Profitina sendiri mengekspos datanya secara eksklusif lewat getter @property, bukan akses atribut telanjang, dan setiap atribut internal memakai konvensi garis-bawah-di-depan secara konsisten — konvensi yang sama yang diikuti class Book bab ini, hanya dengan taruhan yang jauh lebih besar daripada katalog perpustakaan.

2.10 Ringkasan Bab dan Poin Penting

- Sebuah class punya (paling banyak) tiga jenis anggota: atribut (data objek), constructor (bagaimana objek dibangun), dan method (perilaku objek).
- Tugas __init__ adalah membawa satu objek ke keberadaan dalam keadaan valid; self. selalu wajib dipakai untuk menjangkau atribut, sehingga Python tidak punya padanan tabrakan penamaan this.-vs-nama-telanjang milik Java.
- Python tidak punya constructor overloading — hanya ada tepat satu __init__ per class — tapi nilai parameter default mencakup kasus penggunaan yang sama yang di Java membutuhkan beberapa constructor, dengan logika penetapan tetap hidup di tepat satu tempat baik dengan cara apa pun.
- @property adalah method kecil yang mengembalikan nilai satu atribut sambil terlihat, bagi kode pemanggil, persis seperti pembacaan atribut biasa — memulihkan akses baca tanpa membuka kembali akses tulis.
- Java punya empat level akses yang ditegakkan compiler — private, package-private, protected, public; Python punya tepat satu konvensi (garis bawah di depan) dan mempercayai programmer untuk selebihnya.
- Menamai setiap atribut tanpa garis bawah di depan mengalahkan tujuan enkapsulasi yang ditetapkan Bab 1 — atribut berprefiks-garis-bawah yang sengaja dipasangkan dengan

sekumpulan kecil method dan property public itulah yang membuat kebenaran sebuah objek bisa diperdebatkan sama sekali.

Sebuah class adalah janji tentang apa yang boleh terjadi pada datanya, dan constructor adalah tempat janji itu mulai ditepati. Semua yang ditambahkan bab ini pada Book — atribut baru, constructor yang lebih kaya, enam property — ada untuk menepati persis janji yang sama yang dibuat Bab 1, hanya untuk objek yang sedikit lebih kaya.

2.11 Soal Latihan

Soal-soal ini untuk belajar mandiri dan diskusi kelas; tidak dinilai. Kerjakan sebelum Lecture 3.

1. __init__ tunggal Python dengan argumen default dan dua constructor overload Java sama-sama menjamin logika penetapan hidup di tepat satu tempat, tapi mereka mencapainya lewat mekanisme berbeda. Jelaskan, dengan kata-kata Anda sendiri, apa sebenarnya mekanisme masing-masing bahasa.
2. Apa yang akan terjadi — sebutkan secara spesifik error apa yang muncul dan kapan — jika Anda menulis publication_year = publication_year (alih-alih self._publication_year = publication_year) di dalam __init__ milik Book?
3. is_on_loan adalah nama property tanpa prefiks get_, tidak seperti get_is_on_loan yang hipotetis. Tulis satu kalimat menjelaskan mengapa property Python secara konvensional menghilangkan prefiks kata kerja sepenuhnya, tidak seperti konvensi getter get-/is- milik Java.
4. Misalkan sebuah property publishers ditambahkan yang mengembalikan list penerbit-penerbit sebelumnya (mutable) alih-alih satu str. Jelaskan mengapa argumen keamanan kotak INSIGHT Bagian 2.5 tidak lagi sepenuhnya berlaku, dan apa yang bisa salah.
5. Tulis property remaining_renewals untuk Book yang mengembalikan berapa banyak perpanjangan yang masih tersedia (maksimum 2, dikurangi times_renewed). Haruskah ini @property atau method biasa yang menerima argumen, dan apakah menambahkannya membutuhkan penyentuhan konvensi penamaan atribut mana pun yang sudah ada? Justifikasi jawaban Anda.

Lampiran 2.A — Log Verifikasi Eksekusi

File lengkap pustaka.py dan library_demo.py (Code/Python/Chapter02/, disertakan bersama buku ini) diperiksa sintaksnya dan dijalankan pada CPython 3.11.15 sebelum bab ini ditulis, untuk memastikan kodenya benar, bukan sekadar "terlihat benar". Keluaran direproduksi verbatim di bawah.

```
$ python3 -m py_compile pustaka.py library_demo.py
(tidak ada error)

$ python3 library_demo.py
"Clean Code" by Robert C. Martin [ISBN 978-0132350884, 2008] - on the shelf
"The Pragmatic Programmer" by Hunt & Thomas [ISBN 978-0135957059] - on the shelf
Clean Code author: Robert C. Martin
Pragmatic Programmer year: 0 (0 = unrecorded)
"Clean Code" by Robert C. Martin [ISBN 978-0132350884, 2008] - ON LOAN (renewed
0x)
```

Referensi

[1] Python Software Foundation. "Classes." https://docs.python.org/3/tutorial/classes.html (diakses Agustus 2026).

[2] Python Software Foundation. "Status of Python Versions." https://devguide.python.org/versions/ (diakses Agustus 2026) — mengonfirmasi Python 3.14 sebagai rilis stabil terbaru (Oktober 2025), 3.13 masih dalam dukungan aktif.

[3] R. C. Martin. Clean Code: A Handbook of Agile Software Craftsmanship. Prentice Hall, 2008 (sumber judul contoh yang dipakai bab ini).

[4] Microsoft. "Visual Studio Code — Python." https://code.visualstudio.com/docs/languages/python (diakses Agustus 2026).

[5] JetBrains. "PyCharm." https://www.jetbrains.com/pycharm/ (Community Edition, diakses Agustus 2026).

BAB 3

Tipe Bawaan, Alur Kontrol, String & Alat Debugging

Bab 1 dan 2 membangun dan memperluas satu class. Bab ini mundur selangkah ke fondasi bahasa yang sebenarnya dijalankan oleh setiap badan method: tipe bawaan Python yang dinamis, dua cara bercabang (if/elif/else dan pernyataan match), bentuk loop milik Python sendiri, mengapa string dibandingkan dengan cara tertentu, dan bagaimana mengetahui apa yang sebenarnya dilakukan program Anda ketika hasilnya tidak sesuai harapan.

Tujuan Pembelajaran — setelah bab ini Anda mampu:

(1) Menyebutkan tipe bawaan inti Python (int, float, bool, str, complex) serta mana yang mutable. (2) Menulis rantai if/elif/else dan pernyataan match Python 3.10+ untuk menyandikan logika percabangan, serta menjelaskan mengapa match adalah pernyataan, bukan ekspresi. (3) Memilih dengan tepat antara loop for dan while, serta menjelaskan mengapa Python tidak punya for bergaya-C atau do-while. (4) Menjelaskan mengapa == Python sudah membandingkan konten string, dan mengapa mengandalkan is untuk string tidak pernah aman meski kebetulan berhasil. (5) Menggunakan print debugging, breakpoint pdb, dan pernyataan assert untuk mendiagnosis program yang salah. (6) Memperluas class Book milik Pustaka dengan method perhitungan denda dan method status-perpanjangan, lalu memeriksa sintaks, menjalankan, dan memverifikasinya sendiri.

3.1 Melanjutkan dari Bab 2

Bab 2 membuka class Book dan mengamati tiga jenis anggotanya: attribute, __init__, dan property. Setiap method yang Anda tulis di Bab 1 dan 2 -- borrow(), renew(), setiap @property -- punya badan, dan setiap badan itu dibangun dari kumpulan kecil bahan yang persis sama: nilai-nilai dari segelintir tipe bawaan, keputusan (jika ini, maka itu), pengulangan (lakukan ini beberapa kali), dan teks (str). Bab ini mengamati langsung bahan-bahan tersebut, memakai Book sendiri sebagai contoh berjalan di mana pun membantu, dan ditutup dengan keterampilan praktis yang telah diasumsikan dimiliki oleh setiap bab sebelumnya: bagaimana mengetahui mengapa program melakukan sesuatu selain yang Anda harapkan.

Gambar 3.1 — Peta jalan 16-lecture OOP. Bab ini adalah Lecture 3, pemberhentian terakhir sebelum Kuis 1 (Lecture 4) menguji Lecture 1 hingga 3 sekaligus.

3.2 Tipe Bawaan Python

Setiap nilai di Python adalah objek -- bahkan int atau bool -- tapi sekumpulan kecil tipe bawaan mencakup mayoritas besar kode sehari-hari: int, float, bool, str, dan complex. Berbeda dari delapan primitif Java, tidak satu pun dari ini punya ukuran tetap yang diekspos Python ke Anda: int bisa membesar tanpa batas tanpa overflow (Python otomatis mengalokasikan lebih banyak memori), dan tidak ada byte, short, atau long terpisah untuk dipilih. Book sudah memakai dua dari tipe ini: int (publication_year, times_renewed) dan bool (on_loan, dan hasil setiap perbandingan ==) (Gambar 3.1 dan 3.2).

Tipe Bawaan Python: Dinamis, Bukan Primitif

Python tidak memiliki kategori "primitif" terpisah -- setiap nilai, termasuk angka dan boolean, adalah objek.

Tipe	Contoh	Bisa Diubah?	Analog Terdekat di Java
`int`	`42, -7, 10**20`	Tidak	int / long, tapi tak terbatas
`float`	`3.14, 2.0`	Tidak	double (tidak ada tipe float terpisah)
`bool`	`True, False`	Tidak	boolean (tapi bool ADALAH subclass int)
`str`	`"Clean Code"`	Tidak	String
`complex`	`3+4j`	Tidak	(tidak ada analog bawaan)

int tidak memiliki ukuran tetap atau titik overflow -- ia tumbuh otomatis untuk menampung bilangan bulat berapa pun. bool sebenarnya adalah subclass dari int (True == 1 bernilai True) -- perbedaan bahasa yang genuine, bukan penyederhanaan.

Gambar 3.2 — Tipe bawaan inti Python, dengan contoh, mutabilitas, dan analog Java terdekat masing-masing. bool secara teknis adalah SUBCLASS dari int, detail dengan konsekuensi nyata (kotak catatan Bagian 3.2).

Dua aturan praktis mencakup mayoritas besar kode sungguhan. Pertama, pakai int sebagai default untuk bilangan bulat dan float untuk bilangan berpecahan; Python tidak punya double terpisah untuk dipilih -- float selalu berupa nilai IEEE 754 64-bit, representasi dasar yang sama dengan double Java. Kedua, str menyimpan teks sepanjang apa pun dan bukan pengganti satu karakter seperti char Java -- Python bahkan tidak punya tipe karakter terpisah sama sekali; str satu-karakter tetap saja str.

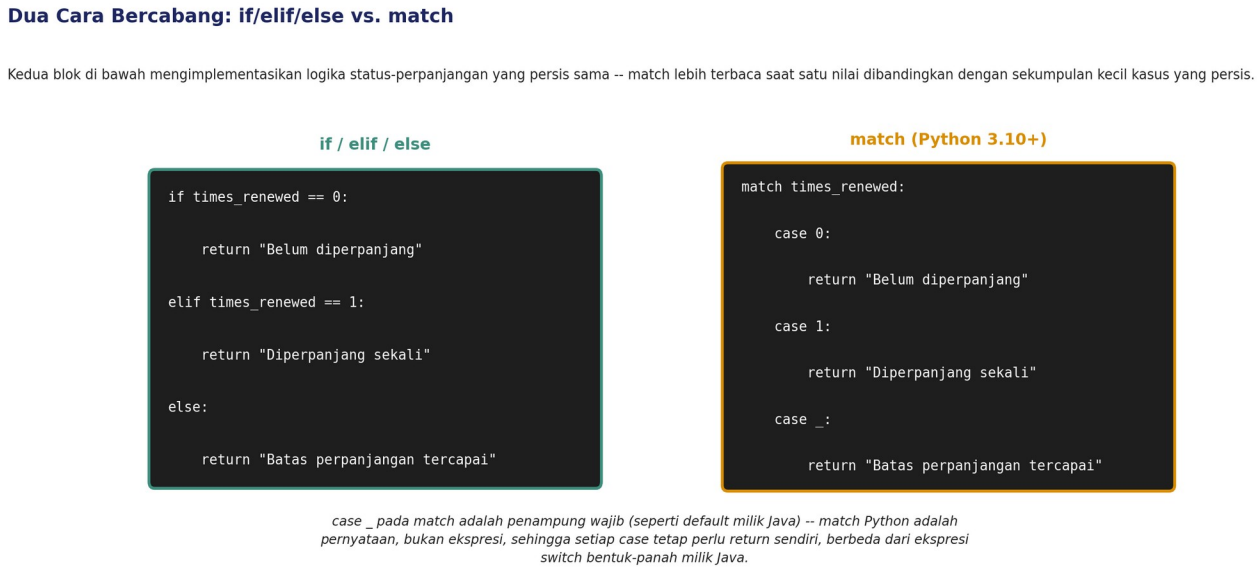
> **bool sebagai subclass int bukan sekadar trivia**
>
> Karena bool adalah subclass dari int, True == 1 dan False == 0 keduanya bernilai True di Python, dan True + True sungguh-sungguh bernilai 2. Ini kadang mengejutkan kode yang mengharapkan perbandingan True/False ketat gagal terhadap 1 atau 0 numerik -- padahal tidak akan gagal. Dan persis seperti di Java, membandingkan nilai float dengan == adalah bug diam-diam klasik: 0.1 + 0.2 == 0.3 bernilai False di Python karena alasan IEEE 754 yang identik dengan masalah yang sama di Java. Jangan pernah membandingkan nilai float untuk kesamaan tepat; bandingkan abs(a - b) terhadap toleransi kecil, atau pakai math.isclose(a, b).

3.3 Alur Kontrol: if/elif/else dan match

Kondisi pernyataan if dievaluasi berdasarkan truthiness -- Python lebih permisif di sini dibanding Java: 0, 0.0, "", [], dan None semuanya falsy, dan nilai lainnya truthy, sehingga int atau str BISA muncul

langsung dalam kondisi tanpa perbandingan eksplisit, berbeda dari Java. Merangkai elif membiarkan sebuah keputusan mengalir lewat beberapa kemungkinan yang saling eksklusif, diakhiri else opsional yang menangkap semua yang belum tertangkap cabang sebelumnya.

Python 3.10 memperkenalkan match, cara kedua untuk membuat jenis keputusan yang sama ketika setiap cabang membandingkan satu nilai terhadap sekumpulan kecil case tertentu. Lengan case sebuah pernyataan match diperiksa dari atas ke bawah, dan case _: adalah wildcard penangkap-semua, memainkan peran yang sama dengan default di dalam switch Java. Perbedaan krusial dari switch Java: match Python selalu berupa PERNYATAAN, tidak pernah ekspresi -- match sendiri tidak bisa dikembalikan atau ditetapkan; badan setiap case harus punya return atau penetapan sendiri (Gambar 3.3).



Gambar 3.3 — Keputusan status-perpanjangan yang sama, ditulis sebagai if/elif/else dan sebagai pernyataan match Python 3.10+. Keduanya benar; match mengomunikasikan "satu nilai, beberapa case tertentu" secara lebih langsung, tapi berbeda dari ekspresi switch Java, ia adalah PERNYATAAN dan tidak mengembalikan apa pun dengan sendirinya.

Kapan memilih match dibanding if/elif/else

Pakai match ketika setiap cabang menguji nilai tunggal yang sama terhadap case tertentu (int, str, atau pola struktural) -- persis situasi renewal_status_label() di Bagian 3.7. Pilih if/elif/else ketika cabang menguji nilai berbeda, atau menguji rentang/ketidaksetaraan alih-alih case kecocokan-tepat -- persis situasi calculate_fine(), yang membandingkan days_overdue terhadap ambang batas (<= 7, <= 30), bukan nilai tepat yang bisa langsung diekspresikan bentuk case sederhana match tanpa klausa guard tambahan.

3.4 Loop: for dan while

Python menawarkan tepat dua kata kunci loop, dan Bab 1-2 sudah memakai salah satunya (for x in some_list:) tanpa menyebutnya secara formal. Python sama sekali tidak punya for bergaya-C (inisialisasi; kondisi; update), dan tidak punya do-while -- for selalu beriterasi atas sebuah ITERABLE (list, string, range, atau apa pun yang diketahui Python cara melangkahinya satu item per waktu), tidak pernah counter polos dengan kondisi yang ditulis manual.

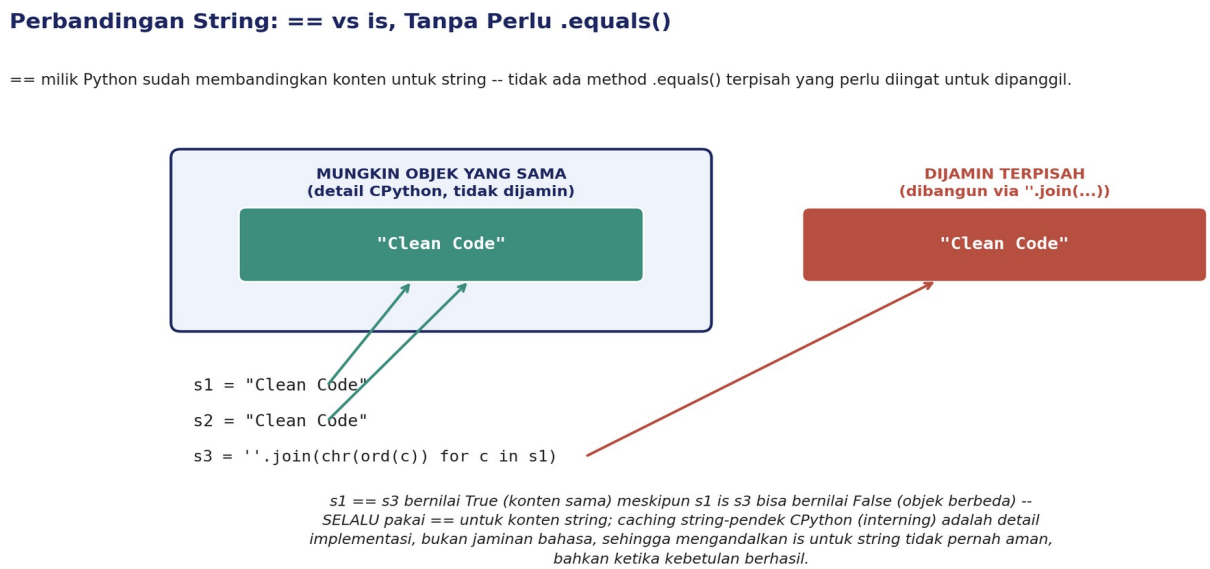
range(5) menghasilkan bilangan bulat 0 hingga 4, sehingga for i in range(5): menjalankan badannya tepat lima kali -- padanan Python terdekat dari for loop berhitung milik Java, tapi diekspresikan sebagai beriterasi atas urutan bilangan bulat, bukan memanipulasi indeks secara langsung. while kondisi: memeriksa kondisinya sebelum setiap iterasi, termasuk yang pertama -- ia bisa berjalan nol kali bila kondisi sejak awal false, persis seperti while Java. Karena Python tidak punya do-while, loop yang harus berjalan minimal sekali ditulis sebagai while True: dengan break eksplisit di dalam badan begitu kondisi keluar terpenuhi.

Beriterasi list sambil mengubahnya adalah bug diam-diam klasik

for item in my_list: my_list.remove(item) diam-diam melewati elemen, karena for loop Python melacak posisi numerik ke dalam list, dan menghapus item menggeser setiap elemen berikutnya satu posisi lebih awal tanpa posisi loop bergerak mundur untuk mengompensasi. Beriterasi atas SALINAN (for item in my_list[:]: atau for item in list(my_list):) kapan pun badan loop mungkin menambah atau menghapus dari list yang sedang diiterasi itu sendiri.

3.5 String: Perbandingan Nilai dan Identitas

str Python, begitu dibuat, tidak pernah bisa diubah -- setiap method str yang tampak mengubah string (upper(), strip(), replace(...)) sebenarnya mengembalikan str baru sama sekali, membiarkan yang asli tidak tersentuh. Ini adalah imutabilitas str, properti yang sama dengan String Java, tapi Python menyelesaikan pertanyaan perbandingan jauh lebih sederhana dibanding Java: == selalu membandingkan KONTEN untuk str (dan untuk setiap tipe bawaan lain), tidak pernah identitas. Tidak ada padanan .equals() Java untuk diingat, dan tidak ada jebakan serupa membandingkan konten dengan operator yang salah -- == sudah melakukan hal yang benar untuk string, list, dan setiap tipe umum lainnya.



Gambar 3.4 — s1 dan s2, keduanya literal str dengan teks sama, MUNGKIN kebetulan menjadi objek yang sama berkat small-string interning milik CPython -- tapi ini detail implementasi, bukan pernah jaminan bahasa. s3, dibangun via ''.join(...), DIJAMIN menjadi objek terpisah, namun s1 == s3 tetap True, karena == selalu membandingkan konten.

is, operator identitas Python, menjawab pertanyaan yang sama sekali berbeda: "apakah kedua nama ini terikat ke objek yang persis sama di memori?" CPython (implementasi referensi) kebetulan memakai ulang ("interning") literal string kecil dan bilangan bulat kecil demi efisiensi, sehingga "exit" is

"exit" bisa mencetak True dalam praktiknya -- tapi ini detail implementasi CPython yang tidak dijamin spesifikasi bahasa Python, bisa berbeda antar implementasi Python, dan bisa berubah antar versi. Mengandalkan is untuk string tidak pernah aman, bahkan pada kesempatan langka ketika ia tampak berfungsi.

is untuk string: berfungsi karena kebetulan, rusak karena desain

isbn_from_scanner = "".join(chr(ord(c)) for c in isbn_from_catalog) membangun string dengan konten identik dengan isbn_from_catalog lewat objek yang sungguh-sungguh terpisah -- tidak ada interning yang berlaku untuk string yang dirakit dengan cara ini. isbn_from_catalog == isbn_from_scanner bernilai True (konten cocok); isbn_from_catalog is isbn_from_scanner bernilai False (objek berbeda). Selalu pakai == untuk membandingkan konten string di Python; sisakan is khusus untuk membandingkan terhadap singleton None (if x is None:), tidak pernah untuk membandingkan dua string, dua angka, atau dua nilai biasa lainnya.

3.6 Alat Debugging: Mengetahui Apa yang Sebenarnya Dilakukan Program Anda

Setiap program yang pernah Anda tulis sejauh ini, pada suatu titik, pernah melakukan sesuatu selain yang Anda harapkan. Tiga alat mencakup hampir semua situasi yang ditemui mata kuliah pengantar, dan mengetahui alat mana yang cocok untuk momen tertentu adalah keterampilan tersendiri yang layak dibangun secara sengaja.

Alat	Cocok untuk	Catatan
Debugging print()	Pemeriksaan cepat dan langsung satu-dua nilai, di mana pun dalam kode	Harus dihapus (atau diubah menjadi logging sungguhan) sebelum kode dianggap selesai
pdb / breakpoint()	Mengikuti jalur eksekusi program yang tepat dan memeriksa banyak variabel sekaligus, tanpa mengedit kode sumber	Butuh mempelajari perintah pdb sendiri (n, s, c, p, l); berlebihan untuk pemeriksaan satu-nilai
Pernyataan assert	Mendokumentasikan dan memeriksa invarian internal yang TIDAK BOLEH pernah false bila kode benar	Dihapus sepenuhnya saat Python dijalankan dengan flag -O -- tidak boleh pernah diandalkan berjalan di setiap lingkungan

Biseksi: cara tercepat melokalisasi bug yang tidak diketahui

Ketika output program salah di suatu tempat dalam urutan langkah yang panjang tapi Anda belum tahu di mana, cetak (atau pasang breakpoint() pada) nilai kira-kira di tengah urutan, bukan di awal sekali. Bila sudah salah di situ, bug ada di paruh pertama; bila masih benar, bug ada di paruh kedua. Mengulangi biseksi ini mempersempit pencarian seratus baris menjadi segelintir baris dalam sekitar tujuh pemeriksaan, ide pembagian-dua yang sama yang diformalkan Struktur Data (mata kuliah berikutnya) sebagai binary search.

Alat	Cocok untuk	Catatan
assert untuk bug ANDA sendiri, tidak pernah untuk memvalidasi input pengguna assert first_name is not None tepat untuk mendokumentasikan "ini tidak boleh pernah None bila kode saya sendiri benar" -- dan karena flag -O Python menghapus setiap pernyataan assert sepenuhnya dari bytecode terkompilasi, kode yang bergantung pada assert benar-benar berjalan untuk menolak DATA buruk (berbeda dari menangkap bug di logika ANDA sendiri) akan diam-diam meloloskan data buruk itu begitu saja kapan pun -O dipakai. Input pengguna dan data tak terpercaya lainnya harus berada di balik pemeriksaan if eksplisit (atau, mulai Bab 6, exception yang dilempar), tidak pernah hanya di balik assert.		

3.7 Contoh Kerja: Perhitungan Denda dan Status Perpanjangan untuk Book

Contoh kerja bab ini menambahkan dua method kecil ke Book: calculate_fine(self, days_overdue: int) -> float, mendemonstrasikan aritmetika bawaan int/float dan struktur tingkat if/elif/else, dan renewal_status_label(self) -> str, menulis ulang keputusan tiga-arah yang persis sama yang sudah dikenal dari times_renewed sebagai pernyataan match.

Fase 1 — Understand

```
Kebutuhan baru: diberikan berapa hari sebuah buku terlambat,
hitung jumlah denda yang meningkat semakin lama buku itu
tertahan, lalu dibatasi maksimum -- plus label yang bisa dibaca
manusia untuk berapa kali buku diperpanjang, ditulis ulang
memakai pernyataan match bab ini.
```

Fase 2 — Abstract

```
Satu PERILAKU baru yang menerima int dan mengembalikan float
(jadwal denda yang monoton meningkat, lalu dibatasi), plus
PERILAKU kedua yang menyatakan ulang state yang sudah ada
(times_renewed) lewat konstruksi alur-kontrol berbeda namun
setara.
```

Fase 3 — Design

```
FASE 3 -- DESIGN

Method baru (tanpa attribute baru -- keduanya membaca state
yang sudah ada atau parameter):
  calculate_fine(self, days_overdue: int) -> float
    days_overdue <= 0            -> 0.00
    days_overdue dalam 1..7      -> days_overdue * 0.50
    days_overdue dalam 8..30     -> 3.50+(days_overdue-7)*1.00
    days_overdue > 30            -> 50.00 (dibatasi)
  renewal_status_label(self) -> str
    match self._times_renewed: case 0/1/_ -> ...
```

Fase 4 — Analyze

```
Argumen kebenaran: tiga perbandingan calculate_fine (<= 0,
<= 7, <= 30) saling eksklusif dan kolektif menyeluruh -- setiap
int jatuh ke tepat satu cabang, dan jumlah denda tidak pernah
menurun seiring days_overdue membesar, sesuai kebutuhan.
Pernyataan match renewal_status_label menyeluruh berdasarkan
konstruksi (case _ mencakup setiap nilai yang tidak eksplisit
terdaftar), sehingga tidak bisa pernah gagal mengembalikan
nilai seperti rantai if yang tidak lengkap bisa diam-diam gagal.
```

Fase 5 — Implement

```
# attribute, __init__, dan property: tak berubah dari Bab 2

def calculate_fine(self, days_overdue: int) -> float:
    if days_overdue <= 0:
        return 0.0
    elif days_overdue <= 7:
        return days_overdue * 0.50
    elif days_overdue <= 30:
        return 3.50 + (days_overdue - 7) * 1.00
    else:
        return 50.00

def renewal_status_label(self) -> str:
    match self._times_renewed:
        case 0:
            return "No renewals yet"
        case 1:
            return "Renewed once"
        case _:
            return "Max renewals reached"
```

Driver di bawah mendemonstrasikan kedua method baru, beriterasi lewat sebuah list sampel jumlah hari terlambat dengan for loop (Bagian 3.4), dan ditutup dengan perbandingan == vs is yang diperingatkan Bagian 3.5, sengaja membangun str non-interned dengan "".join(...) agar perbedaannya terlihat. Ditampilkan di sini secara utuh, persis seperti tampil di Code/Python/Chapter03/library_demo.py, dan output lengkapnya persis apa yang diverifikasi Lampiran 3.A:

```
clean_code = Book("Clean Code", "Robert C. Martin",
                  "978-0132350884", 2008)
print(clean_code)

clean_code.borrow()
clean_code.renew()
print(f"Renewal status: {clean_code.renewal_status_label()}")

overdue_days_samples = [0, 3, 15, 45]
for days in overdue_days_samples:
    fine = clean_code.calculate_fine(days)
    print(f"Overdue {days:2d} day(s) -> fine: {fine:.2f}")

isbn_from_catalog = "978-0132350884"
isbn_from_scanner = "".join(
    chr(ord(c)) for c in isbn_from_catalog)
print(f"== comparison: {isbn_from_catalog == isbn_from_scanner}")
print(f"is comparison: {isbn_from_catalog is isbn_from_scanner}")
```

Telusuri sendiri sebelum melanjutkan

Setelah borrow() dan satu panggilan renew(), times_renewed bernilai 1 -- apa yang dikembalikan renewal_status_label()? Telusuri tingkatan denda: 0 hari persis 0.00 (batas milik cabang pertama, days_overdue <= 0); 3 hari jatuh di tingkat 1-7 (3 * 0.50); 15 hari jatuh di tingkat 8-30 (3.50 + 8 * 1.00); 45 hari melebihi 30 dan dibatasi 50.00. Dan isbn_from_scanner, dibangun dengan "".join(...), dijamin menjadi objek berbeda dari isbn_from_catalog -- jadi apa yang dicetak ==, dan apa yang dicetak is?

3.8 Menjalankan Program Anda

Tidak ada yang berubah dari proses yang diperkenalkan Bab 1-2: python3 -m py_compile memeriksa kedua file untuk error sintaks, dan python3 library_demo.py meng-import pustaka.py sekaligus menjalankan kode library_demo.py sendiri. Pernyataan match tidak butuh flag khusus pada Python mana pun mulai versi 3.10 (baseline CPython 3.11-3.14 mata kuliah ini jauh melampaui ambang itu).

```
$ python3 -m py_compile pustaka.py library_demo.py
$ python3 library_demo.py
```

3.9 OOP dalam Praktik: Profitina

Tentang kotak ini

Seperti di setiap bab, kotak ini menghubungkan materi mata kuliah dengan Profitina, platform AI-visibility (Generative Engine Optimization) Indonesia yang nyata dan aktif dibangun oleh penulis -- menjelaskan DI MANA dan MENGAPA sebuah konsep dipakai dalam sistem sungguhan yang berjalan, tidak pernah logika bisnis proprietary atau kode sumber lengkap Profitina, yang tetap rahasia. Lihat dokumen pendamping "Profitina: A Non-Confidential Technical Overview" untuk gambaran lengkapnya.

Proses rilis Profitina sendiri dijaga gerbang oleh rangkaian tes otomatis lebih dari 978 tes (pytest) yang harus lulus penuh sebelum perubahan kode apa pun mencapai server produksi -- persis disiplin yang

dilayani perbedaan assert-vs-input Bagian 3.6 dan teknik biseksi, diskalakan dari program satu mahasiswa ke produk sungguhan yang dipakai bisnis nyata. Tes yang gagal dalam rangkaian itu didiagnosis dengan alat inti yang persis sama yang diperkenalkan bab ini: memeriksa nilai tertentu pada titik tertentu, mempersempit di mana hasil tak terduga pertama kali muncul, dan tidak pernah memercayai asumsi yang belum benar-benar diperiksa.

Backend Profitina juga melakukan percabangan dan kategorisasi bertingkat substansial secara internal -- misalnya, memutuskan bagaimana menyajikan hasil AI-visibility sebuah bisnis melibatkan perbandingan nilai terhitung terhadap sejumlah kecil ambang batas, secara konseptual bentuk yang sama dengan tingkatan calculate_fine bab ini, hanya diterapkan pada masalah berbeda. Dan karena mesin AI-visibility Profitina mem-parsing teks yang dihasilkan mesin jawaban AI eksternal untuk mencari penyebutan merek, ia terus-menerus dan dengan benar bergantung persis pada perbandingan string berbasis-nilai yang dijelaskan Bagian 3.5 -- tidak pernah pada perbandingan identitas string dengan is, yang ditunjukkan bagian yang sama sebagai pertanyaan berbeda yang mudah tertukar, yang harus diselesaikan jalur Java (Bab 3, Bagian 3.5) dengan .equals().

3.10 Ringkasan Bab dan Poin Penting

- Tipe bawaan inti Python -- int, float, bool, str, complex -- tidak punya ukuran tetap yang diekspos Python ke Anda; bool secara teknis adalah subclass dari int, dengan konsekuensi aritmetika nyata.
- if/elif/else dan match keduanya menyandikan percabangan; match adalah PERNYATAAN (berbeda dari ekspresi switch Java) dan mengomunikasikan "satu nilai, beberapa case tertentu" secara langsung ketika berlaku.
- Python menawarkan tepat dua kata kunci loop, for dan while, dan tidak punya for berhitung bergaya-C atau do-while; for selalu beriterasi atas iterable, tidak pernah counter polos yang ditambah manual.
- String bersifat immutable; == selalu membandingkan konten di Python (tanpa perlu .equals()), sementara is membandingkan identitas dan tidak pernah aman diandalkan untuk string, bahkan ketika interning CPython kebetulan membuatnya tampak berfungsi.
- Debugging print(), pdb/breakpoint(), dan assert masing-masing cocok untuk situasi berbeda; biseksi melokalisasi bug tak diketahui dengan cepat; assert mendokumentasikan invarian kode ANDA sendiri, dihapus sepenuhnya di bawah -O, dan tidak pernah pengganti validasi input tak terpercaya.
- Book mendapat calculate_fine(int) -> float dan renewal_status_label() -> str bab ini, tanpa menambah attribute baru -- kedua method adalah fungsi murni dari state yang sudah dimiliki Book.

Setiap alat yang diperkenalkan bab ini -- tipe bawaan, percabangan, loop, perbandingan string, teknik debugging -- sudah ada di setiap program yang pernah Anda tulis sejauh ini, dipakai tanpa disebut namanya. Menamainya adalah yang memungkinkan Anda menalarkannya secara sengaja, bukan secara kebetulan.

3.11 Pertanyaan Tinjauan

Pertanyaan-pertanyaan ini untuk belajar mandiri dan diskusi kelas; tidak dinilai. Kerjakan sebelum Lecture 4 (Kuis 1, mencakup Lecture 1-3).

1. Tulis ulang rantai if/elif/else calculate_fine sebagai ekspresi kondisional bersarang (a if kondisi else b). Apakah hasilnya lebih atau kurang terbaca dibanding versi if/elif/else? Justifikasi jawaban Anda.
2. Jelaskan, secara presisi, mengapa 0.1 + 0.2 == 0.3 bernilai False di Python, dan sebutkan cara yang benar untuk membandingkan dua nilai float untuk kesamaan "cukup dekat".
3. Apa yang dicetak print("exit" is "exit")? Apa yang dicetak print("exit" is "".join(["e","x","i","t"]))? Jelaskan perbedaannya memakai Gambar 3.4, dan mengapa mengandalkan salah satu hasil itu adalah kesalahan.
4. Tulis method singkat berbasis-loop count_overdue_tiers(self, days_list: list[int]) -> int yang mengembalikan berapa banyak hari dalam list jatuh ke tingkat denda 8-30 hari. Bentuk loop mana (for atau while) paling cocok, dan mengapa?
5. Bagian 3.6 mengatakan assert tidak boleh pernah memvalidasi input tak terpercaya. Tulis ulang baris ini agar benar menolak argumen days_overdue negatif ke calculate_fine bahkan ketika Python dijalankan dengan -O: assert days_overdue >= 0

Lampiran 3.A — Log Verifikasi Eksekusi

File pustaka.py dan library_demo.py lengkap (Code/Python/Chapter03/, disediakan bersama buku ini) diperiksa sintaksnya dan dijalankan pada CPython 3.11.15 sebelum bab ini ditulis, untuk memastikan kode benar, bukan sekadar "terlihat benar". Output direproduksi verbatim di bawah.

```
$ python3 -m py_compile pustaka.py library_demo.py
(tidak ada error)

$ python3 library_demo.py
"Clean Code" by Robert C. Martin [ISBN 978-0132350884, 2008] - on the shelf
Renewal status: Renewed once
Overdue  0 day(s) -> fine: 0.00
Overdue  3 day(s) -> fine: 1.50
Overdue 15 day(s) -> fine: 11.50
Overdue 45 day(s) -> fine: 50.00
== comparison: True
is comparison: False
```

Referensi

[1] Python Software Foundation. "Built-in Types." https://docs.python.org/3/library/stdtypes.html (diakses Agustus 2026).

[2] Python Software Foundation. "PEP 634 — Structural Pattern Matching: Specification." https://peps.python.org/pep-0634/ (diakses Agustus 2026) — pernyataan match, diperkenalkan di Python 3.10.

[3] Python Software Foundation. "CPython's implementation of the -O flag and assert removal." https://docs.python.org/3/using/cmdline.html#cmdoption-O (diakses Agustus 2026).

[4] R. C. Martin. Clean Code: A Handbook of Agile Software Craftsmanship. Prentice Hall, 2008 (sumber judul contoh berjalan bab ini).

BAB 4 · BAB LATIHAN

Soal Latihan: Kelas, Field, Enkapsulasi & Dasar Bahasa

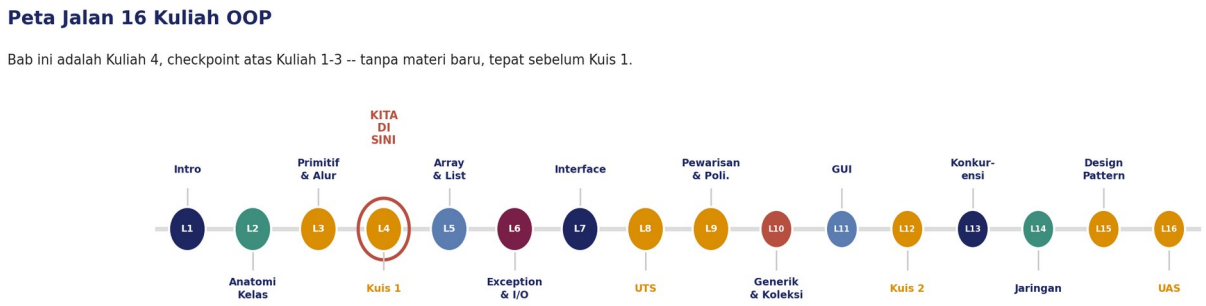
Tidak ada materi baru di sini -- hanya tujuh soal yang dirancang untuk memastikan Kuliah 1 sampai 3 benar-benar melekat, sebelum Kuis 1.

Tujuan Pembelajaran — setelah mengerjakan bab ini Anda akan mampu:

(1) Menjelaskan, dengan kata-kata Anda sendiri, apa itu kelas, objek, atribut, dan konstruktor, serta mengapa enkapsulasi penting. (2) Menulis kelas dari spesifikasi bahasa biasa: memilih atribut yang tepat, menulis __init__ yang menetapkan semuanya, dan mengekspos state hanya lewat method atau property. (3) Menulis fungsi atau method yang melakukan aritmetika dengan benar, termasuk nilai yang harus dinormalisasi ke rentang tetap. (4) Menulis method yang mengembalikan instance baru dari kelasnya sendiri, bukan memutasi yang sudah ada. (5) Menulis dua kelas yang berkolaborasi -- method satu kelas memanggil method pada instance kelas lain.

4.1 Rekap: Kuliah 1–3 Secara Singkat

Kuliah 1 memperkenalkan pergeseran dari pemikiran prosedural ke pemikiran berorientasi objek -- membungkus data dan perilaku yang mengoperasikannya menjadi satu unit, sebuah kelas, alih-alih mengoper data mentah antar fungsi yang berdiri sendiri. Kuliah 2 memberi anatomi presisi pada ide itu: atribut menyimpan state sebuah objek, __init__ menginisialisasi state itu tepat sekali saat pembuatan, dan enkapsulasi (atribut berawalan underscore, @property, method public) adalah yang menjaga kode luar agar tidak bisa masuk dan merusak state itu secara langsung. Kuliah 3 kemudian melengkapi dasar-dasar bahasa Python di bawah semua itu -- tipe numerik, aturan aritmetika dan normalisasi yang mengaturnya, alur kontrol, string, dan alat debugging dasar yang Anda pakai ketika output program tidak sesuai harapan (Gambar 4.1).



Gambar 4.1 — Bab ini berada di Kuliah 4 dalam peta jalan 16-kuliah OOP: sebuah checkpoint, bukan topik baru, tepat sebelum Kuis 1.

4.2 Cara Menggunakan Bab Latihan Ini

> **Apa bab ini -- dan bukan apa**
>
> Ini adalah Bab Latihan: bab praktik, bukan bab materi baru. Setiap soal di bawah berakar pada materi spesifik dari Kuliah 1–3 (lihat Gambar 4.2). Setiap soal disertai PETUNJUK -- dorongan ke arah cara berpikir yang tepat -- tapi sengaja BUKAN solusi lengkap atau kode sumber. Mengerjakan soal itu sendiri, sekalipun tidak sempurna, mengajarkan jauh lebih banyak daripada membaca jawaban yang sudah jadi.

Asal Setiap Soal Latihan

Setiap soal di Bagian 4.3 dapat ditelusuri kembali ke salah satu dari tiga kuliah sebelumnya.

#	Soal	Berasal dari
1	Kalkulator Pecahan	K3 -- aritmetika primitif, fungsi biasa
2	Sudut Jam Analog	K3 -- aritmetika float, modulo, normalisasi
3	Kelas Akun Komputer	K2 -- atribut, konstruktor, enkapsulasi
4	Kelas Bilangan Kompleks	K2 -- atribut, method yang mengembalikan objek baru
5	Kelas Konversi Berat	K2 -- konstruktor, getter, nilai turunan
6	Kelas Pecahan Lanjut	K2/K3 -- __str__, method yang mengembalikan tipe kelasnya sendiri
7	Kelas Player dan Enemy	K2 -- dua kelas berkolaborasi, mutasi lewat method

Gambar 4.2 — Setiap soal di Bagian 4.3 dapat ditelusuri kembali ke salah satu dari tiga kuliah sebelumnya.

> **Sebelum Anda mulai**
>
> Coba setiap soal sungguh-sungguh dalam file .py baru sebelum membaca petunjuknya. Jika buntu, baca hanya petunjuknya -- bukan solusi -- lalu coba lagi. Ini persis disiplin yang akan dihargai oleh Kuis 1 di Bagian 4.4: kuisnya open-book, tapi itu bukan pengganti penalaran Anda sendiri.

4.3 Soal Latihan

4.3.1 Aritmetika Primitif & Alur Kontrol

Soal 1 ***[Mudah]***. Misalkan e1/d1 mewakili satu pecahan dan e2/d2 pecahan kedua, dengan e1 dan e2 bilangan bulat serta d1 dan d2 bilangan bulat positif. Tulis fungsi compute(e1, d1, e2, d2) yang mengembalikan es/ds (jumlah) dan ep/dp (hasil kali) dari kedua pecahan, menggunakan rumus standar: es/ds = (e1*d2 + e2*d1) / (d1*d2), dan ep/dp = (e1*e2) / (d1*d2). Kedua hasil tidak perlu disederhanakan. Uji fungsi Anda pada pasangan 1/2 & 1/3, 1/3 & 3/4, 1/2 & 2/3, dan 1/4 & 2/3.

> **Petunjuk**
>
> Soal sudah memberi Anda kedua rumus secara langsung. Return tuple dan unpacking Python (return es, ds, ep, dp) adalah cara idiomatis untuk mengembalikan empat angka terkait tanpa perlu kelas dulu -- Soal 6 di bawah adalah tempat kelas Fraction sungguhan seharusnya berada.

Soal 2 ***[Mudah]***. Tulis fungsi angle_between(hours, minutes) yang menghitung sudut berlawanan arah jarum jam, dalam derajat bulat yang dinormalisasi ke 0-359, dari jarum jam ke jarum menit pada jam

analog tradisional. Ingat: jarum menit bergerak 6 derajat per menit, dan jarum jam merayap maju 0,5 derajat per menit selain 30 derajat per jam. Uji pada 9:00, 3:00, 18:00, 1:00, 2:30, dan 4:41.

> **Petunjuk**
>
> Hitung sudut jarum jam termasuk rayapan 0,5 derajat per menitnya, kurangi dengan sudut jarum menit, lalu normalisasi dengan modulo float. Pada 4:41 sudut mentahnya persis 254,5 -- pikirkan baik-baik apakah round() bawaan Python (pembulatan banker's) atau int() (pemotongan ke arah nol) yang benar dan bisa dipertanggungjawabkan di sini, bukan sekadar yang kebetulan menghasilkan angka yang diharapkan untuk satu input ini.

4.3.2 Atribut, Konstruktor & Enkapsulasi

Soal 3 *[Mudah]*. Definisikan kelas ComputerAccount, dibangun dari tiga string: real_name, user_name, dan password. Implementasikan print_real_name(), print_user_name(), dan print_password() (tanpa argumen), plus change_password(new_password) (tanpa nilai kembali).

> **Petunjuk**
>
> Ini adalah kelas konstruktor-plus-method-akses paling sederhana yang mungkin: tiga atribut ditetapkan di __init__, satu method per atribut untuk mencetaknya, dan satu method yang memutasi satu atribut. Tidak ada di sini yang seharusnya membutuhkan konsep baru di luar Kuliah 2.

Soal 4 *[Sedang]*. Definisikan kelas ComplexNumber, menyimpan bagian real dan bagian imajiner, dengan konstruktor dan getter. Implementasikan add, subtract, dan multiply (masing-masing mengembalikan ComplexNumber baru), serta __str__ yang menghasilkan "a + bi". Gunakan: (a+bi)+(c+di) = (a+c)+(b+d)i; (a+bi)-(c+di) = (a-c)+(b-d)i; (a+bi)*(c+di) = (ac-bd)+(ad+bc)i.

> **Petunjuk**
>
> add, subtract, dan multiply masing-masing menerima satu argumen ComplexNumber dan harus mengembalikan ComplexNumber BARU, bukan memodifikasi salah satu operand -- ketiga rumus aritmetika sudah diberikan langsung di soal, jadi kerja desain di sini sepenuhnya soal tipe kembalian, bukan matematikanya.

Soal 5 *[Mudah]*. Tulis kelas Weight yang menerima pounds di konstruktornya, dengan get_pounds() dan get_kilograms() (menggunakan 1 pound = 0,45359237 kilogram).

> **Petunjuk**
>
> Simpan hanya pounds sebagai atribut. get_kilograms() harus menghitung konversi sesuai permintaan dari satu atribut itu, bukan menyimpan atribut kilogram kedua yang redundan yang bisa jadi tidak sinkron jika pounds pernah diubah. Konstanta konversinya sendiri sebaiknya berada di level modul, bukan sebagai atribut instance -- ia menjelaskan aturan konversi, bukan milik satu Weight tertentu.

4.3.3 Menggabungkan Semuanya

Soal 6 *[Sedang]*. Definisikan kelas Fraction dengan konstruktor, get_numerator/get_denominator, __str__ (mis. "1/2"), serta get_sum/get_product, masing-masing mengembalikan Fraction baru. Untuk f1 = Fraction(1, 2) dan f2 = Fraction(3, 7): str(f1) harus mengembalikan "1/2"; f2.get_product(f1) harus mencetak 3/14; f2.get_sum(f1) harus mencetak 13/14.

> **Petunjuk**
>
> Ini adalah dua rumus Soal 1 lagi, tapi kali ini get_sum dan get_product masing-masing harus mengembalikan objek Fraction yang utuh, bukan tuple mentah -- sehingga hasilnya bisa dijumlahkan atau dikalikan lagi tanpa perlu membongkar apa pun terlebih dahulu.

Soal 7 ***[Sedang]***. Definisikan dua kelas, Player dan Enemy, masing-masing dengan name, health, power, dan defense, plus konstruktor, getter, dan setter. Implementasikan attack(opposing) (pihak lawan menerima damage sebesar power pihak ini) dan take_damage(opponent_attack) (health pihak ini berkurang sebesar opponent_attack - self.defense; jika health turun di bawah nol, cetak "{name} died!").

> **Petunjuk**
>
> attack() sebaiknya hanya menyerahkan SATU angka ke lawan -- power pihak ini -- dan biarkan take_damage() milik lawan sendiri yang menerapkan defense-nya sendiri dan memutuskan apakah health sudah turun di bawah nol. Jangan menjangkau atribut lawan secara langsung dari dalam attack(); kedua kelas seharusnya hanya berbicara lewat method publiknya masing-masing.

4.4 Format Kuis 1: Open-Book, Individu, Berwaktu

Kuis 1 adalah penilaian take-home open-book, individu, dan berwaktu (sekitar 90 menit) yang mencakup persis tujuh jenis soal yang direview di bab ini -- tidak lebih. Penilaian dilakukan per-soal, bukan semua-atau-tidak-sama-sekali: nilai parsial diberikan untuk kode yang berjalan dan menangani kasus umum dengan benar sekalipun satu edge case terlewat.

Open-book berarti referensi, bukan kolaborator

Anda boleh berkonsultasi dengan buku teks, catatan Anda sendiri, dan slide kuliah saat menjawab. Anda TIDAK BOLEH berkolaborasi dengan teman sekelas atau mengumpulkan kode yang bukan benar-benar milik Anda sendiri -- kuis open-book menguji apakah Anda bisa MENERAPKAN apa yang sudah Anda pelajari, tanpa pengawasan.

Kriteria	Yang dinilai	Bobot
Kebenaran output	Apakah kode menghasilkan hasil yang persis sesuai harapan untuk setiap kasus uji yang diberikan?	50%
Desain kelas & enkapsulasi	Apakah atribut berawalan underscore, dan state hanya bisa dijangkau lewat konstruktor dan method?	25%
Kejelasan kode	Apakah kode mudah dibaca, idiomatis, dan bebas dari kode mati atau duplikat?	15%
Berjalan bersih	Apakah kode berjalan di python3 tanpa error dan tanpa exception yang tidak ditangani?	10%

4.5 OOP dalam Praktik: Profitina

Tentang kotak ini

Seperti di setiap bab, kotak ini menghubungkan materi kuliah dengan Profitina, platform AI-visibility (Generative Engine Optimization) Indonesia yang nyata dan aktif, dibangun oleh penulis -- menjelaskan DI MANA dan MENGAPA sebuah konsep digunakan dalam sistem nyata yang berjalan, tidak pernah logika bisnis atau kode sumber lengkap Profitina, yang tetap rahasia.

Setiap satu dari tujuh soal bab ini mengikuti bentuk yang sama dengan model data backend Python/FastAPI Profitina sendiri pada skala yang jauh lebih besar: atribut berawalan underscore, konstruktor yang menginisialisasi semuanya secara penuh, dan method atau property yang menjadi satu-satunya cara sah untuk membaca atau mengubah state itu dari luar kelas. Sebelum sebuah perubahan naik ke produksi di Profitina, engineer menjalankannya melalui checklist self-review singkat -- mirip sekali dengan Lampiran 4.A di bawah -- sebelum meminta pasang mata kedua: apakah perubahan itu menangani input persis yang dirancang untuknya, dan sudahkah diperiksa terhadap edge case yang mudah terlewat sekilas mata? Kebiasaan memverifikasi penalaran Anda sendiri sebelum review eksternal itulah yang membuat format kuis take-home open-book berfungsi, dan itulah tepatnya mengapa kasus 4:41 Soal 2 dan pengecekan health-di-bawah-nol Soal 7 keduanya penting jauh melampaui satu bab latihan ini.

4.6 Ringkasan Bab

- Bab ini tidak memperkenalkan materi baru -- ini adalah checkpoint yang mencakup Kuliah 1 sampai 3.
- Tujuh soal mencakup seluruh rentang yang direview: aritmetika primitif dan normalisasi (Kuliah 3), serta atribut, konstruktor, enkapsulasi, dan method yang mengembalikan objek baru (Kuliah 2).
- Setiap soal menyertakan petunjuk, bukan solusi -- mengerjakan penalaran dan kodenya sendiri adalah intinya.
- Kuis 1 adalah penilaian take-home open-book, individu, berwaktu: referensi diperbolehkan, tapi kodenya harus milik Anda sendiri, dan kebenaran pada setiap kasus uji yang diberikan adalah mayoritas nilai.

Kode yang berjalan tanpa error tidak sama dengan kode yang benar -- bedanya selalu ada pada kasus uji yang tidak terpikirkan oleh Anda untuk dicoba.

Lampiran 4.A — Checklist Self-Check (Tidak Dinilai)

Sebelum mengumpulkan jawaban apa pun -- pada soal bab ini atau pada Kuis 1 itu sendiri -- jalankan lewat checklist ini. Butuh waktu beberapa menit dan menangkap sebagian besar kesalahan yang paling sering dilihat penilai.

- Apakah setiap file berjalan di python3 tanpa exception yang tidak ditangani?
- Apakah setiap atribut berawalan underscore, dengan setiap nilai yang dibutuhkan pemanggil diekspos hanya lewat method atau @property?
- Apakah __init__ setiap kelas menetapkan setiap atribut yang dipakainya -- tidak ada atribut yang baru ditetapkan belakangan oleh method lain?
- Apakah method aritmetika Soal 4 dan 6 mengembalikan objek BARU alih-alih memutasi salah satu operand?
- Sudahkah Anda menguji setiap nilai contoh yang diberikan soal, bukan hanya yang kebetulan berupa angka bulat (kasus 4:41 Soal 2 sengaja ada untuk menangkap bug pembulatan yang tidak bisa diungkap oleh 9:00 dan 3:00)?
- Sudahkah Anda membaca ulang kode Anda sendiri seolah-olah Anda penilai skeptis yang mencari satu input yang bisa merusaknya?

Referensi

[1] Bab latihan ini dimodelkan dari Kuis 1 nyata mata kuliah ini (EF234302 Pemrograman Berorientasi Objek, tugas take-home open-book individu) -- tujuh soal yang sama, bobot poin yang sama.

[2] Python Software Foundation. "Classes." The Python Tutorial. https://docs.python.org/3/tutorial/classes.html (diakses Agustus 2026).

BAB 5

List & Kisah Membangun String

Setiap objek Book, Player, atau Fraction sejauh ini berdiri sendiri. Program nyata menyimpan banyak objek sekaligus -- seluruh rak buku sebuah perpustakaan, bukan satu buku saja. Bab ini memperkenalkan satu-satunya wadah bawaan Python yang bisa tumbuh, mengapa Python tidak memiliki array berukuran tetap yang terpisah, dan jebakan klasik (serta solusinya) membangun string panjang sepotong demi sepotong (Gambar 5.1).

Tujuan Pembelajaran — setelah bab ini Anda akan mampu:

(1) Membuat list, dan menggunakan append, remove, indexing, dan len() untuk mengelola koleksi yang tumbuh. (2) Menjelaskan mengapa Python tidak memiliki tipe array berukuran tetap yang terpisah seperti Java, dan apa artinya dalam praktik. (3) Menjelaskan mengapa penggabungan string berulang dengan + di dalam loop bersifat kuadratik, dan memperbaikinya dengan idiom kumpulkan-lalu-''.join(). (4) Menyebutkan mengapa Python tidak memiliki kelas StringBuilder publik, dan idiom apa yang menggantikannya. (5) Memperluas Pustaka dengan kelas Library yang mengelola koleksi objek Book, menjalankannya, dan memverifikasinya sendiri.

5.1 Dari Satu Buku ke Satu Rak Buku

Bab 1 hingga 4 bekerja dengan objek individual: satu Book, satu Fraction, satu Player. Setiap perpustakaan nyata, tentu saja, menyimpan banyak buku sekaligus, dan program yang hanya bisa menyimpan satu objek Book belum menjadi sistem perpustakaan sama sekali -- itu baru demonstrasi satu Book. Bab ini menutup celah tersebut dengan memperkenalkan list Python, satu-satunya wadah bawaan yang dibutuhkan mata kuliah ini untuk koleksi objek, dan sekaligus bertemu dengan masalah kedua yang tampak tidak berhubungan -- membangun sepotong teks panjang sedikit demi sedikit -- yang ternyata berbagi jebakan biaya dasar yang sama persis, dan bentuk solusi yang sama persis, dengan versi Java dari kisah yang sama ini.

Gambar 5.1 — Peta jalan 16 kuliah OOP. Bab ini adalah Kuliah 5, pemberhentian pertama setelah Kuis 1 dan awal bekerja dengan koleksi objek.

5.2 List Python: Satu Wadah, Selalu Bisa Tumbuh

List Python dibuat dengan tanda kurung siku -- shelf = [] dimulai kosong -- dan tidak perlu panjang dideklarasikan di awal: shelf.append(book) menumbuhkannya satu setiap kali dipanggil, tanpa batas

atas yang akan tercapai. Berbeda dengan Java, Python tidak memiliki tipe array berukuran tetap yang terpisah dalam penggunaan biasa; tipe list yang sama melayani setiap situasi yang di Java akan dibagi antara array dan ArrayList.

Ini adalah perbedaan desain bahasa yang sesungguhnya, bukan fitur yang hilang: array Java ada sebagai primitif berukuran-tetap level rendah yang sengaja dibuat, tempat ArrayList dibangun di atasnya secara internal; Python justru hanya mengekspos versi dinamisnya secara langsung, karena dalam praktik hampir setiap koleksi nyata -- termasuk rak buku Pustaka -- perlu tumbuh dan menyusut, dan kasus berukuran-tetap cukup jarang sehingga Python tidak memberinya sintaks khusus (Gambar 5.2).

Array Berukuran Tetap vs. List yang Bisa Tumbuh

Array Java dan ArrayList Java, serta satu-satunya list bawaan Python, dibandingkan pada properti yang penting sehari-hari.

Properti	Array Java (Book[])	ArrayList<Book> Java	List Python
Ukuran	`Tetap sejak dibuat`	`Tumbuh/menyusut otomatis`	`Tumbuh/menyusut otomatis`
Deklarasi + buat	`Book[] b = new Book[10];`	`List<Book> b = new ArrayList<>();`	`b = []`
Tambah elemen	`b[i] = book;  (i harus ada)`	`b.add(book);`	`b.append(book)`
Hapus elemen	`tidak didukung langsung`	`b.remove(book);`	`b.remove(book)`
Ukuran saat ini	`b.length`	`b.size()`	`len(b)`
Tipe elemen	`Tipe apa pun, termasuk primitif`	`Hanya objek (bukan int; pakai Integer)`	`Tipe apa pun, boleh campur`

Array Java adalah kontainer level paling rendah -- cepat dan sederhana, tapi panjangnya terkunci selamanya.
ArrayList membungkus array yang bisa diubah ukurannya secara internal; list Python dinamis secara default, tanpa saudara berukuran-tetap yang terpisah.

Gambar 5.2 — Array dan ArrayList Java dibandingkan dengan satu-satunya list bawaan Python. Perhatikan len() dan append() milik Python terbaca hampir identik dengan size() dan add() milik ArrayList.

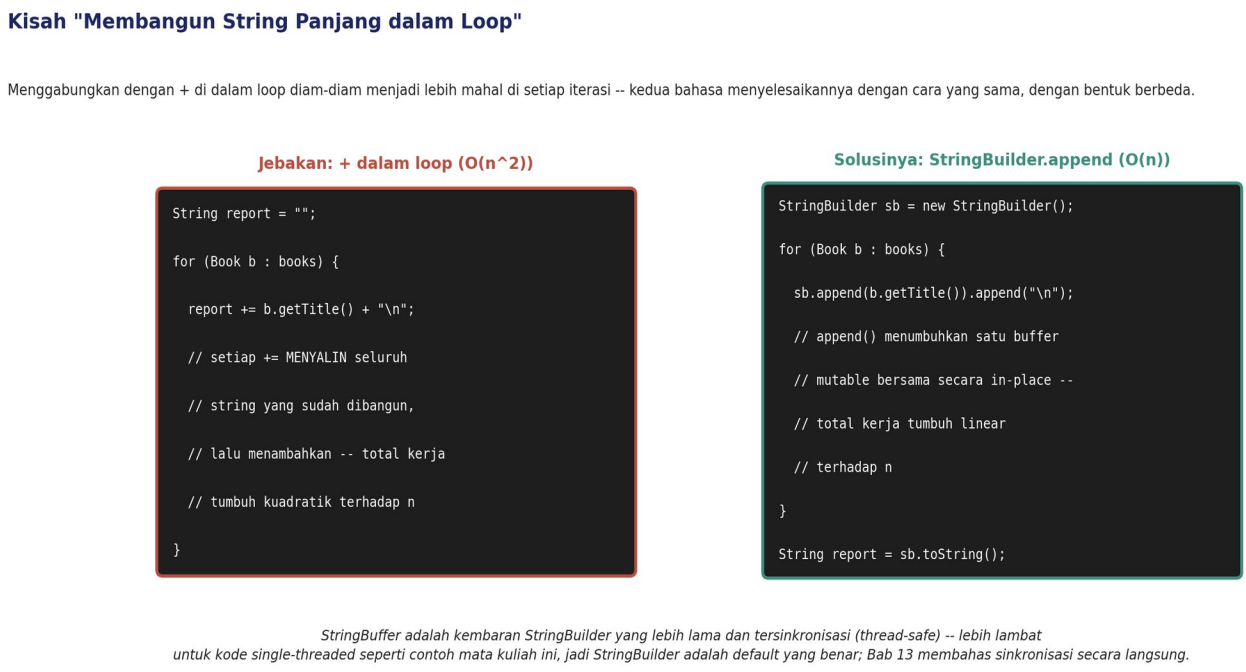
5.3 Operasi Inti List

shelf.append(book) menambahkan ke akhir; shelf[i] membaca (atau, berbeda dengan batasan panjang-tetap array Java, juga bisa menetapkan) item di indeks i; shelf.remove(book) menghapus berdasarkan nilai (item pertama yang cocok); len(shelf) melaporkan jumlah saat ini. Loop for b in shelf: biasa mengiterasi list secara langsung -- tidak perlu variabel indeks kecuali indeksnya sendiri penting, sebuah keunggulan keterbacaan kecil yang dimiliki list Python dibanding akses berindeks pada kasus umum.

> **Satu list, tipe apa pun -- bahkan campuran**
>
> Berbeda dengan List<Book> milik Java, yang dipaksakan oleh compiler untuk hanya menyimpan objek Book, list Python bisa mencampur tipe apa pun sekaligus -- [1, "two", Book(...)] adalah sintaks yang sepenuhnya legal. Dalam kode yang dirancang dengan baik Anda hampir selalu tetap menjaga satu list untuk satu tipe item secara konseptual (persis seperti Library._books di bawah, hanya menyimpan objek Book) -- Python hanya saja tidak memaksakan disiplin itu untuk Anda seperti yang dilakukan generik Java; itu adalah konvensi yang Anda pilih untuk diikuti, bukan aturan yang diperiksa bahasa.

5.4 Kisah Membangun String

String, seperti yang ditetapkan Bab 3, bersifat immutable: setiap += pada str tidak mengubahnya secara in-place tetapi membuat objek string yang sama sekali baru dan menyalin konten lama ke dalamnya. Di dalam loop yang berjalan n kali, itu berarti iterasi pertama menyalin string kecil, iterasi kedua menyalin string yang sedikit lebih panjang, dan seterusnya -- total kerja penyalinan tumbuh proporsional terhadap n^2 (kuadratik), bukan n, meskipun loop itu sendiri hanya berjalan n kali. Ini persis kisah StringBuilder milik Java (Bab 5 Java, Bagian 5.4) -- jebakan biaya dasarnya identik di kedua bahasa, karena berasal dari sifat immutable string, bukan dari desain khusus satu bahasa tertentu (Gambar 5.3).



Gambar 5.3 — Loop pembangunan laporan yang sama ditulis dengan cara yang mahal (+= dalam loop) dan solusi Java, StringBuilder.append(). Python tidak memiliki kelas builder publiknya sendiri -- lihat kotak di bawah untuk idiom asli Python.

Solusi Python: kumpulkan potongan dalam list, gabungkan sekali

Solusi Java adalah objek StringBuilder yang mutable; solusi idiomatis Python tampak berbeda tetapi menyelesaikan masalah yang sama persis: kumpulkan setiap potongan ke dalam list biasa (pieces.append(next_piece) -- append pada list sudah O(1) amortized, tidak perlu builder khusus), lalu panggil "".join(pieces) tepat sekali di akhir. join() mengetahui total panjangnya sebelumnya dan mengalokasikan string akhir sekali, sehingga seluruh operasi bersifat linear (O(n)), persis seperti StringBuilder.append() yang diikuti satu toString(). io.StringIO milik Python menawarkan buffer mutable yang lebih mirip StringBuilder Java untuk kasus teks yang sangat besar atau streaming, tetapi untuk kasus biasa yang dibahas bab ini, kumpulkan-lalu-join adalah idiom yang akan Anda lihat di hampir semua kode Python nyata.

5.5 Memperluas Pustaka: Kelas Library

Library, diperkenalkan di bab ini, mengelola list[Book] dan mempraktikkan Bagian 5.3 dan 5.4 bersama-sama: add_book dan remove_book mengelola koleksi, dan generate_report membangun ringkasan multi-baris dari seluruh rak menggunakan idiom kumpulkan-lalu-join alih-alih +=.

Fase 5 — Implement (library.py, diringkas)

```
class Library:
    def __init__(self) -> None:
        self._books: list[Book] = []

    def add_book(self, book: Book) -> None:
        self._books.append(book)

    def remove_book(self, isbn: str) -> bool:
        for i, b in enumerate(self._books):
            if b.isbn == isbn:
                del self._books[i]
                return True
        return False

    def generate_report(self) -> str:
        lines = [f"Pustaka Library Report ({len(self._books)} book(s))"]
        for b in self._books:
            status = " [on loan]" if b.is_on_loan else " [on shelf]"
            lines.append(f"- {b.title} by {b.author}{status}")
        return "\n".join(lines) + "\n"
```

Perhatikan remove_book menggunakan enumerate(self._books) untuk mendapatkan indeks dan item sekaligus dalam satu loop -- cara idiomatis Python ketika butuh "posisi item yang cocok", di sini agar del self._books[i] bisa menghapusnya berdasarkan posisi, mencerminkan tepat loop for (int i = 0; ...) berindeks milik Library.removeBook Java, hanya dengan idiom Python sendiri untuk memasangkan indeks dengan nilainya.

5.6 Menjalankan Program

Tidak ada yang berubah dari prosesnya: python3 library_demo.py. library_demo.py mengimpor baik Book (dari pustaka) maupun Library (dari library) di bagian atas file -- sistem modul Python, satu file per kelas di sini, adalah padanan langsung dari kebutuhan Java akan setiap kelas dalam file .java yang dikompilasi sendiri-sendiri.

```
$ python3 library_demo.py
```

5.7 OOP dalam Praktik: Profitina

> **Tentang kotak ini**
>
> Seperti pada setiap bab, kotak ini menghubungkan materi kuliah dengan Profitina, platform AI-visibility (Generative Engine Optimization) Indonesia yang nyata dan aktif, dibangun oleh penulis -- menjelaskan DI MANA dan MENGAPA sebuah konsep digunakan dalam sistem nyata yang berjalan, bukan logika bisnis atau kode sumber lengkap Profitina yang bersifat rahasia.

Backend Python/FastAPI Profitina secara rutin menyimpan koleksi yang ukurannya tidak pernah diketahui sebelumnya -- sekumpulan kata kunci yang dilacak sebuah bisnis, sekelompok respons mesin jawaban AI yang diambil untuk satu analisis -- persis situasi yang menurut Bagian 5.2 dilayani secara native oleh list Python, tanpa alternatif berukuran-tetap untuk dipertimbangkan sama sekali. Dan karena lapisan pelaporan Profitina secara rutin menyusun teks yang lebih panjang -- ringkasan terformat dari hasil AI-visibility sebuah bisnis, misalnya -- ia mengikuti disiplin kumpulkan-lalu-join yang sama yang diperkenalkan Bagian 5.4: kumpulkan potongan-potongannya dulu, gabungkan sekali, jangan pernah menggabungkan dengan += di dalam loop.

5.8 Ringkasan Bab dan Poin-Poin Utama

- Python hanya memiliki satu tipe list bawaan, selalu bisa tumbuh -- tidak ada array berukuran tetap yang terpisah seperti Java, karena dalam praktik hampir setiap koleksi nyata perlu tumbuh dan menyusut.
- Operasi inti list adalah append, remove, indexing, dan len() -- loop for item in list: biasa mengiterasinya secara langsung.
- Berbeda dengan List<T> milik Java, list Python tidak dibatasi ke satu tipe oleh bahasa itu sendiri -- menjaga satu list untuk satu tipe konseptual adalah konvensi yang Anda pilih untuk diikuti, bukan sesuatu yang dipaksakan Python.
- str += di dalam loop bersifat kuadratik (O(n^2)) karena setiap += menyalin seluruh string yang sudah dibangun; mengumpulkan potongan dalam list dan memanggil ''.join() sekali di akhir bersifat linear (O(n)).
- Python tidak memiliki kelas setara StringBuilder yang publik; kumpulkan-lalu-join adalah solusi idiomatis di mana-mana dalam kode Python nyata (io.StringIO ada untuk kasus streaming yang sangat besar).
- Library mendapatkan add_book, remove_book, find_by_title, __len__, dan generate_report di bab ini -- Book itu sendiri tidak berubah sama sekali.

Setiap koleksi yang akan Anda kelola selama sisa mata kuliah ini -- rak sebuah perpustakaan, daftar pemain, sekelompok permintaan jaringan -- dibangun di atas persis dua gagasan yang diperkenalkan bab ini: wadah yang tumbuh bersama Anda, dan teks yang disusun sekali alih-alih disalin berulang-ulang.

5.9 Pertanyaan Tinjauan

Pertanyaan-pertanyaan ini untuk belajar mandiri dan diskusi kelas; tidak dinilai.

1. Bagian 5.2 versi Java menghabiskan ruang nyata untuk menjelaskan keterbatasan panjang-tetap array. Mengapa Bagian 5.2 bab ini tidak membutuhkan peringatan setara tentang list Python?
2. Tulis ulang Library.generate_report() menggunakan += di dalam loop alih-alih idiom kumpulkan-lalu-join. Untuk perpustakaan dengan 10.000 buku, kira-kira berapa kali lebih banyak total kerja penyalinan karakter yang dilakukan versi ini dibandingkan versi join()? (Petunjuk: bandingkan n vs. n^2 untuk n = 10.000.)
3. List Python secara legal bisa mencampur tipe ([1, "two", Book(...)]). Berikan satu alasan konkret mengapa Library._books tetap seharusnya tidak pernah melakukan ini, meskipun Python tidak akan mencegahnya.
4. Tambahkan method Library.count_on_loan() yang mengembalikan berapa banyak buku dalam koleksi yang sedang dipinjam, menggunakan loop for biasa dan property is_on_loan milik Book yang sudah ada.
5. Dengan kata-kata Anda sendiri, jelaskan apa itu io.StringIO, dan mengapa idiom kumpulkan-lalu-join biasa tetap lebih disukai untuk kasus sekecil laporan perpustakaan bab ini.

Lampiran 5.A — Log Verifikasi Eksekusi

Berkas pustaka.py, library.py, dan library_demo.py yang lengkap (Code/Python/Chapter05/, disediakan bersama buku ini) telah dijalankan pada Python 3.13 sebelum bab ini ditulis, untuk memastikan kode benar-benar benar, bukan sekadar "tampak benar". Outputnya direproduksi persis di bawah ini.

```
$ python3 library_demo.py
Library size after adding 3 books: 3
Pustaka Library Report (3 book(s))
- Clean Code by Robert C. Martin [on shelf]
- Effective Java by Joshua Bloch [on loan]
- The Pragmatic Programmer by Andrew Hunt [on shelf]
Removed The Pragmatic Programmer: True
Library size now: 2
Pustaka Library Report (2 book(s))
- Clean Code by Robert C. Martin [on shelf]
- Effective Java by Joshua Bloch [on loan]
```

Referensi

[1] Python Software Foundation. "Data Structures — More on Lists." The Python Tutorial. https://docs.python.org/3/tutorial/datastructures.html (diakses Agustus 2026).

[2] Python Software Foundation. "str.join()." The Python Standard Library. https://docs.python.org/3/library/stdtypes.html#str.join (diakses Agustus 2026).

[3] Python Software Foundation. "io — Core tools for working with streams (StringIO)." https://docs.python.org/3/library/io.html#io.StringIO (diakses Agustus 2026).

[4] J. Bloch. Effective Java, 3rd ed. Addison-Wesley, 2017 (judul kedua yang berjalan berdampingan dengan "Clean Code" untuk contoh multi-buku bab ini).

BAB 6

Penanganan Exception & File I/O

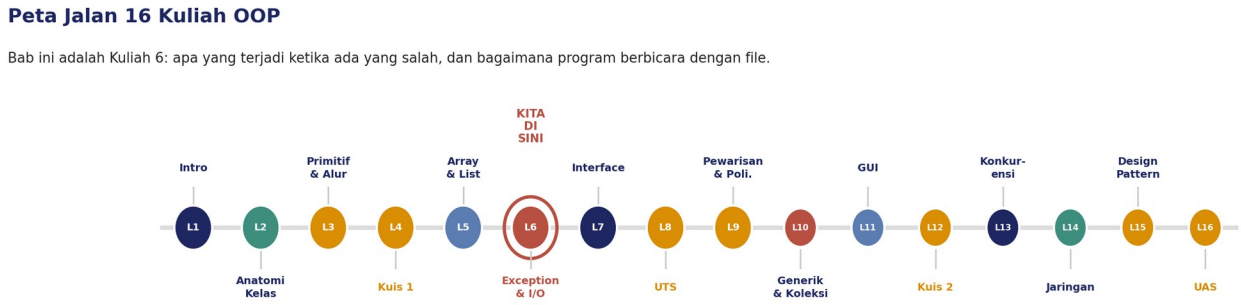
Setiap method sejauh ini mengasumsikan jalur mulus: input valid, file yang ada, ISBN yang benar-benar ada di perpustakaan. Program nyata tidak bisa berasumsi seperti itu. Bab ini memperkenalkan mekanisme exception Python untuk menangani kegagalan secara elegan, dan file I/O agar data Pustaka bisa bertahan lebih lama dari program yang membuatnya (Gambar 6.1).

Tujuan Pembelajaran — setelah bab ini Anda akan mampu:

(1) Menjelaskan model exception tunggal Python dan bagaimana perbedaannya dengan pemisahan checked/unchecked Java. (2) Menulis blok try-except-finally, dan menyatakan dengan tepat kapan blok finally berjalan. (3) Mendefinisikan exception kustom dengan subclass Exception, serta me-raise dan menangkapnya dengan benar. (4) Membaca dan menulis file teks biasa menggunakan pernyataan with (context manager), dan menjelaskan mengapa itu lebih disukai daripada memanggil close() secara manual. (5) Memperluas Pustaka dengan persistensi save/load dan method penghapusan yang lebih ketat yang me-raise exception, menjalankannya, dan memverifikasinya sendiri.

6.1 Dari Objek dalam Memori ke Program yang Bisa Gagal

Bab 1 sampai 5 membangun Pustaka yang bekerja sempurna -- selama tidak ada yang pernah salah. remove_book diam-diam mengembalikan False jika ISBN tidak ditemukan, dan setiap buku yang pernah dilihat program hanya hidup di memori: tutup programnya dan seluruh perpustakaan lenyap. Keduanya tidak bisa diterima dalam sistem nyata. Bab ini memperbaiki kedua celah itu: mekanisme exception Python memberi method cara yang prinsipil untuk menandakan "hal spesifik ini gagal," dan file I/O memberi Library cara untuk menyimpan buku-bukunya ke disk dan memuatnya kembali nanti.



Gambar 6.1 — Peta jalan 16 kuliah OOP. Bab ini adalah Kuliah 6: apa yang terjadi ketika ada yang salah, dan bagaimana program berbicara dengan file.

6.2 Model Exception Python: Satu Kategori, Tanpa Penegakan Compiler

Setiap kondisi error di Python direpresentasikan oleh sebuah objek -- pada akhirnya, setiap exception diturunkan dari BaseException. Kode biasa hampir tidak pernah menangkap BaseException secara langsung, karena turunan langsungnya yang paling utama termasuk hal-hal seperti SystemExit dan KeyboardInterrupt yang merepresentasikan "program sedang diminta berhenti," bukan error yang

perlu dipulihkan. Exception adalah cabang yang sungguh digunakan penanganan error biasa. Berbeda dengan Java, Python tidak menarik garis yang ditegakkan compiler antara exception "checked" dan "unchecked": tidak ada yang memaksa pemanggil untuk menangkap apa pun, atau mendeklarasikan di signature fungsi bahwa fungsi itu mungkin me-raise sesuatu. Pemanggil mengetahui apa yang bisa di-raise sebuah fungsi dengan membaca dokumentasinya (atau kode sumbernya) -- inilah gaya EAFP Python ("lebih mudah meminta maaf daripada izin"): coba operasinya, dan siap menangkap apa yang mungkin di-raise-nya, alih-alih memeriksa setiap precondition secara defensif terlebih dahulu (Gambar 6.2).

Checked Exception (Java) vs. Satu Model Python

Java memaksa pemanggil mengakui kegagalan tertentu saat kompilasi; Python memercayai pemanggil dan melempar semuanya dengan cara yang sama.

Properti	Java	Python
Kategori exception	`Checked (wajib ditangani/dideklarasikan) + Unchecked`	`Satu kategori -- tanpa perbedaan yang dipaksakan compiler`
Deklarasi method berisiko	`throws BookNotFoundException`	`tidak ada yang wajib -- pemanggil tahu dari membaca dokumentasi`
Menangkap	`catch (BookNotFoundException e) { ... }`	`except BookNotFoundError as e:`
Exception kustom	`extends Exception (checked) atau RuntimeException`	`class X(Exception): ...`
Cleanup terjamin	`try-with-resources / finally`	`pernyataan with (context manager) / finally`

Tidak ada model yang "lebih benar" -- checked exception menangkap penanganan error yang hilang saat kompilasi, dengan biaya seremoni; gaya EAFP Python ("lebih mudah meminta maaf daripada izin") memercayai developer untuk membaca dokumentasi dan menangkap apa yang sungguh perlu ditangkap.

Gambar 6.2 — Model checked-exception Java dibandingkan dengan model tunggal dan seragam Python. BookNotFoundError milik bab ini me-raise persis seperti exception Python lainnya.

> **Memercayai pemanggil adalah trade-off desain sungguhan, bukan jalan pintas**
>
> Checked exception Java menangkap jalur penanganan error yang terlupakan saat kompilasi, dengan biaya seremoni deklarasi throws yang terlihat di seluruh jalur Java mata kuliah ini. Model Python tidak punya jaring pengaman waktu-kompilasi semacam itu -- BookNotFoundError (Bagian 6.4) pada prinsipnya bisa di-raise oleh titik panggilan yang tidak pernah mengharapkannya, dengan error hanya muncul saat runtime. Desainer Python menilai ini sebagai trade-off yang bisa diterima demi bahasa yang lebih sederhana dan kurang seremonial; artinya kode Python memikul tanggung jawab yang sepadan lebih tinggi untuk mendokumentasikan apa yang bisa di-raise sebuah fungsi, dan untuk sungguh membaca dokumentasi itu.

6.3 try, except, else, dan finally

Blok try membungkus kode yang mungkin me-raise; satu atau lebih klausa except masing-masing menangani satu tipe exception spesifik; klausa else opsional berjalan hanya jika blok try selesai tanpa exception sama sekali; dan klausa finally opsional berjalan apa pun yang terjadi -- baik blok try selesai normal, me-raise exception yang tertangkap, atau me-raise yang tidak tertangkap dan akan menjalar lebih jauh ke atas call stack. (try-catch-finally Java tidak punya padanan langsung untuk else; ini adalah sedikit tambahan yang ditawarkan Python, untuk kode yang seharusnya berjalan hanya saat sukses.) (Gambar 6.3)

Gambar 6.3 — Jalur alur kontrol try/except/finally (struktur dasar yang sama dengan try/catch/finally Java). Sifat khas finally adalah ia selalu dieksekusi.

try-except-finally minimal

```
try:
    library.remove_book_or_raise(isbn)
except BookNotFoundError as e:
    print(f"Menangkap exception yang diharapkan: {e}")
finally:
    print("Ini selalu tercetak, exception atau tidak.")
```

except: telanjang hampir selalu salah

except: (tanpa tipe exception apa pun) menangkap secara harfiah segalanya, termasuk KeyboardInterrupt dan SystemExit, dan blok except kosong diam-diam membuang kegagalan itu sepenuhnya, yang jauh lebih buruk daripada membiarkan program crash dengan traceback yang jelas. Selalu tangkap tipe exception yang paling SPESIFIK yang kode Anda benar-benar bisa lakukan sesuatu yang berarti terhadapnya -- except Exception: hanya sedikit lebih baik, dan masih lebih luas dari yang dibutuhkan hampir semua handler sungguhan.

6.4 Exception Kustom: BookNotFoundError

Mendefinisikan exception kustom adalah pewarisan biasa: subclass Exception (jangan pernah Exception telanjang untuk apa pun yang Anda rencanakan ditangkap secara spesifik, dan jangan pernah subclass BaseException langsung), panggil super().__init__(message) di __init__, dan kelas baru itu adalah tipe exception yang sepenuhnya sah yang bisa di-raise dan ditangkap Python berdasarkan namanya.

Fase 6 — Implement (book_not_found_error.py)

```
class BookNotFoundError(Exception):
    """Raised when a lookup or removal by ISBN finds no matching book."""

    def __init__(self, isbn: str) -> None:
        super().__init__(f"No book found with ISBN {isbn}")
        self.isbn = isbn
```

Library.remove_book_or_raise (Bagian 6.6) me-raise exception ini ketika pencarian ISBN remove_book gagal, memberi pemanggil kondisi error yang bernama, spesifik, dan bisa ditangkap, alih-alih ValueError generik atau False yang diam-diam diabaikan.

6.5 File I/O dengan Pernyataan with

Membaca atau menulis file membuka resource sistem (file handle) yang HARUS ditutup ketika Anda selesai menggunakannya, atau resource itu bocor -- masalah nyata dalam program yang berjalan lama yang membuka banyak file sepanjang masa hidupnya. Pernyataan with Python, menggunakan objek apa pun yang mengimplementasikan protokol context manager (__enter__ dan __exit__), menjamin resource ditutup ketika blok with keluar, baik secara normal maupun lewat exception -- tidak perlu blok finally eksplisit. open() mengembalikan persis objek semacam itu, itulah mengapa with open(...) as f: adalah cara idiomatis bekerja dengan file di Python.

Pernyataan with vs. pola manual lama

```
# Python modern dan idiomatis: pernyataan with -- selalu utamakan ini
with open(path, "r", encoding="utf-8") as f:
    line = f.readline()
    # f.close() dipanggil otomatis di sini, terjamin

# Pola manual lama -- verbose dan mudah salah
f = open(path, "r", encoding="utf-8")
try:
    line = f.readline()
finally:
    f.close()  # mudah terlupakan, atau ditempatkan salah
```

OSError adalah exception terkait-file milik Python

Setiap fungsi yang menyentuh filesystem bisa gagal karena alasan yang sepenuhnya di luar kendali program Anda -- file yang hilang, disk penuh, error izin akses -- dan Python me-raise OSError (atau salah satu subclass-nya yang lebih spesifik, seperti FileNotFoundError) untuk persis situasi semacam ini. Berbeda dengan IOException Java, tidak ada yang memaksa save_to_file dan load_from_file di Bagian 6.6 untuk mendeklarasikan ini di signature-nya -- model exception seragam Python (Bagian 6.2) berlaku di sini persis seperti di mana pun.

6.6 Memperluas Pustaka: Persistensi dan Penghapusan yang Lebih Ketat

Library mendapat dua kemampuan terkait di bab ini: remove_book_or_raise, saudara yang lebih ketat dari remove_book (Bab 5) yang me-raise BookNotFoundError alih-alih diam-diam mengembalikan False; serta save_to_file/load_from_file, persistensi file teks sederhana yang dibangun di atas pernyataan with.

Fase 6 — Implement (library.py, dipersingkat)

```
def remove_book_or_raise(self, isbn: str) -> None:
    if not self.remove_book(isbn):
        raise BookNotFoundError(isbn)

def save_to_file(self, path: str) -> None:
    with open(path, "w", encoding="utf-8") as f:
        for b in self._books:
            f.write(f"{b.title}|{b.author}|{b.isbn}|{b.publication_year}\n")

def load_from_file(self, path: str) -> None:
    with open(path, "r", encoding="utf-8") as f:
        for line in f:
            title, author, isbn, year = line.rstrip("\n").split("|")
            self._books.append(Book(title, author, isbn, int(year)))
```

Setiap baris yang ditulis save_to_file adalah record sederhana yang dipisahkan pipe -- title|author|isbn|year -- sengaja dibuat sederhana daripada format serialisasi sungguhan (modul json, dan pekerjaan networking Bab 14, membahas itu dengan semestinya), karena inti bab ini adalah disiplin penanganan exception dan manajemen resource, bukan format filenya.

6.7 Menjalankan Program Anda

Tidak ada yang berubah dari prosesnya: python3 library_demo.py. library_demo.py mengimpor Book (dari pustaka), Library (dari library), dan BookNotFoundError (dari book_not_found_error) di bagian atas file.

```
$ python3 library_demo.py
```

6.8 OOP dalam Praktik: Profitina

> **Tentang kotak ini**
>
> Seperti di setiap bab, kotak ini menghubungkan materi kuliah dengan Profitina, platform AI-visibility (Generative Engine Optimization) Indonesia yang nyata dan aktif, dibangun oleh penulis -- menjelaskan DI MANA dan MENGAPA sebuah konsep digunakan dalam sistem nyata yang berjalan, tidak pernah logika bisnis atau kode sumber lengkap Profitina, yang tetap rahasia.

Backend Python/FastAPI Profitina memanggil mesin jawaban AI eksternal dan API pihak ketiga secara terus-menerus -- panggilan yang bisa timeout, mengembalikan data yang salah bentuk, atau gagal sepenuhnya karena alasan yang sepenuhnya di luar kendali Profitina sendiri, persis situasi yang dijelaskan Bagian 6.2 sebagai alasan penanganan exception ada. Setiap panggilan semacam itu dibungkus dalam try/except-nya sendiri dengan tipe exception yang spesifik dan bernama -- tidak pernah except: telanjang, persis karena alasan yang diperingatkan kotak jebakan Bagian 6.3 -- sehingga kegagalan sungguhan selalu terlihat dan ditangani secara sengaja, tidak pernah ditelan diam-diam. Dan karena backend Profitina menyimpan data jauh lebih tahan lama daripada file teks biasa bab ini (database sungguhan, bukan file teks -- mata kuliah Databases/FBP membahas mekanisme itu secara langsung), disiplin dasarnya identik: jangan pernah membiarkan resource terbuka lebih lama

dari yang diperlukan (pernyataan with, dipakai di mana pun backend Profitina menyentuh file atau koneksi jaringan), dan jangan pernah membiarkan kegagalan I/O lewat tanpa disadari.

6.9 Ringkasan Bab dan Poin-Poin Utama

- Setiap exception Python diturunkan dari BaseException; penanganan error biasa bekerja dengan Exception, cabang subclass utamanya.
- Berbeda dengan Java, Python tidak punya perbedaan checked/unchecked yang ditegakkan compiler -- tidak ada yang memaksa pemanggil menangkap apa pun atau mendeklarasikan apa yang mungkin di-raise sebuah fungsi; ini adalah filosofi EAFP Python.
- try-except-finally: finally selalu berjalan, baik blok try berhasil, me-raise exception yang tertangkap, atau me-raise yang menjalar tanpa tertangkap; else berjalan hanya ketika blok try berhasil tanpa exception sama sekali.
- Exception kustom adalah pewarisan biasa -- subclass Exception -- dan BookNotFoundError (milik bab ini) mengikuti persis pola itu.
- Selalu tangkap tipe exception yang paling spesifik yang mungkin; jangan pernah membiarkan blok except kosong, dan hindari except: telanjang atau except Exception: yang terlalu luas.
- Pernyataan with otomatis menutup resource context-manager apa pun (file di sini) ketika blok keluar, menggantikan pola manual try-finally-close() lama yang rawan salah.
- Library mendapat remove_book_or_raise, save_to_file, dan load_from_file di bab ini -- Book sendiri sama sekali tidak berubah.

Program yang hanya pernah berjalan pada input sempurna, terhadap file yang selalu ada, belum menjadi program sungguhan -- itu adalah demonstrasi jalur mulus. Penanganan exception dan file I/O adalah yang memungkinkan Pustaka bertahan dari kontak dengan dunia nyata.

6.10 Pertanyaan Review

Pertanyaan-pertanyaan ini untuk belajar mandiri dan diskusi kelas; tidak dinilai.

1. Jelaskan dengan kata-kata Anda sendiri mengapa Python tidak punya padanan "checked exception" Java -- apa artinya ini bagi bagaimana developer Python mempelajari apa yang bisa di-raise sebuah fungsi?
2. Tulis ulang contoh pernyataan with di Bagian 6.5 sebagai pola manual try-finally-close() lama, dan identifikasi satu cara konkret seorang maintainer bisa salah secara halus pada versi manual yang dibuat tidak mungkin oleh pernyataan with.
3. Mengapa Library.save_to_file membiarkan OSError menjalar ke pemanggilnya alih-alih menangkapnya secara internal dan mencetak pesan error?
4. Tambahkan method Library.count_by_author(author) yang mengembalikan berapa banyak buku dalam koleksi yang ditulis oleh penulis itu, dan putuskan (dengan justifikasi satu kalimat) apakah penulis yang tidak ditemukan seharusnya me-raise exception atau cukup mengembalikan 0.
5. Dengan kata-kata Anda sendiri, jelaskan mengapa blok except: telanjang dianggap salah satu kesalahan terburuk yang bisa dibuat program Python, meskipun ia berjalan tanpa error yang terlihat.

Lampiran 6.A — Log Verifikasi Eksekusi

File book_not_found_error.py, pustaka.py, library.py, dan library_demo.py lengkap (Code/Python/Chapter06/, disediakan bersama buku ini) telah dijalankan pada Python 3.13 sebelum bab ini ditulis, untuk memastikan kodenya benar, bukan sekadar "terlihat benar". Output direproduksi verbatim di bawah.

```
$ python3 library_demo.py
Library size after adding 3 books: 3
Caught expected exception: No book found with ISBN 0000000000000
Library size now: 2
Pustaka Library Report (2 book(s))
- Clean Code by Robert C. Martin [on shelf]
- Effective Java by Joshua Bloch [on shelf]
Saved library to library_export.txt
Reloaded library size: 2
Pustaka Library Report (2 book(s))
- Clean Code by Robert C. Martin [on shelf]
- Effective Java by Joshua Bloch [on shelf]
```

Referensi

[1] Python Software Foundation. "Errors and Exceptions." The Python Tutorial. https://docs.python.org/3/tutorial/errors.html (diakses Agustus 2026).

[2] Python Software Foundation. "with statement." The Python Language Reference. https://docs.python.org/3/reference/compound_stmts.html#with (diakses Agustus 2026).

[3] Python Software Foundation. "Reading and Writing Files." The Python Tutorial. https://docs.python.org/3/tutorial/inputoutput.html#reading-and-writing-files (diakses Agustus 2026).

[4] J. Bloch. Effective Java, 3rd ed. Addison-Wesley, 2017, Bab 10 ("Exceptions") -- dirujuk di sini untuk perbandingan lintas-bahasa di Bagian 6.2.

BAB 7

Interface & Kelas Abstrak

Sejauh ini, Pustaka hanya berisi satu jenis benda: Book. Perpustakaan nyata juga meminjamkan DVD, majalah, dan peralatan, dan tidak satu pun dari itu adalah buku. Bab ini memperkenalkan dua alat yang diberikan Python untuk berbagi struktur dan perilaku di antara kelas-kelas yang sungguh berbeda -- abc.ABC, untuk tipe yang berbagi state dan kode sungguhan, dan typing.Protocol, untuk kemampuan yang melintasi hierarki mana pun (Gambar 7.1).

Tujuan Pembelajaran — setelah bab ini Anda akan mampu:

(1) Menjelaskan mengapa subclass ABC dengan @abstractmethod yang belum diimplementasikan tidak bisa diinstansiasi, dan menulis satu dengan method abstrak maupun method konkret. (2) Mendefinisikan typing.Protocol dan menjelaskan bagaimana sebuah kelas memenuhinya secara struktural, tanpa pernah mewarisinya. (3) Menyatakan aturan praktis untuk memilih antara ABC dan Protocol untuk masalah desain tertentu. (4) Merefaktor kelas yang sudah ada agar mewarisi superclass abstrak dan memenuhi Protocol, tanpa merusak perilaku yang sudah ada. (5) Menulis loop yang memanggil method secara polimorfik di dua subtipe konkret yang tak terkait, menjalankannya, dan memverifikasinya sendiri.

7.1 Rekap: Pustaka Sejauh Ini

Bab 1 sampai 6 membangun satu kelas Book yang terus bertambah kemampuannya dan Library yang mengelola koleksinya: atribut dan enkapsulasi lewat property (Bab 2), model tipe dan alur kontrol Python (Bab 3), list dan kisah membangun string (Bab 5), serta penanganan exception dengan persistensi file (Bab 6). Setiap bab itu memperbaiki Book atau Library tanpa pernah menanyakan pertanyaan yang lebih sulit: apa yang terjadi ketika Pustaka perlu menyimpan sesuatu yang sama sekali bukan Book?

Gambar 7.1 — Peta jalan 16 kuliah OOP. Bab ini adalah Kuliah 7: memberi kelas-kelas yang tak terkait sebuah kontrak bersama, tanpa berbagi implementasi.

7.2 Jawaban Python untuk Kelas Abstrak: ABC dan @abstractmethod

Python tidak punya kata kunci kelas abstrak. Kelas dasar ABC milik modul abc, dikombinasikan dengan decorator @abstractmethod, adalah alat pustaka standar yang paling dekat dengan kelas abstrak Java: kelas yang mewarisi ABC dan mendeklarasikan satu atau lebih method @abstractmethod tidak bisa

diinstansiasi langsung, dan Python melempar TypeError saat konstruksi jika subclass konkret lupa meng-override salah satunya. Ini adalah salah satu dari sedikit tempat Python menegakkan kontrak sama sekali -- di tempat lain dalam mata kuliah ini, termasuk penanganan exception (Bab 6), Python memercayai pemanggil.

ABC minimal dengan method abstrak

```
from abc import ABC, abstractmethod

class LibraryItem(ABC):
    def __init__(self, title: str) -> None:
        self._title = title
        self._on_loan = False

    # Method konkret -- setiap subclass mewarisi kode persis ini.
    @property
    def is_on_loan(self) -> bool:
        return self._on_loan

    # Method abstrak -- setiap subclass WAJIB menyediakan versinya sendiri.
    @abstractmethod
    def calculate_fine(self, days_overdue: int) -> float:
        raise NotImplementedError
```

@abstractmethod ditegakkan saat konstruksi, bukan saat definisi kelas

Python dengan senang hati membiarkan Anda MENDEFINISIKAN subclass yang lupa meng-override method abstrak -- TypeError baru muncul persis saat sesuatu mencoba menginstansiasinya. Override yang hilang bisa tersembunyi tak terdeteksi dalam codebase besar sampai baris tepat yang mencoba mengonstruksi subclass yang belum lengkap itu dijalankan.

7.3 Jawaban Python untuk Interface: Protocol dan Typing Struktural

typing.Protocol (PEP 544) mengambil pendekatan STRUKTURAL yang sama sekali berbeda untuk "setiap tipe yang sesuai wajib menyediakan ini." Sebuah kelas memenuhi Protocol hanya dengan memiliki method bernama dan bersignature sama -- tidak diperlukan deklarasi pewarisan jenis apa pun, dan tidak ada yang pernah menulis class Book(Renewable). Ini adalah filosofi duck-typing Python diterapkan secara formal: "jika ia berperilaku me-renew seperti Renewable, ia adalah Renewable."

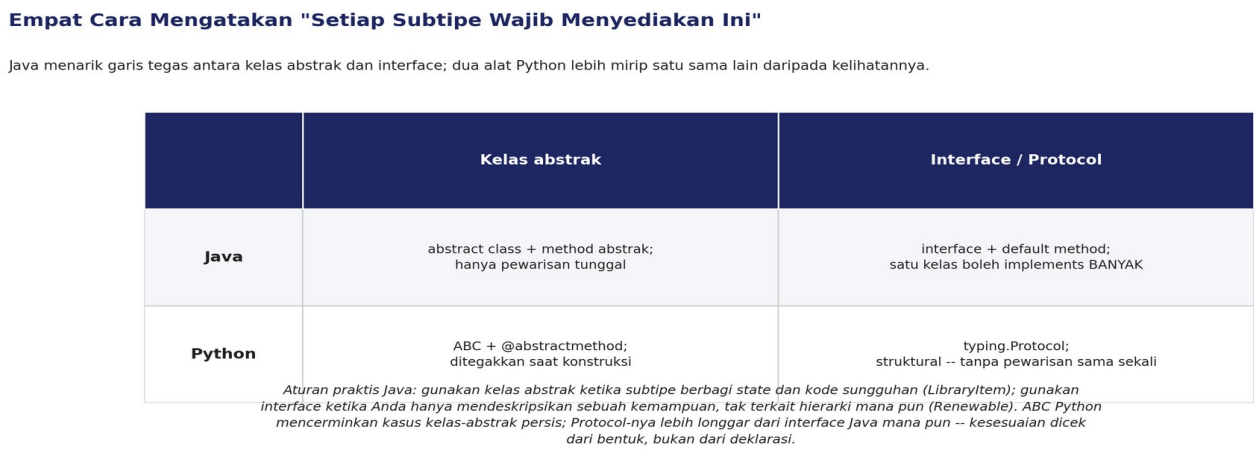
Protocol minimal

```
from typing import Protocol, runtime_checkable

@runtime_checkable
class Renewable(Protocol):
    def renew(self) -> bool: ...
    def renewal_status_label(self) -> str: ...

# Book dan DVD memenuhi Renewable hanya dengan memiliki dua method ini --
# tidak ada satu pun yang menulis `class Book(Renewable)` di mana pun.
```

Decorator @runtime_checkable adalah yang membuat isinstance(some_item, Renewable) sungguh berfungsi saat runtime, hanya memeriksa bahwa nama method yang benar ada -- ia tidak memeriksa

tipe argumen atau tipe kembalian, yang merupakan keterbatasan nyata dibandingkan interface Java yang diperiksa compiler (Gambar 7.2).

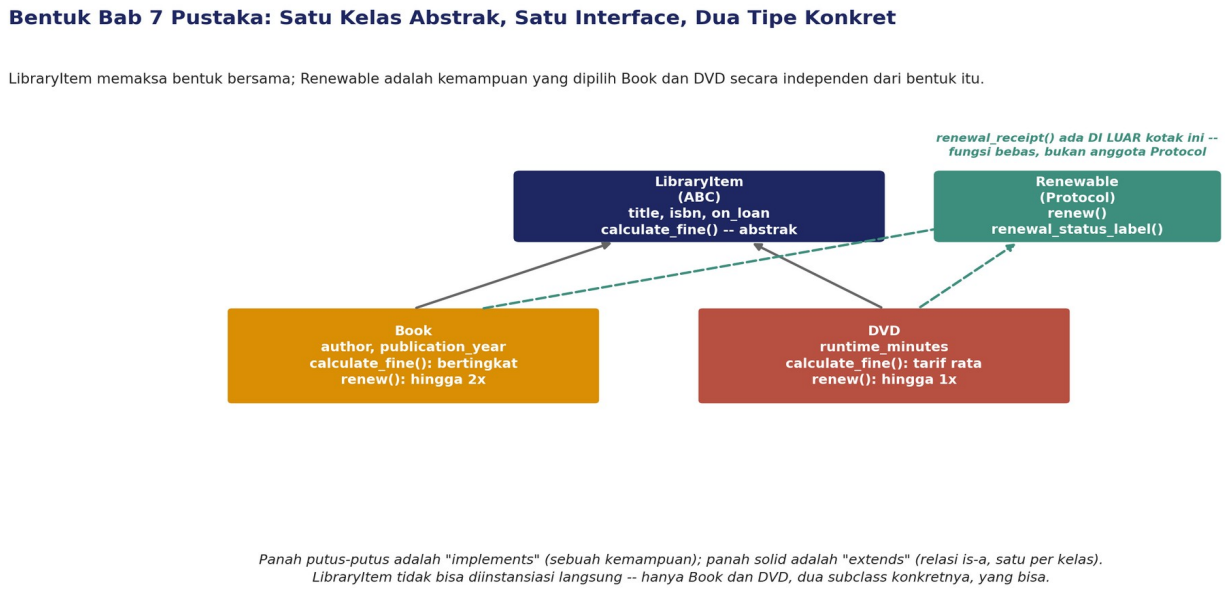
7.4 Membandingkan dengan Java, dan Memilih Antara ABC dan Protocol



Gambar 7.2 — Empat cara mengekspresikan "setiap subtipe wajib menyediakan ini," lintas Java dan Python.

Java menarik garis struktural tegas: kelas abstrak untuk state dan kode bersama dengan pewarisan tunggal, interface untuk kemampuan yang boleh diimplementasikan sebanyak apa pun oleh sebuah kelas, sejak Java 8 secara opsional membawa default method dengan implementasi lengkap yang bisa diwarisi. Dua alat Python memetakan ke perbedaan yang sama itu, tetapi Protocol lebih longgar dari interface Java mana pun -- kesesuaian dicek dari bentuk, bukan dari deklarasi, dan Protocol sendiri tidak bisa membawa implementasi bersama seperti default method Java (Bagian 7.5 menunjukkan solusi umum Python). Aturan praktis: gunakan ABC ketika subtipe berbagi state dan perilaku sungguhan yang non-trivial yang jika tidak akan diduplikasi (state title, isbn, dan status pinjam milik LibraryItem, ditambah logika borrow()-nya yang konkret); gunakan Protocol ketika Anda mendeskripsikan kemampuan yang melintasi hierarki mana pun (Renewable) (Gambar 7.3).

7.5 Memperluas Pustaka: LibraryItem, Renewable, dan Tipe DVD Baru



Gambar 7.3 — Bentuk Bab 7 Pustaka: LibraryItem adalah kelas dasar abstrak; Renewable adalah Protocol yang dipenuhi Book dan DVD secara struktural.

State title, isbn, dan status pinjam milik Book, beserta logika borrow()/return_item()-nya, dipindahkan ke kelas dasar abstrak baru, LibraryItem, yang juga mendeklarasikan calculate_fine() sebagai abstrak -- setiap tipe item konkret wajib menyediakan aturan denda-terlambatnya sendiri, karena kebijakan keterlambatan buku dan kebijakan keterlambatan DVD sungguh berbeda aturannya, bukan sekadar berbeda angkanya. Book hanya menyimpan yang benar-benar khusus book: author, publication_year, dan times_renewed.

Fase 7 — Implement (library_item.py, dipersingkat)

```python
class LibraryItem(ABC):
    def __init__(self, title: str, isbn: str) -> None:
        self._title = title
        self._isbn = isbn
        self._on_loan = False

    def borrow(self) -> bool:
        if self._on_loan:
            return False
        self._on_loan = True
        return True

    @abstractmethod
    def calculate_fine(self, days_overdue: int) -> float:
        raise NotImplementedError

    def describe(self) -> str:
        status = "ON LOAN" if self._on_loan else "on the shelf"
        return f'"{self._title}" [ISBN {self._isbn}] - {status}'
```

DVD adalah tipe item yang sama sekali baru di bab ini -- ia tidak berbagi apa pun dengan Book kecuali state LibraryItem dan bentuk Renewable. Ia bisa diperpanjang paling banyak sekali (Book mengizinkan

dua kali perpanjangan) dan mengenakan tarif harian rata yang lebih tinggi tanpa batas atas (denda Book bertingkat setelah 30 hari).

Fase 7 — Implement (dvd.py, dipersingkat)

```
_MAX_RENEWALS = 1
_DAILY_FINE = 0.75

class DVD(LibraryItem):
    def __init__(self, title: str, isbn: str, runtime_minutes: int) -> None:
        super().__init__(title, isbn)
        self._runtime_minutes = runtime_minutes
        self._times_renewed = 0

    def calculate_fine(self, days_overdue: int) -> float:
        return 0.0 if days_overdue <= 0 else days_overdue * _DAILY_FINE

    def renew(self) -> bool:
        if not self._on_loan or self._times_renewed >= _MAX_RENEWALS:
            return False
        self._times_renewed += 1
        return True

    def renewal_status_label(self) -> str:
        return (
            "No renewals left" if self._times_renewed >= _MAX_RENEWALS
            else "Renewable once"
        )
```

Default method Java vs. fungsi bebas Python

Interface Renewable milik Java bisa membawa default method (renewalReceipt) yang dibangun dari method abstraknya sendiri, diwarisi gratis oleh setiap kelas yang meng-implement. Protocol tidak punya padanan -- ia tidak bisa membawa implementasi bersama, karena kelas yang sesuai tidak pernah sungguh mewarisinya. Jawaban umum Python adalah fungsi tingkat-modul biasa: renewal_receipt(item, attempt_succeeded), menerima objek berbentuk Renewable apa pun sebagai argumen pertamanya.

7.6 Polimorfisme dalam Aksi: Program Driver

Hasil sungguhan bab ini adalah satu loop atas list[LibraryItem] yang berisi Book dan DVD sekaligus, di mana calculate_fine(), renew(), dan describe() semuanya dipanggil secara polimorfik -- loop itu tidak pernah sekali pun bertanya "apakah ini Book atau DVD?" Python secara otomatis mengarahkan setiap panggilan ke override yang benar, berdasarkan tipe runtime objek yang sesungguhnya (Gambar 7.4).

Gambar 7.4 — Dispatch polimorfik: satu badan loop, dua aturan calculate_fine() yang sungguh berbeda.

Fase 7 — Implement (library_demo.py, dipersingkat)

```
items: list[LibraryItem] = [
    Book("Clean Code", "Robert C. Martin", "9780132350884", 2008),
    DVD("The Imitation Game", "9900000000001", 114),
]

for item in items:
    item.borrow()

for item in items:
    print(f"{item.describe()} -> fine: ${item.calculate_fine(10):.2f}")

for item in items:
    if isinstance(item, Renewable):
        ok = item.renew()
        print(f"{item.title}: {renewal_receipt(item, ok)}")
```

Library sendiri sengaja dibiarkan tidak berubah di bab ini -- ia masih mengelola list[Book] persis seperti sejak Bab 5. Menggeneralisasi Library agar bisa menyimpan LibraryItem mana pun adalah tugas Bab 9, setelah pewarisan dan polimorfisme mendapat bab penuh miliknya sendiri; program driver bab ini mendemonstrasikan polimorfisme secara langsung, dalam list mandiri, sehingga idenya tidak tercampur dengan logika persistensi file Library dari Bab 6.

7.7 Menjalankan Program Anda

Tidak ada yang berubah dari prosesnya: python3 library_demo.py. library_demo.py mengimpor LibraryItem, Book, DVD, Renewable, dan renewal_receipt di bagian atas file; tidak ada paket pihak ketiga yang dibutuhkan di luar modul pustaka standar abc dan typing.

```
$ python3 library_demo.py
```

7.8 OOP dalam Praktik: Profitina

> **Tentang kotak ini**
>
> Seperti di setiap bab, kotak ini menghubungkan materi kuliah dengan Profitina, platform AI-visibility (Generative Engine Optimization) Indonesia yang nyata dan aktif, dibangun oleh penulis -- menjelaskan DI MANA dan MENGAPA sebuah konsep digunakan dalam sistem nyata yang berjalan, tidak pernah logika bisnis atau kode sumber lengkap Profitina, yang tetap rahasia.

Backend Python/FastAPI Profitina berbicara dengan beberapa mesin jawaban AI eksternal yang berbeda, masing-masing dengan bentuk API, perilaku timeout, dan format respons sendiri. Alih-alih menulis kode pemanggilan terpisah yang terduplikasi untuk masing-masing, backend mendefinisikan sebuah Protocol bersama (persis pola bab ini -- kontrak struktural, tanpa implementasi bersama yang dibutuhkan) yang dipenuhi setiap konektor spesifik-provider, sehingga bagian lain sistem bisa memanggil salah satu dari mereka secara polimorfik tanpa peduli sedang berbicara dengan yang mana. Menambahkan provider AI baru nanti berarti menulis satu kelas baru dengan bentuk method yang benar, tanpa menyentuh kode mana pun yang sudah memanggilnya.

7.9 Ringkasan Bab dan Poin-Poin Utama

- Subclass ABC dengan @abstractmethod yang belum diimplementasikan tidak bisa diinstansiasi -- Python menegakkan ini saat konstruksi, salah satu dari sedikit kontrak yang ditegakkannya sama sekali.
- typing.Protocol mendeskripsikan kemampuan secara struktural: sebuah kelas memenuhinya murni dengan memiliki nama dan signature method yang cocok, tanpa deklarasi pewarisan yang dibutuhkan.
- Aturan praktis: ABC untuk subtipe yang berbagi state dan kode sungguhan; Protocol untuk kemampuan yang melintasi hierarki mana pun.
- Kelas abstrak dan interface Java memetakan ke ABC dan Protocol Python, tetapi Protocol lebih longgar dari interface Java mana pun, dan tidak bisa membawa implementasi bersama seperti default method Java.
- LibraryItem (ABC) dan Renewable (Protocol) baru di bab ini; Book direfaktor untuk memakai keduanya, dan DVD adalah tipe konkret baru yang hampir tidak berbagi apa pun dengan Book kecuali dua kontrak ini.
- Satu loop atas list[LibraryItem] memanggil calculate_fine(), renew(), dan describe() secara polimorfik di Book dan DVD -- badan loop tidak pernah memeriksa tipe konkret mana yang sedang dipegangnya.

Hierarki kelas yang hanya pernah menyimpan satu jenis objek belum pernah sungguh diuji -- begitu tipe kedua yang sungguh berbeda muncul, kelas dasar abstrak dan Protocol adalah yang membuat kode lama tetap bekerja tanpa satu baris pun berubah.

7.10 Pertanyaan Review

Pertanyaan-pertanyaan ini untuk belajar mandiri dan diskusi kelas; tidak dinilai.

1. Mengapa Python mengizinkan LibraryItem DIDEFINISIKAN dengan @abstractmethod yang belum diimplementasikan, tetapi menolak membiarkan Anda menginstansiasinya?

2. Tambahkan tipe LibraryItem ketiga, Magazine, yang TIDAK memenuhi Renewable sama sekali (majalah di perpustakaan ini hanya bisa dikembalikan, tidak pernah diperpanjang). Method LibraryItem mana yang tetap perlu diimplementasikan Magazine, dan dua method mana yang benar untuk tidak diikutinya?
3. Tulis ulang renewal_receipt agar ia memanggil item.renew() sendiri alih-alih menerima hasilnya sebagai parameter. Jelaskan satu bug konkret yang muncul dari ini jika pemanggil pernah memanggil renewal_receipt() lebih dari sekali.
4. Dengan kata-kata Anda sendiri, jelaskan mengapa pemeriksaan isinstance() milik @runtime_checkable Protocol lebih lemah daripada pemeriksaan interface Java saat kompilasi -- kesalahan jenis apa yang bisa lolos pemeriksaan Python tetapi akan ditangkap compiler Java?
5. Andaikan list[Book] milik Library diubah hari ini menjadi list[LibraryItem] tanpa menunggu Bab 9. Sebutkan satu method Library yang sudah ada yang kodenya perlu berubah, dan jelaskan mengapa.

Lampiran 7.A — Log Verifikasi Eksekusi

File library_item.py, renewable.py, book.py, dvd.py, dan library_demo.py lengkap (Code/Python/Chapter07/, disediakan bersama buku ini) telah dijalankan pada Python 3.13 sebelum bab ini ditulis, untuk memastikan kodenya benar, bukan sekadar "terlihat benar". Output direproduksi verbatim di bawah.

```
$ python3 library_demo.py
--- Overdue fines after 10 days (calculate_fine) ---
"Clean Code" [ISBN 9780132350884] - ON LOAN -> fine: $6.50
"The Imitation Game" [ISBN 9900000000001] - ON LOAN -> fine: $7.50

--- Renewal attempts (Renewable) ---
Clean Code: Renewal attempt: SUCCESS -- Renewed once
The Imitation Game: Renewal attempt: SUCCESS -- No renewals left

--- Full item details (__str__, type-specific) ---
"Clean Code" by Robert C. Martin [ISBN 9780132350884, 2008] - ON LOAN (renewed
1x)
"The Imitation Game" [ISBN 9900000000001, 114 min] - ON LOAN (renewed 1x)
```

Referensi

[1] Python Software Foundation. "abc — Abstract Base Classes." Python 3 documentation. https://docs.python.org/3/library/abc.html (diakses Agustus 2026).

[2] Python Software Foundation. "typing — Protocol." Python 3 documentation. https://docs.python.org/3/library/typing.html#typing.Protocol (diakses Agustus 2026).

[3] Oracle Corporation. "Abstract Methods and Classes" dan "Default Methods." The Java Tutorials. https://docs.oracle.com/javase/tutorial/java/IandI/ (diakses Agustus 2026, rujukan lintas-bahasa untuk Bagian 7.4).

[4] J. Bloch. Effective Java, 3rd ed. Addison-Wesley, 2017, Bab 20 ("Prefer interfaces to abstract classes").

BAB 8 · BAB LATIHAN

Soal Latihan: UTS — Meninjau Ulang Kuliah 1 Sampai 7

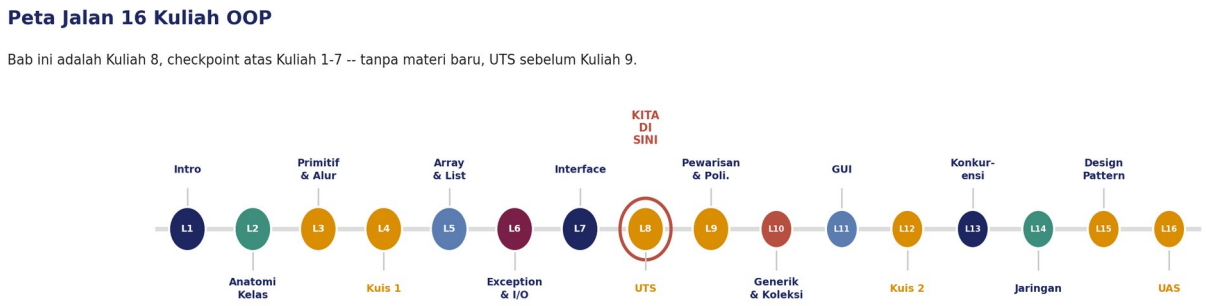
Tidak ada materi baru di sini -- delapan soal yang mencakup semuanya dari enkapsulasi hingga Protocol, dirancang untuk memastikan separuh pertama mata kuliah ini benar-benar melekat, sebelum UTS.

Tujuan Pembelajaran — setelah mengerjakan bab ini Anda akan mampu:

(1) Merancang kelas terenkapsulasi kecil yang menyimpan satu atribut dan menurunkan atribut lain darinya sesuai permintaan. (2) Menulis logika alur kontrol yang berperilaku benar pada setiap nilai batas, bukan hanya kasus yang jelas. (3) Membangun string terformat secara efisien dan memilih subset dari sebuah list sambil menjaga urutan aslinya. (4) Mendefinisikan dan melempar exception kustom, serta membaca dan menulis berkas teks polos dengan aman. (5) Merancang ABC dengan method konkret yang dibangun di atas method abstrak, dan Protocol yang dipenuhi secara struktural, lalu melakukan dispatch atas sebuah list bertipe Protocol itu secara polimorfik.

8.1 Rekap: Kuliah 1–7 Secara Singkat

Tujuh kuliah pertama membangun seluruh perkakas sehari-hari bahasa ini. Kuliah 1 memperkenalkan pergeseran dari pemikiran prosedural ke pemikiran berorientasi objek. Kuliah 2 memberi anatomi presisi pada ide itu -- atribut, konstruktor, dan enkapsulasi. Kuliah 3 melengkapi tipe numerik Python, alur kontrol, string, dan alat debugging dasar. Kuliah 4 adalah Kuis 1, checkpoint pertama atas semua itu. Kuliah 5 memperkenalkan list dan kisah pembangunan string (konkatenasi versus join versus f-string). Kuliah 6 menambahkan penanganan exception dan I/O berkas -- perkakas yang dibutuhkan program begitu ia harus bertahan dari input buruk dan hidup lebih lama dari satu kali eksekusi. Kuliah 7 memperkenalkan abc.ABC dan typing.Protocol, memungkinkan Book milik Pustaka dan tipe DVD yang sama sekali baru berbagi kontrak tanpa berbagi implementasi (Gambar 8.1).



Gambar 8.1 — Bab ini berada di Kuliah 8 dalam peta jalan 16-kuliah OOP: sebuah checkpoint, bukan topik baru, UTS sebelum Kuliah 9.

8.2 Cara Menggunakan Bab Latihan Ini

Apa bab ini -- dan bukan apa

Ini adalah Bab Latihan: bab praktik, bukan bab materi baru. Setiap soal di bawah berakar pada materi spesifik dari Kuliah 1–7 (lihat Gambar 8.2). Setiap soal disertai PETUNJUK -- dorongan ke arah cara berpikir yang tepat -- tapi sengaja BUKAN solusi lengkap atau kode sumber. Kedelapan soal ini berdiri sendiri, independen dari studi kasus Pustaka yang berjalan di seluruh bab biasa -- persis seperti soal-soal Kuis 1 di Bab 4.

Asal Setiap Soal Latihan

Setiap soal di Bagian 8.3 dapat ditelusuri kembali ke salah satu dari tujuh kuliah sebelumnya.

#	Soal	Berasal dari
1	Kelas Suhu	K2 -- atribut, konstruktor, nilai turunan
2	Konversi Nilai	K3 -- alur kendali, kondisi batas
3	Pemformat Daftar	K5 -- pembangunan string, list
4	Top-N Skor	K5 -- list, seleksi yang menjaga urutan
5	Cek Rentang Skor	K6 -- kelas exception kustom
6	File Log Skor	K6 -- I/O berkas, pernyataan with
7	Gaji Karyawan	K7 -- ABC, method konkret di atas method abstrak
8	Item Berdiskon	K7 -- Protocol, typing struktural, dispatch polimorfik

Gambar 8.2 — Setiap soal di Bagian 8.3 dapat ditelusuri kembali ke salah satu dari tujuh kuliah sebelumnya.

Sebelum Anda mulai

Coba setiap soal sungguh-sungguh dalam berkas .py baru sebelum membaca petunjuknya. Jika buntu, baca hanya petunjuknya -- bukan solusi -- lalu coba lagi. Ini persis disiplin yang akan dihargai oleh UTS di Bagian 8.4: ujiannya open-book, tapi itu bukan pengganti penalaran Anda sendiri.

8.3 Soal Latihan

8.3.1 Review Atribut, Konstruktor & Enkapsulasi

Soal 1 *[Mudah]*. Definisikan kelas Temperature(celsius), hanya menyimpan nilai Celsius, dengan get_celsius(), get_fahrenheit() (F = C * 9/5 + 32), dan get_kelvin() (K = C + 273,15).

Petunjuk

Simpan hanya atribut celsius di __init__. Baik get_fahrenheit() maupun get_kelvin() harus menghitung hasilnya sesuai permintaan dari satu atribut itu, bukan menyimpan dua atribut tambahan yang bisa jadi tidak sinkron jika celsius pernah diubah -- persis disiplin yang sama dengan kelas Weight di Bab 4.

8.3.2 Review Tipe Numerik & Alur Kontrol

Soal 2 *[Mudah]*. Tulis fungsi letter_grade(score) yang mengembalikan "A" untuk score >= 85, "B" untuk score >= 70, "C" untuk score >= 55, "D" untuk score >= 40, dan "E" jika tidak. Uji pada 85,0, 84,99, 70,0, 69,99, 55,0, 54,99, 40,0, dan 39,99.

> **Petunjuk**
>
> Urutkan perbandingan Anda dari batas tertinggi ke terendah dengan elif, dan kembalikan begitu satu cocok. Kasus 84,99/69,99/54,99/39,99 sengaja ada untuk menangkap kesalahan off-by-one antara >= dan > yang tidak akan pernah terungkap hanya dengan angka bulat.

8.3.3 List & Pembangunan String

Soal 3 *[Sedang]*. Tulis fungsi format_roster(names) yang menggabungkan nama dengan ", " kecuali dua nama terakhir digabung dengan " and " -- mis. ["Ann", "Budi", "Citra"] menjadi "Ann, Budi, and Citra". Tangani dengan benar kasus 0, 1, 2, dan 3-atau-lebih nama, dan bangun hasilnya secara idiomatis, bukan dengan konkatenasi string berulang dalam loop.

> **Petunjuk**
>
> Tangani kasus 0/1/2-nama sebagai cabang short-circuit tersendiri lebih dulu -- keduanya tidak butuh logika penggabungan sama sekali. Untuk 3 atau lebih, ", ".join(names[:-2]) menangani semuanya kecuali dua terakhir, lalu tambahkan ", " + names[-2] + " and " + names[-1] -- tapi perhatikan penempatan koma agar 3 nama tidak mendapat ", " yang nyasar di depan.

Soal 4 *[Sedang]*. Tulis fungsi top_n(scores, n) yang mengembalikan n nilai terbesar di scores, tetapi dalam urutan relatif ASLI dari list input, bukan diurutkan berdasarkan nilai. Contoh: scores = [70, 95, 60, 88, 99], n = 3 -> [95, 88, 99] (tiga nilai terbesar, 95/88/99, dipertahankan dalam urutan kemunculan aslinya).

> **Petunjuk**
>
> Jangan urutkan list yang Anda kembalikan -- urutkan SALINANNYA (sorted(scores, reverse=True)) untuk menemukan nilai pada posisi cutoff (nilai terbesar ke-n), lalu telusuri list ASLI dalam urutan aslinya dengan list comprehension dan simpan setiap elemen yang melewati cutoff itu, berhati-hatilah dengan nilai duplikat agar Anda tidak menyimpan lebih dari n elemen ketika beberapa nilai sama dengan cutoff.

8.3.4 Penanganan Exception & I/O Berkas

Soal 5 *[Sedang]*. Definisikan kelas exception kustom InvalidScoreError(Exception), dan fungsi validate_score(score) yang mengembalikan score tanpa perubahan jika berada antara 0 dan 100 inklusif, dan selain itu melempar InvalidScoreError yang pesannya menyertakan nilai yang tidak valid.

> **Petunjuk**
>
> InvalidScoreError tidak butuh isi apa pun selain `pass` -- mewarisi dari Exception, persis seperti BookNotFoundError di Bab 6. validate_score() hanya butuh satu kondisi if yang mencakup kedua arah di luar rentang sekaligus, lalu satu pernyataan raise dengan pesan f-string.

Soal 6 ***[Sedang]***. Tulis dua fungsi: append_score_record(path, name, score), yang menambahkan satu baris "name,score" ke berkas di path, dan read_score_records(path), yang membaca kembali setiap baris dan mengembalikan list tuple (name, score), dengan score dikonversi kembali ke int.

> **Petunjuk**
>
> Gunakan with open(path, "a") as f: untuk fungsi pertama (mode append -- JANGAN gunakan "w", yang akan menghapus baris-baris berkas yang sudah ada), dan with open(path) as f: untuk yang kedua, memisah setiap baris pada koma dan membuang newline di akhir sebelum mengonversi score kembali ke int -- persis pola yang dipakai save_to_file/load_from_file di Bab 6 untuk Pustaka.

8.3.5 ABC & Protocol

Soal 7 ***[Sulit]***. Definisikan kelas abstrak Employee(ABC) dengan atribut name, method abstrak monthly_pay(self), dan method KONKRET annual_pay(self) yang mengembalikan self.monthly_pay() * 12. Lalu definisikan dua subclass konkret: SalariedEmployee (gaji bulanan tetap) dan CommissionEmployee (jumlah bulanan dasar ditambah tarif komisi dikalikan angka penjualan bulanan).

> **Petunjuk**
>
> annual_pay() sebaiknya berada di kelas abstrak itu sendiri, bukan di salah satu subclass -- ia sudah sepenuhnya bisa diimplementasikan dalam istilah monthly_pay() apa pun yang akhirnya disediakan tiap subclass, persis seperti LibraryItem.describe() di Bab 7 yang merupakan method konkret dibangun di atas calculate_fine() yang abstrak.

Soal 8 ***[Sulit]***. Definisikan Protocol @runtime_checkable bernama Discountable dengan method discounted_price(self) -> float. Lalu definisikan ClearanceItem(original_price) (discounted_price() = original_price * 0,5) dan MemberItem(original_price, member_discount_rate) (discounted_price() = original_price * (1 - member_discount_rate)), tak satu pun pernah menulis class ClearanceItem(Discountable) di mana pun. Tulis juga fungsi bebas describe_discount(item) yang memformat discounted_price() menjadi string. Terakhir, jumlahkan discounted_price() atas sebuah list[Discountable] yang berisi campuran kedua tipe.

> **Petunjuk**
>
> Tidak satu pun kelas mendeklarasikan pewarisan dari Discountable sama sekali -- keduanya memenuhinya murni dengan memiliki method discounted_price() yang cocok, persis seperti Book dan DVD di Bab 7 yang memenuhi Renewable. describe_discount() harus berada DI LUAR kedua kelas sebagai fungsinya sendiri, karena Protocol tidak bisa membawa implementasi bersama seperti default method interface Java -- alasan yang sama mengapa renewal_receipt() di Bab 7 adalah fungsi bebas, bukan anggota Protocol.

8.4 Format UTS: Open-Book, Individu, Take-Home

UTS adalah penilaian take-home open-book dan individu yang mencakup persis delapan jenis soal yang direview di bab ini -- tidak lebih. Penilaian dilakukan per-soal, bukan semua-atau-tidak-sama-sekali: nilai parsial diberikan untuk kode yang berjalan dan menangani kasus umum dengan benar sekalipun satu edge case terlewat.

Open-book berarti referensi, bukan kolaborator

Anda boleh berkonsultasi dengan buku teks, catatan Anda sendiri, dan slide kuliah saat menjawab. Anda TIDAK BOLEH berkolaborasi dengan teman sekelas atau mengumpulkan kode yang bukan benar-benar milik Anda sendiri, dan setiap pengumpulan harus menyertakan Ikrar Integritas Akademik yang ditandatangani -- ujian take-home open-book menguji apakah Anda bisa MENERAPKAN apa yang sudah Anda pelajari, tanpa pengawasan.

Kriteria	Yang dinilai	Bobot
Kebenaran output	Apakah fungsi atau kelas menghasilkan hasil yang persis sesuai harapan untuk setiap kasus uji yang diberikan, termasuk nilai batas?	45%
Desain kelas & Protocol	Apakah atribut hanya dijangkau lewat method, dan pemisahan ABC/Protocol (Soal 7-8) digunakan dengan benar?	25%
Keamanan exception & I/O	Apakah exception Soal 5 membawa pesan yang berguna, dan apakah I/O berkas Soal 6 menggunakan pernyataan with dengan benar?	15%
Kejelasan kode & berjalan bersih	Apakah kode mudah dibaca, penamaannya masuk akal, dan bebas dari kode mati -- serta berjalan tanpa error?	15%

8.5 OOP dalam Praktik: Profitina

Tentang kotak ini

Seperti di setiap bab, kotak ini menghubungkan materi kuliah dengan Profitina, platform AI-visibility (Generative Engine Optimization) Indonesia yang nyata dan aktif, dibangun oleh penulis -- menjelaskan DI MANA dan MENGAPA sebuah konsep digunakan dalam sistem nyata yang berjalan, tidak pernah logika bisnis atau kode sumber lengkap Profitina, yang tetap rahasia.

Kedelapan soal bab ini, digabungkan, mendekati versi mini dari apa yang sungguh dibutuhkan satu fitur di backend Python/FastAPI Profitina: satu atau dua kelas terenkapsulasi kecil (Soal 1), logika yang diuji-batas dengan cermat (Soal 2), penanganan teks dan list yang efisien untuk sebuah laporan atau respons API (Soal 3-4), penanganan aman untuk input buruk dan state yang persisten (Soal 5-6), dan, ketika beberapa implementasi dari kemampuan yang sama perlu ditukar masuk-keluar -- seperti yang dilakukan adapter mesin-jawaban-AI milik Profitina -- sebuah ABC atau Protocol yang membiarkan bagian lain sistem tetap tidak peduli tipe konkret mana yang sedang dipegangnya (Soal 7-8). UTS yang hanya menguji satu dari kemampuan ini akan melewatkan betapa seringnya fitur nyata membutuhkan beberapa di antaranya bekerja bersama dalam satu potongan kode kecil yang sama.

8.6 Ringkasan Bab

- Bab ini tidak memperkenalkan materi baru -- ini adalah checkpoint yang mencakup Kuliah 1 sampai 7.
- Delapan soal berdiri sendiri mencakup seluruh rentang yang direview: enkapsulasi (K2), batas alur kontrol (K3), pembangunan string dan list (K5), exception kustom dan I/O berkas (K6), serta ABC dan Protocol (K7).
- Setiap soal menyertakan petunjuk, bukan solusi -- mengerjakan penalaran dan kodenya sendiri adalah intinya.
- UTS adalah penilaian take-home open-book, individu: referensi diperbolehkan, tapi kodenya harus milik Anda sendiri, kebenaran pada setiap kasus uji yang diberikan (termasuk nilai batas) adalah porsi terbesar nilai, dan Ikrar Integritas Akademik yang ditandatangani wajib disertakan pada setiap pengumpulan.

Kode yang berjalan tidak sama dengan kode yang benar -- dan kode yang benar untuk angka bulat tidak sama dengan yang benar untuk nilai yang persis berada di batas.

Lampiran 8.A — Checklist Self-Check (Tidak Dinilai)

Sebelum mengumpulkan jawaban apa pun -- pada soal bab ini atau pada UTS itu sendiri -- jalankan lewat checklist ini. Butuh waktu beberapa menit dan menangkap sebagian besar kesalahan yang paling sering dilihat penilai.

- Apakah setiap fungsi dan kelas berjalan tanpa error pada setiap kasus uji yang diberikan?
- Apakah atribut hanya dijangkau lewat method, tanpa kode di luar kelas yang menjangkau langsung?
- Sudahkah Anda menguji nilai batas persis Soal 2 (84,99, 69,99, ...), bukan hanya angka bulat yang bisa menyembunyikan bug >= versus >?
- Apakah top_n() pada Soal 4 mengembalikan nilai dalam urutan ASLI-nya, bukan diurutkan, dan tidak pernah lebih dari n elemen ketika beberapa nilai sama pada cutoff?
- Apakah pesan exception Soal 5 benar-benar menyertakan nilai yang tidak valid, dan apakah I/O berkas Soal 6 menggunakan pernyataan with alih-alih open()/close() manual?
- Apakah method abstrak dan method konkret Soal 7 berada pada kelas abstrak yang SAMA, dengan method konkret memanggil yang abstrak alih-alih menduplikasi logika di setiap subclass?
- Sudahkah Anda membaca ulang kode Anda sendiri seolah-olah Anda penilai skeptis yang mencari satu input yang bisa merusaknya?

Referensi

[1] Format penilaian bab latihan ini (open-book, individu, take-home, dengan Ikrar Integritas Akademik yang ditandatangani) dimodelkan dari UTS nyata mata kuliah ini (EF234302 Pemrograman Berorientasi Objek). Delapan soal latihannya dirancang baru agar sesuai dengan cakupan Kuliah 1-7 yang sesungguhnya diajarkan pada pembaruan mata kuliah ini -- soal-soal ujian asli berpusat pada algoritma tree dan linked-list rekursif, materi yang dalam pembaruan ini sengaja ditempatkan pada mata kuliah Struktur Data, bukan OOP (lihat catatan cakupan Bab 1), sehingga menggunakan ulang soal itu di sini akan menguji materi yang tidak pernah diajarkan mata kuliah ini.

[2] Python Software Foundation. "abc — Abstract Base Classes." Python 3 documentation. https://docs.python.org/3/library/abc.html (diakses Agustus 2026).

[3] Python Software Foundation. "typing — Protocol." Python 3 documentation. https://docs.python.org/3/library/typing.html#typing.Protocol (diakses Agustus 2026).

BAB 9

Pewarisan & Polimorfisme

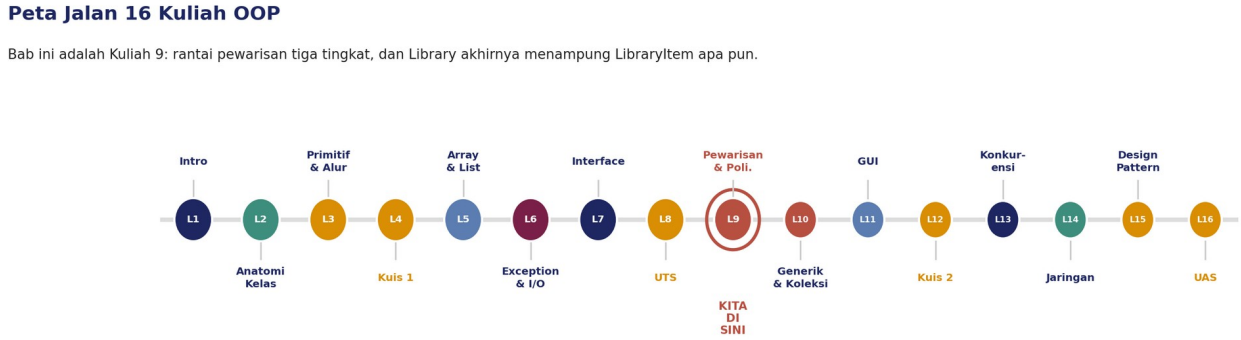
Bab 7 memperkenalkan LibraryItem, Renewable, dan tipe DVD yang baru, tetapi sengaja meninggalkan dua pertanyaan tanpa jawaban: seberapa dalam rantai pewarisan bisa sungguhan pergi, dan kapan Library sendiri berhenti bersifat khusus-Book? Bab ini menjawab keduanya -- menambahkan anak tangga ketiga ke hierarki, memformalkan apa arti polimorfisme sesungguhnya di level runtime, dan akhirnya menggeneralisasi Library agar menampung LibraryItem apa pun.

Tujuan Pembelajaran — setelah bab ini Anda akan mampu:

(1) Menjelaskan perbedaan antara tipe statis (anotasi) sebuah objek dan tipe runtime-nya, serta mengapa pemanggilan method melakukan dispatch berdasarkan yang terakhir. (2) Menulis subclass yang extends kelas yang sudah konkret (bukan hanya ABC), dan memutuskan dengan benar apakah sebuah override sebaiknya memanggil super() atau menggantikan perilaku warisan sepenuhnya. (3) Meng-override __eq__ dan __hash__ bersama-sama, dengan benar, dan menjelaskan mengapa Python memperlakukan override hanya salah satunya sebagai bug yang cukup serius untuk membuat kelas tidak bisa di-hash. (4) Menjelaskan mengapa merekonstruksi koleksi polimorfik dari berkas datar membutuhkan tag tipe eksplisit, meskipun koleksinya sendiri bekerja secara polimorfik setelah dibangun. (5) Menggeneralisasi kelas koleksi yang khusus-tipe agar menampung supertipe bersama, memperbarui setiap method yang terpengaruh tanpa mengubah perilaku publiknya bagi pemanggil yang sudah ada.

9.1 Rekap: Pustaka Sejauh Ini

Bab 7 memberi Pustaka hierarki sungguhannya yang pertama: LibraryItem, ABC yang menyimpan title, isbn, dan on_loan plus calculate_fine() abstrak yang wajib disediakan setiap subtipe konkret; Renewable, Protocol untuk kemampuan perpanjangan pinjaman; dan DVD, tipe konkret baru yang tidak berbagi apa pun dengan Book kecuali kedua kontrak itu. Bab 8 adalah checkpoint, bukan materi baru. Library, sementara itu, diam-diam tetap seperti yang ditinggalkan Bab 5 -- sebuah list[Book], tidak mampu menampung DVD yang baru diperkenalkan Bab 7. Bab ini adalah saat itu akhirnya berubah.



Gambar 9.1 — Peta jalan 16-kuliah OOP. Bab ini adalah Kuliah 9: rantai pewarisan tiga tingkat, dan Library akhirnya menampung LibraryItem apa pun.

9.2 Pewarisan, Secara Formal: Subclassing, super(), dan Kontrak Override

Kelas yang mewarisi dari kelas lain mendeklarasikan relasi is-a: setiap Book is-a LibraryItem, dan (sejak bab ini) setiap ReferenceBook is-a Book, yang membuatnya secara transitif juga LibraryItem. __init__ milik subclass biasanya memanggil super().__init__(...) sebagai langkah pertama, meneruskan argumen ke atas rantai -- Python tidak mewajibkan ini seperti Java yang secara otomatis menyisipkan pemanggilan super() tanpa argumen setiap kali sebuah konstruktor tidak menuliskannya sendiri, tetapi melewatkannya biasanya berarti atribut milik induk tidak pernah disiapkan. Di dalam method yang meng-override, super().nama_method(...) memanggil versi spesifik yang didefinisikan superclass, persis seperti Book.borrow() (Bab 7) menggunakan kembali pemeriksaan "sudah dipinjam?" milik LibraryItem sebelum menambahkan langkah khusus book-nya (Gambar 9.1).

Book.borrow() -- override yang memanggil super() (Bab 7, tidak berubah)

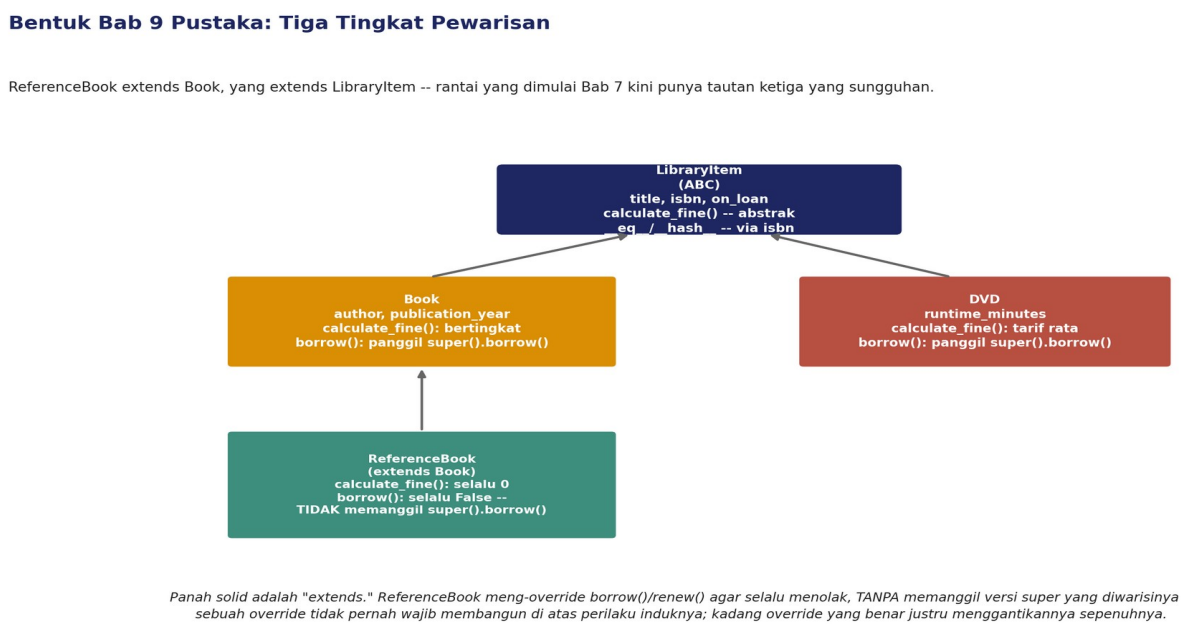
```
def borrow(self) -> bool:
    if not super().borrow():
        return False
    self._times_renewed = 0
    return True
```

Meng-override tidak mengharuskan memanggil super()

Tidak ada apa pun pada override method di Python yang memaksa sebuah method memanggil versi induknya -- memanggil super() adalah pilihan yang dibuat override ketika ingin memperluas, bukan menggantikan, perilaku warisan. Bagian 9.3 menunjukkan pilihan sebaliknya pada hierarki yang sama persis.

9.3 Anak Tangga Ketiga: ReferenceBook dan Pewarisan Multi-Tingkat

ReferenceBook extends Book secara langsung, bukan LibraryItem -- menjadikan LibraryItem → Book → ReferenceBook rantai tiga tingkat yang sungguhan, pertama kalinya studi kasus mata kuliah ini melewati satu tingkat. ReferenceBook mewarisi setiap atribut dan method milik Book, termasuk yang tidak akan pernah dipakainya (_times_renewed, kata-kata bertingkat renewal_status_label()), dan meng-override tepat empat method untuk menegakkan satu aturan: item referensi tidak pernah meninggalkan gedung (Gambar 9.2).



Gambar 9.2 — Bentuk Bab 9 Pustaka: LibraryItem → Book → ReferenceBook, tautan ketiga yang sungguhan dalam rantai yang dimulai Bab 7.

reference_book.py (dipersingkat -- keempat override ditampilkan)

```
class ReferenceBook(Book):
    def __init__(self, title, author, isbn, publication_year=0):
        super().__init__(title, author, isbn, publication_year)

    # Sengaja TIDAK memanggil super().borrow() -- menggantikannya sepenuhnya.
    def borrow(self) -> bool:
        return False

    def renew(self) -> bool:
        return False

    def renewal_status_label(self) -> str:
        return "Hanya referensi -- tidak dapat diperpanjang"

    def calculate_fine(self, days_overdue: int) -> float:
        return 0.0
```

Atribut ber-underscore-tunggal menjangkau setiap tingkat, bukan cuma satu

self._title dan self._isbn diset oleh LibraryItem.__init__, dua tingkat di atas ReferenceBook -- atribut "protected by convention" milik Python diwarisi secara transitif seperti atribut lainnya, sehingga ReferenceBook.__str__ bisa membaca self.title dan self.author (via property milik Book) secara langsung, tanpa LibraryItem perlu memberikan akses itu kedua kalinya.

9.4 Polimorfisme, Secara Formal: Tipe Statis vs. Tipe Runtime

Parameter yang dianotasi item: LibraryItem membawa anotasi itu hanya demi keterbacaan dan untuk alat seperti mypy -- Python sendiri tidak pernah memeriksanya saat runtime. Yang penting ketika item.calculate_fine(10) dieksekusi adalah tipe runtime item yang SESUNGGUHNYA: Python mencari calculate_fine pada kelas apa pun yang sungguh membangun objek itu dan memanggil versi tersebut,

setiap saat. Inilah dynamic dispatch, dan inilah mekanisme di balik setiap loop polimorfik yang ditulis mata kuliah ini sejak Bab 7 -- bisa dibilang lebih sentral bagi Python daripada bagi Java, karena Python tidak pernah memeriksa anotasinya sama sekali.

Mengoper Book atau DVD ke mana pun LibraryItem diharapkan (upcasting, dalam istilah Java) tidak butuh sintaks khusus apa pun di Python -- tidak ada cast yang perlu dituliskan. Menyempitkan ke arah sebaliknya (memperlakukan LibraryItem yang Anda duga sebenarnya DVD sebagai DVD, untuk menjangkau atribut khusus-DVD seperti runtime_minutes) menggunakan isinstance(), yang memeriksa lebih dulu dan hanya melanjutkan jika dugaannya benar -- Python tidak punya padanan downcast Java yang diperiksa yang bisa melempar tepat di titik cast itu sendiri; salah menebak tipe malah memunculkan AttributeError, tepat di titik runtime_minutes sungguh diakses.

Mengoper subtipe tidak butuh sintaks; menyempitkan pakai isinstance()

```
item: LibraryItem = DVD("The Imitation Game", "9900000000001", 114)  # tidak
butuh cast

if isinstance(item, DVD):        # sempitkan, diperiksa lebih dulu
    print(f"{item.runtime_minutes} minutes")
```

9.5 Setiap Objek Diam-Diam Mewarisi dari object: __eq__ dan __hash__

Setiap kelas di Python -- entah dinyatakan atau tidak -- pada akhirnya mewarisi dari object bawaan, dan mewarisi __eq__ dan __hash__ default milik object: perbandingan berbasis identitas, setara dengan is dan dengan id(). Dua objek Book yang dibangun terpisah dari title, author, dan ISBN yang persis sama TIDAK == di bawah default itu, yang jarang menjadi hal yang sungguh ingin diketahui program nyata. LibraryItem meng-override keduanya di bab ini agar membandingkan via isbn -- mengubah "apakah ini objek yang sama?" menjadi "apakah ini item dunia-nyata yang sama?"

Kesetaraan Identitas vs. Kesetaraan Nilai, Java dan Python Berdampingan

Setiap kelas di kedua bahasa mewarisi default berbasis identitas -- LibraryItem meng-override-nya di kedua jalur agar membandingkan via ISBN.

	Default warisan	Override LibraryItem
Java	Object.equals(): true hanya jika objek yang sama (seperti ==)	equals()/hashCode() via isbn; dibangun di atas Book(...).equals(Book(...))
Python	object.__eq__: true hanya jika objek yang sama (seperti is)	__eq__/__hash__ via isbn; mendefinisikan __eq__ saja membuat kelas TIDAK BISA di-hash -- __hash__ juga harus didefinisikan, atau diam-diam menjadi None

Kedua bahasa menegakkan aturan yang sama untuk alasan yang sama: hash sebuah objek HARUS dibangun dari field yang sama persis dengan yang dibandingkan equals()/__eq__, atau objek yang setara bisa jatuh ke bucket hash berbeda dan diam-diam hilang dari lookup set/dict.

Gambar 9.3 — Kesetaraan identitas vs. kesetaraan nilai, Java dan Python berdampingan.

LibraryItem.__eq__ dan __hash__ (dipersingkat)

```
def __eq__(self, other: object) -> bool:
    if self is other:
        return True
    if not isinstance(other, LibraryItem):
        return NotImplemented
    return self._isbn == other._isbn

def __hash__(self) -> int:
    return hash(self._isbn)
```

Python tidak akan membiarkan Anda meng-override hanya salah satunya, secara diam-diam

Kontraknya sama seperti Java -- jika a == b bernilai True, maka hash(a) harus sama dengan hash(b) -- tetapi Python menegakkan kasus setengah-hilang ini lebih agresif daripada Java: mendefinisikan __eq__ TANPA juga mendefinisikan __hash__ membuat kelas tidak bisa di-hash (Python otomatis menyetel __hash__ menjadi None), karena Python tidak bisa lagi mengasumsikan hash warisan masih sesuai dengan kesetaraan yang baru. (Compiler Java diam saja atas kesalahan yang sama -- lihat Gambar 9.3.)

9.6 Menggeneralisasi Library: Satu Koleksi, LibraryItem Apa Pun

Atribut private milik Library berubah dari list[Book] menjadi list[LibraryItem] di bab ini, dan setiap method yang dibangun di atasnya -- add_item (dulu add_book), remove_item, remove_item_or_raise (dulu remove_book_or_raise, dan kini menaikkan ItemNotFoundError yang telah digeneralisasi di bawah), find_by_title, generate_report -- kini bekerja lintas Book, DVD, dan ReferenceBook secara seragam, tanpa percabangan khusus-tipe pada satu pun di antaranya. Satu tempat kode khusus-tipe tetap sungguh diperlukan adalah persistensi berkas: satu baris teks polos tidak membawa informasi tipe apa pun, sehingga save_to_file/load_from_file membutuhkan tag tipe eksplisit sebagai field pertama tiap baris.

library.py -- add_item dan generate_report yang digeneralisasi (dipersingkat)

```
class Library:
    def __init__(self) -> None:
        self._items: list[LibraryItem] = []

    def add_item(self, item: LibraryItem) -> None:
        self._items.append(item)

    # Polimorfik dengan sengaja: tidak pernah memeriksa tipe konkret yang
dipegangnya.
    def generate_report(self) -> str:
        lines = [f"Pustaka Library Report ({len(self._items)} item(s))"]
        for item in self._items:
            lines.append(f"- {item.describe()}")
        return "\n".join(lines) + "\n"
```

library.py -- format berkas dengan tag tipe (dipersingkat)

```
def _to_line(self, item: LibraryItem) -> str:
    if isinstance(item, ReferenceBook):
        return f"REFBOOK|{item.title}|{item.author}|{item.isbn}"
    elif isinstance(item, Book):
        return f"BOOK|{item.title}|{item.author}|{item.isbn}"
    elif isinstance(item, DVD):
        return f"DVD|{item.title}|{item.isbn}|{item.runtime_minutes}"
    raise TypeError(f"Unknown LibraryItem subtype: {type(item).__name__}")
```

Polimorfisme membantu di memori; rekonstruksi butuh tag

Setelah Book, DVD, dan ReferenceBook ada sebagai objek, memanggil describe() pada ketiganya secara polimorfik tidak membutuhkan informasi tipe sama sekali. Membangun ulang ketiga objek yang sama dari berkas teks datar adalah masalah yang sama sekali berbeda -- format berkas itu sendiri harus membawa tag yang menyatakan konstruktor mana yang dipanggil, karena teks polos tidak punya konsep "kelas" sama sekali.

BookNotFoundError milik Bab 6 diganti nama menjadi ItemNotFoundError di bab ini, dengan alasan yang sama seperti method Library sendiri diganti nama -- nama sebuah exception seharusnya mendeskripsikan apa yang sesungguhnya dilaporkannya, dan "BookNotFoundError" berhenti akurat sejak Library berhenti bersifat khusus-Book.

9.7 Polimorfisme dalam Aksi: Program Driver

Program driver membangun Library yang menampung satu Book, satu DVD, dan satu ReferenceBook, lalu menjalankan setiap ide yang diperkenalkan bab ini secara berurutan: borrow() polimorfik (di mana override ReferenceBook diam-diam menolak), laporan polimorfik, __eq__/__hash__ berbasis nilai (objek Book kedua yang berbagi ISBN dengan yang pertama), narrowing isinstance() untuk satu operasi yang sungguh khusus-DVD, dan round trip save/load penuh lewat format berkas bertag-tipe.

library_demo.py (dipersingkat)

```
library = Library()
library.add_item(Book("Clean Code", "Robert C. Martin", "9780132350884", 2008))
library.add_item(DVD("The Imitation Game", "9900000000001", 114))
library.add_item(ReferenceBook("Encyclopedia of Computer Science", "Ralston &
Reilly", "9780123456789", 2003))

book = library.find_by_title("Clean Code")
dvd = library.find_by_title("The Imitation Game")
ref_book = library.find_by_title("Encyclopedia of Computer Science")

copy_of_same_book = Book("Clean Code (2nd copy)", "Robert C. Martin",
"9780132350884", 2008)
print(book == copy_of_same_book)  # True -- ISBN sama, objek berbeda

for item in (book, dvd, ref_book):
    if isinstance(item, DVD):
        print(f"{item.runtime_minutes} minutes")
```

9.8 Mengkompilasi dan Menjalankan Program Anda

Python tidak punya langkah kompilasi terpisah -- python3 library_demo.py menjalankan library.py, book.py, dvd.py, reference_book.py, item_not_found_error.py, dan library_item.py begitu masing-masing di-import. Tidak ada paket baru yang dibutuhkan selain abc dan typing milik Bab 7.

```
$ python3 library_demo.py
```

9.9 OOP dalam Praktik: Profitina

> **Tentang kotak ini**
>
> Seperti di setiap bab, kotak ini menghubungkan materi kuliah dengan Profitina, platform AI-visibility (Generative Engine Optimization) Indonesia yang nyata dan aktif, dibangun oleh penulis -- menjelaskan DI MANA dan MENGAPA sebuah konsep digunakan dalam sistem nyata yang berjalan, tidak pernah logika bisnis atau kode sumber lengkap Profitina, yang tetap rahasia.

Backend Profitina secara rutin menemukan merek atau entitas dunia-nyata yang sama disebut lintas berbagai sumber berbeda -- URL berbeda, kata-kata persis berbeda -- dan perlu mengenali bahwa dua rekaman merujuk pada hal yang SAMA, bukan memperlakukannya sebagai tidak berkaitan. Itu persis masalah __eq__/__hash__ bab ini dalam skala lebih besar: identitas (apakah ini objek literal yang sama di memori, atau baris database yang sama) adalah pertanyaan yang salah; kesetaraan nilai terhadap sebuah identifier kanonis (nama merek atau domain yang dinormalisasi, memainkan peran yang sama seperti isbn di bab ini) adalah yang benar, dan salah menerapkan __hash__ dengan cara diam-diam yang sama seperti diperingatkan bab ini berarti entri duplikat diam-diam bertahan dari deduplikasi yang seharusnya menangkapnya.

9.10 Ringkasan Bab dan Poin-Poin Utama

- Pemanggilan method melakukan dispatch berdasarkan tipe runtime objek yang sesungguhnya, tidak pernah anotasi tipe -- ini adalah mekanisme lengkap di balik setiap loop polimorfik sejak Bab 7.
- Mengoper subtipe ke mana pun supertipe diharapkan tidak butuh sintaks khusus di Python; menyempitkan kembali ke bawah menggunakan isinstance(), karena dugaannya bisa salah.
- Meng-override method tidak pernah mengharuskan pemanggilan super() -- ReferenceBook.borrow() sengaja menggantikan perilaku warisan LibraryItem alih-alih memperluasnya.
- Pewarisan bisa berantai lebih dari satu tingkat (LibraryItem → Book → ReferenceBook); atribut ber-underscore-tunggal yang diset di puncak tetap terjangkau di dasar.
- Setiap kelas secara implisit mewarisi dari object dan __eq__/__hash__ berbasis identitasnya; meng-override keduanya bersama (tidak pernah hanya satu) mengalihkan kelas ke kesetaraan berbasis nilai, dibangun dari atribut yang sama di kedua method -- Python otomatis menonaktifkan hashing jika hanya __eq__ yang di-override.
- Library kini menampung list[LibraryItem], bukan list[Book] -- setiap method bersifat polimorfik kecuali persistensi berkas, yang membutuhkan tag tipe eksplisit untuk merekonstruksi objek konkret dari teks polos.

Sebuah hierarki hanya sungguh teruji pada hari seseorang memintanya melakukan satu hal yang tidak pernah dirancang untuknya -- dan hari Library akhirnya harus menampung DVD dan ReferenceBook berdampingan dengan Book, tanpa satu pun pemanggilnya menyadari perbedaannya, adalah hari itu.

9.11 Pertanyaan Review

Pertanyaan-pertanyaan ini untuk belajar mandiri dan diskusi kelas; tidak dinilai.

1. ReferenceBook.borrow() mengembalikan False tanpa syarat dan tidak pernah memanggil super().borrow(). Tuliskan ulang agar method itu MEMANGGIL super().borrow() lebih dulu, dan jelaskan dalam satu kalimat mengapa perilaku hasilnya akan salah untuk item khusus-referensi.
2. Andaikan LibraryItem.__eq__ membandingkan title, bukan isbn. Berikan satu pasangan konkret objek Book yang akan menjadi "setara" secara salah di bawah perubahan itu, dan jelaskan mengapa isbn adalah pilihan yang lebih aman.
3. Tambahkan tingkat keempat ke hierarki: RareBook extends ReferenceBook, menambahkan atribut estimated_value. Method ReferenceBook mana, jika ada, yang perlu di-override RareBook untuk mengubah perilaku, versus cukup diwarisi tanpa perubahan?
4. Library._to_line() memeriksa isinstance(item, ReferenceBook) sebelum isinstance(item, Book). Jelaskan apa yang akan salah jika urutan itu dibalik, mengingat setiap ReferenceBook juga sebuah Book.
5. Dengan kata-kata Anda sendiri, jelaskan perbedaan antara AttributeError yang muncul ketika narrowing isinstance() dilewati dan dugaannya ternyata salah, dan pemeriksaan isinstance() yang digunakan di seluruh bab ini -- yang mana yang sungguh bisa gagal saat runtime, dan yang mana yang tidak bisa?

Lampiran 9.A — Log Verifikasi Eksekusi

Berkas lengkap library_item.py, book.py, dvd.py, reference_book.py, renewable.py, item_not_found_error.py, library.py, dan library_demo.py (Code/Python/Chapter09/, disertakan bersama buku ini) dijalankan pada Python 3.13 sebelum bab ini ditulis, untuk memastikan kodenya benar, bukan sekadar "terlihat benar". Output direproduksi verbatim di bawah.

```
$ python3 library_demo.py
--- Borrow attempts (polymorphic dispatch) ---
Clean Code .borrow() -> True
The Imitation Game .borrow() -> True
Encyclopedia .borrow() -> False  (ReferenceBook overrides borrow() to always
refuse -- never calls super().borrow())

--- __eq__/__hash__: value equality, not identity ---
book is copy_of_same_book        -> False  (different objects in memory)
book == copy_of_same_book        -> True  (same ISBN -> same real-world item)
hash(book) == hash(copy)         -> True

--- isinstance() narrowing: the one DVD-only operation ---
Clean Code is not a DVD -- no runtime to report
The Imitation Game runtime: 114 minutes
Encyclopedia of Computer Science is not a DVD -- no runtime to report

--- remove_item_or_raise on a missing ISBN ---
Caught: No library item found with ISBN: 0000000000000
```

Referensi

[1] Python Software Foundation. "Data model — object.__eq__, object.__hash__." Python 3 documentation. https://docs.python.org/3/reference/datamodel.html#object.__hash__ (diakses Agustus 2026).

[2] Oracle Corporation. "Overriding and Hiding Methods" dan "Polymorphism." The Java Tutorials. https://docs.oracle.com/javase/tutorial/java/IandI/override.html dan https://docs.oracle.com/javase/tutorial/java/IandI/polymorphism.html (diakses Agustus 2026) — dikutip untuk perbandingan jalur Java pada Gambar 9.3.

[3] J. Bloch. Effective Java, 3rd ed. Addison-Wesley, 2017, Item 10 ("Obey the general contract when overriding equals") dan Item 11 ("Always override hashCode when you override equals") — disiplin yang sama yang diterapkan bab ini pada __eq__/__hash__ milik Python.

BAB 10

Generik & Kerangka Kerja Koleksi

Bab 9 akhirnya membuat Library mampu menampung LibraryItem apa pun, tetapi meninggalkannya dengan tepat satu pola akses (pemindaian linear berdasarkan title) dan tanpa urutan terjamin ketika campuran Book, DVD, dan ReferenceBook. Bab ini menambahkan indeks dict Python berdampingan dengan list milik Library, kelas generik Pair yang bisa dipakai ulang, fungsi generik berparameter tipe terbatas, urutan alami berbasis __lt__, dan set berisi tipe konkret yang berbeda-beda.

Tujuan Pembelajaran — setelah bab ini Anda akan mampu:

(1) Menjelaskan apa yang direpresentasikan parameter tipe sebuah kelas generik (Pair[A, B]) dalam sintaks PEP 695 milik Python, dan mengapa -- berbeda dari Java -- tidak ada apa pun darinya yang pernah diperiksa oleh interpreter itu sendiri, di tahap mana pun. (2) Membaca dan menulis fungsi generik berparameter tipe terbatas (def total_fines[T: LibraryItem](...)), dan menjelaskan apa yang dibatasi oleh T: LibraryItem dan apa yang tidak. (3) Menambahkan indeks kedua berbasis dict ke kelas koleksi berbasis list yang sudah ada tanpa merusak method mana pun yang sudah ada, menjaga kedua struktur tetap diperbarui bersama pada setiap penyisipan dan penghapusan. (4) Mengimplementasikan __lt__ (dengan @total_ordering) agar sorted() menghasilkan urutan alami tanpa argumen key=, dan menjelaskan dari mana urutan itu diturunkan. (5) Menjelaskan mengapa set bawaan Python sama sekali tidak menjamin urutan, dan mengapa driver demo yang mencetaknya perlu membungkusnya dalam sorted() agar log verifikasi tetap reproducible.

10.1 Rekap: Pustaka Sejauh Ini

Bab 9 menggeneralisasi atribut milik Library dari list[Book] menjadi list[LibraryItem] dan memformalkan tipe statis vs. tipe runtime, tetapi setiap method Library masih melakukan tepat satu hal terhadap list itu: mencarinya secara linear berdasarkan title. Masih belum ada cara mencari item berdasarkan ISBN lebih cepat daripada O(n), tidak ada urutan terjamin ketika sebuah laporan mencampur Book, DVD, dan ReferenceBook, dan tidak ada cara mengetahui, sekilas saja, subtipe konkret mana yang sungguh dimiliki sebuah Library. Bentuk bab ini menjawab ketiganya, menggunakan dua tipe koleksi bawaan Python lainnya yang belum pernah dibutuhkan mata kuliah ini: dict dan set.

Gambar 10.1 — Peta jalan 16-kuliah OOP. Bab ini adalah Kuliah 10: kelas generik, fungsi generik terbatas, dan indeks kedua berbasis dict/set berdampingan dengan list yang sudah dimiliki Library.

10.2 Kelas Generik yang Bisa Dipakai Ulang: Pair[A, B]

Memasangkan sebuah LibraryItem dengan nilai hasil hitungan -- jumlah dendanya, misalnya -- atau sebuah title dengan tahun terbit, adalah bentuk yang berulang di sepanjang program, dan menulis kelas khusus untuk setiap kombinasi tipe berarti menulis empat baris boilerplate yang sama berulang-ulang. Pair[A, B] menulis bentuk itu tepat sekali, menggunakan sintaks parameter-tipe modern (3.12+) milik Python: A dan B adalah parameter tipe, ditentukan pemanggil pada titik penggunaan (Pair[LibraryItem, float], Pair[str, int], ...) -- kelasnya sendiri ditulis tepat sekali, persis seperti Pair<A, B> milik Java (Gambar 10.1).

pair.py (lengkap)

```
class Pair[A, B]:
    def __init__(self, first: A, second: B) -> None:
        self._first = first
        self._second = second

    @property
    def first(self) -> A:
        return self._first

    @property
    def second(self) -> B:
        return self._second

    def __repr__(self) -> str:
        return f"({self._first}, {self._second})"
```

Generik, Dibandingkan: Diperiksa Saat Compile vs. Tidak Pernah Diperiksa

Kedua bahasa membiarkan satu kelas/fungsi bekerja untuk banyak tipe -- tetapi apa yang sungguh terjadi pada informasi tipe saat runtime sangat berbeda.

	Java: List<T>, Pair<A,B>, <T extends X>	Python: list[T], Pair[A, B], T: X
Diperiksa kapan?	Saat COMPILE -- javac menolak pemanggilan yang tidak cocok sebelum pernah dijalankan	TIDAK PERNAH oleh interpreter itu sendiri -- hanya oleh alat opsional (mypy, pyright) yang Anda jalankan terpisah
Apa yang bertahan sampai runtime?	Tidak ada -- type erasure menghapus seluruh info generik dari berkas .class; List<String> dan List<Integer> berbagi SATU objek kelas saat runtime	Tidak ada yang ditegakkan juga, tapi alasannya beda -- CPython tidak pernah membaca anotasi tipe sebagai aturan, hanya sebagai metadata opsional
Hasil praktis	Ketidakcocokan tipe adalah error COMPILE, tertangkap sebelum program pernah berjalan	Ketidakcocokan tipe tidak terlihat sampai menyebabkan bug nyata -- atau sampai static checker dijalankan dan melaporkannya

Tak satu pun bahasa memeriksa generik SAAT PROGRAM BERJALAN -- Java memeriksa sekali, saat compile, lalu membuang informasinya; Python tidak pernah memeriksa sama sekali kecuali alat terpisah diminta. "Generik Java dihapus (erased)" dan "generik Python tidak ditegakkan" adalah klaim yang berbeda.

Gambar 10.2 — Generik dibandingkan: diperiksa sekali saat compile lalu dihapus (Java) vs. tidak pernah diperiksa sama sekali oleh interpreter (Python).

Bukan "erasure" -- sejak awal memang tidak pernah ditegakkan

Compiler Java memeriksa setiap penggunaan Pair<LibraryItem, Double> untuk konsistensi dan BARU KEMUDIAN membuang informasi generiknya (type erasure). Python tidak punya langkah pemeriksaan setara untuk dibuang: CPython tidak pernah membaca A dan B sebagai aturan sama sekali, di tahap mana pun -- keduanya hanya ada sebagai metadata untuk alat eksternal seperti mypy atau pyright, jika salah satunya dijalankan terpisah. Hasil praktisnya serupa (tidak ada satu pun PROGRAM YANG BERJALAN di kedua bahasa yang menegakkan parameter tipenya), tetapi "dihapus setelah diperiksa" dan "tidak pernah diperiksa sejak awal" bukan klaim yang sama, dan Gambar 10.2 sengaja menjaga keduanya tetap terlihat berbeda karena itu.

10.3 Fungsi Generik Berparameter Tipe Terbatas: total_fines[T: LibraryItem]

total_fines() perlu menjumlahkan calculate_fine() lintas sebuah list item, tetapi seharusnya tidak perlu peduli apakah list itu menampung LibraryItem, hanya Book, atau hanya DVD -- ketiganya adalah argumen yang masuk akal, karena yang dilakukannya hanya membaca setiap item dan memanggil method yang dijamin setiap LibraryItem. Sintaks parameter-tipe Python 3.12 membiarkan fungsi mendeklarasikan persis itu: def total_fines[T: LibraryItem](items: list[T], ...) dibaca sebagai "T boleh LibraryItem atau subtipe apa pun darinya," batasnya setelah titik dua, dan tubuh fungsi kemudian bisa menerima list[Book] atau list[LibraryItem] secara setara, tanpa sintaks wildcard apa pun yang dibutuhkan.

library_utils.py (lengkap)

```
from library_item import LibraryItem


def total_fines[T: LibraryItem](items: list[T], days_overdue: int) -> float:
    return sum(item.calculate_fine(days_overdue) for item in items)
```

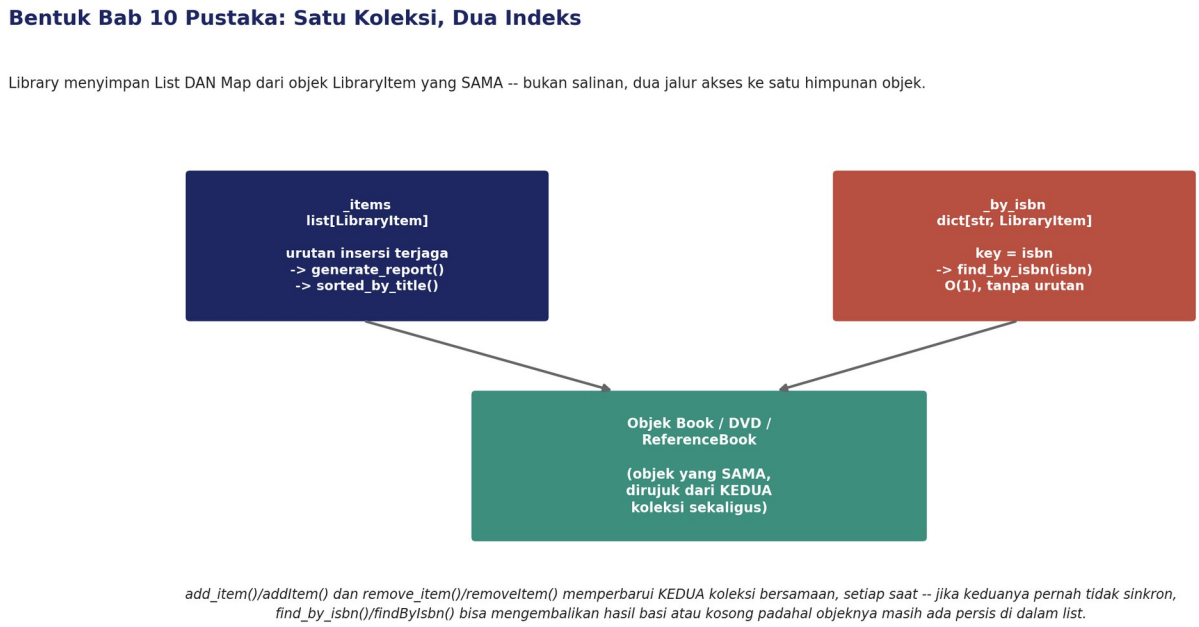
Python tidak butuh wildcard di sini -- List<? extends LibraryItem> Java ada untuk mengakali generik invarian yang di-erasure

Ini ADALAH fungsi generik sungguhan -- ia mendeklarasikan parameter tipenya sendiri, T, dibatasi oleh LibraryItem. Padanan jalur Java (LibraryUtils.totalFines(), Bagian 10.3 buku jalur Java) justru menggunakan WILDCARD terbatas, List<? extends LibraryItem>, bukan parameter tipe -- karena List<Book> milik Java bukan List<LibraryItem> (generiknya invarian di sana) dan wildcard itu ada khusus untuk mengakali pembatasan tersebut. list[Book] Python yang dioper ke tempat list[LibraryItem] dianotasikan sama sekali tidak menimbulkan konflik semacam itu sejak awal, karena tidak ada yang menegakkan anotasi itu saat runtime -- sehingga Python langsung memakai bentuk fungsi-generik yang lebih alami alih-alih membutuhkan jalan-keluar bergaya wildcard.

10.4 Satu Koleksi, Dua Indeks: _by_isbn Berdampingan dengan _items

Atribut _items milik Library (sebuah list[LibraryItem], tidak berubah sejak Bab 9) di bab ini bergabung dengan atribut kedua, _by_isbn, sebuah dict[str, LibraryItem] berkunci ISBN -- padanan Map bawaan

Python, tidak butuh import sama sekali, berbeda dari java.util.HashMap milik Java. add_item dan remove_item kini memperbarui kedua koleksi bersamaan, setiap saat, sehingga keduanya tidak pernah tidak-sinkron. find_by_isbn(isbn) kemudian menjawab dalam O(1) apa yang masih dijawab find_by_title() dalam O(n): keduanya mencari objek LibraryItem yang persis sama, hanya lewat dua jalur berbeda (Gambar 10.3).



Gambar 10.3 — Bentuk Bab 10 Pustaka: indeks list dan indeks dict milik Library, merujuk objek LibraryItem yang sama.

library.py -- indeks _by_isbn (dipersingkat)

```
class Library:
    def __init__(self) -> None:
        self._items: list[LibraryItem] = []
        self._by_isbn: dict[str, LibraryItem] = {}

    def add_item(self, item: LibraryItem) -> None:
        self._items.append(item)
        self._by_isbn[item.isbn] = item

    def remove_item(self, isbn: str) -> bool:
        for i, item in enumerate(self._items):
            if item.isbn == isbn:
                del self._items[i]
                del self._by_isbn[isbn]
                return True
        return False

    def find_by_isbn(self, isbn: str) -> LibraryItem | None:
        return self._by_isbn.get(isbn)
```

Mengapa tetap mempertahankan list, kalau sudah ada dict?

Sebuah dict saja sudah cukup menjawab find_by_isbn() dengan baik, dan (sejak CPython 3.7) bahkan menjaga urutan insersi -- tetapi mengandalkan itu sebagai JAMINAN urutan berarti mengandalkan detail implementasi iterasi, bukan pada apa yang sungguh dibutuhkan generate_report() dan sorted_by_title() bab ini, yaitu titik awal stabil yang kemudian mereka urutkan atau format secara eksplisit. Library mempertahankan kedua struktur karena keduanya melayani pekerjaan yang sungguh berbeda: list adalah sumber kebenaran eksplisit untuk urutan, dict adalah indeks kedua yang dibangun murni demi kecepatan lookup O(1).

10.5 Urutan Alami: @total_ordering dan __lt__

sorted(items) (atau items.sort()) perlu tahu cara mengurutkan dua LibraryItem, dan ia mendapatkannya baik dari argumen key= eksplisit maupun dari tipe elemen itu sendiri yang menyediakan __lt__ -- Bab 10 memilih yang kedua, sehingga setiap list[LibraryItem] dalam program ini memiliki satu urutan default yang jelas (berdasarkan title) tanpa pemanggil mana pun perlu menyediakan key= hanya untuk mengurutkan sebuah list. @total_ordering (dari functools) secara otomatis mengisi __le__, __gt__, dan __ge__ begitu __eq__ (Bab 9) dan __lt__ (bab ini) sama-sama ada.

library_item.py -- __lt__() (baru di bab ini)

```
from functools import total_ordering

@total_ordering
class LibraryItem(ABC):
    # ... title, isbn, __eq__/__hash__ tidak berubah dari Bab 9 ...

    def __lt__(self, other: "LibraryItem") -> bool:
        if not isinstance(other, LibraryItem):
            return NotImplemented
        return self._title.lower() < other._title.lower()
```

library.py -- sorted_by_title()

```
def sorted_by_title(self) -> list[LibraryItem]:
    return sorted(self._items)
```

Urutan alami vs. key= eksplisit

__lt__ menyediakan urutan ALAMI sebuah kelas -- satu urutan default yang diklaim masuk akal oleh kelas itu sendiri, di sini berdasarkan title, tidak peduli huruf besar/kecil. Pemanggil yang menginginkan urutan berbeda (berdasarkan ISBN, jumlah denda, tahun terbit) tidak pernah terjebak pada title: mengoper sorted(items, key=lambda i: i.calculate_fine(10)) menimpa urutan alami untuk satu pemanggilan itu saja, tanpa mengubah __lt__ atau memengaruhi pemanggil lain mana pun.

10.6 set Berisi Tipe-Tipe yang Berbeda -- dan Mengapa Demo Mengurutkannya

distinct_types() menjawab pertanyaan yang tidak bisa dijawab Library dalam satu baris sebelum bab ini: subtipe konkret mana yang sungguh dimiliki Library ini sekarang? Seluruh kontrak sebuah set adalah "tanpa duplikat" -- tiga Book menyumbang string "Book" ke hasil hanya sekali, berapa pun banyaknya objek Book yang sungguh ada -- dan set bawaan Python (tidak butuh import, berbeda dari HashSet/LinkedHashSet milik Java) SAMA SEKALI TIDAK menjanjikan urutan elemennya kembali ketika di-iterasi.

library.py -- distinct_types()

```
def distinct_types(self) -> set[str]:
    return {type(item).__name__ for item in self._items}
```

set Python biasa bukan cuma tidak-terurut -- ia bisa berbeda antar-eksekusi PROGRAM YANG SAMA

Jalur Java menyelesaikan masalah reproduktibilitas yang persis sama ini dengan memilih LinkedHashSet, implementasi Set yang sengaja menjaga urutan pertama-kali-terlihat. set bawaan Python tidak punya varian terurut untuk dipakai sama sekali, dan lebih buruk lagi: karena CPython secara default merandomisasi hashing string per proses (PYTHONHASHSEED), pemanggilan distinct_types() yang persis sama bisa mencetak elemennya dalam urutan berbeda pada dua eksekusi terpisah dari program yang identik -- dikonfirmasi dengan menjalankan satu baris kode yang sama tiga kali dan mengamati dua urutan berbeda. Log Lampiran 10.A perlu menjadi transkrip yang jujur dan REPRODUCIBLE, sehingga library_demo.py membungkus hanya kedua pernyataan print-nya dalam sorted(...) untuk tujuan tampilan -- distinct_types() sendiri tetap mengembalikan set[str] yang sungguh tidak-terurut; tidak ada apa pun pada tipe kembalian atau kontrak sesungguhnya yang berubah.

10.7 Menyatukan Semuanya: Program Driver

Driver membangun Library yang menampung empat item -- trio Bab 9 plus Book keempat, Algorithms karya Sedgewick -- lalu menjalankan lima tambahan bab ini secara berurutan: lookup find_by_isbn() O(1), sorted_by_title() yang langsung menjadi masukan total_fines() generik terbatas, distinct_types() (kutipan Bagian 10.6 sudah menunjukkan yang satu ini, plus pembungkus tampilan sorted() yang dibutuhkannya), dan akhirnya dua instance Pair yang dibangun dari data yang sama. Kutipan dipersingkat di bawah menunjukkan yang pertama, ketiga, dan kelima secara langsung; urutan lengkapnya, plus langkah rekap Bab 9 yang mendahuluinya, ada di Code/Python/Chapter10/library_demo.py.

library_demo.py (dipersingkat)

```
library = Library()
library.add_item(Book("Clean Code", "Robert C. Martin", "9780132350884", 2008))
library.add_item(DVD("The Imitation Game", "9900000000001", 114))
library.add_item(ReferenceBook("Encyclopedia of Computer Science", "Ralston &
Reilly", "9780123456789", 2003))
library.add_item(Book("Algorithms", "Robert Sedgewick", "9780321573513", 2011))

found_by_isbn = library.find_by_isbn("9780321573513")
print(found_by_isbn.title)

by_title = library.sorted_by_title()
all_fines = total_fines(by_title, 10)
print(f"${all_fines:.2f}")

book = library.find_by_title("Clean Code")
most_expensive = Pair(book, book.calculate_fine(10))
print(most_expensive)
title_and_year = Pair("Clean Code", 2008)
print(title_and_year)
```

10.8 Mengkompilasi dan Menjalankan Program Anda

Python tidak punya langkah kompilasi terpisah, tetapi kode bab ini membutuhkan interpreter yang lebih baru daripada bab mana pun sebelumnya: class Pair[A, B]: milik pair.py dan def total_fines[T: LibraryItem](...) milik library_utils.py sama-sama memakai sintaks parameter-tipe PEP 695, diperkenalkan di Python 3.12. Ini adalah bab pertama di mana kebutuhan itu bukan sekadar direkomendasikan, melainkan benar-benar wajib secara sintaksis -- menjalankan library_demo.py pada Python 3.11 melempar SyntaxError tepat pada baris class Pair[A, B]: itu sendiri, sebelum logika bab ini sempat dijalankan sama sekali. Lingkungan mata kuliah ini secara default memakai Python 3.11, sehingga Bab 10 dan seterusnya harus dijalankan secara eksplisit dengan python3.12 atau python3.13.

```
$ python3.13 library_demo.py
```

python3 saja tidak lagi cukup mulai bab ini

Kode setiap bab sebelumnya berjalan baik-baik saja di bawah python3 bawaan lingkungan ini (3.11.15). pair.py dan library_utils.py adalah berkas pertama di mata kuliah ini yang sungguh membutuhkan 3.12+ secara sintaksis -- python3 library_demo.py langsung gagal dengan SyntaxError: invalid syntax pada definisi class/function PEP 695 pertama yang di-import-nya, bukan pada apa pun yang sungguh dilakukan Bab 10 saat runtime. Periksa versi interpreter Anda sendiri dengan python3 --version, dan jika hasilnya 3.11 atau lebih lama, gunakan python3.12 atau python3.13 secara eksplisit, seperti yang dilakukan log bab ini di bawah.

10.9 OOP dalam Praktik: Profitina

> **Tentang kotak ini**
>
> Seperti di setiap bab, kotak ini menghubungkan materi kuliah dengan Profitina, platform AI-visibility (Generative Engine Optimization) Indonesia yang nyata dan aktif, dibangun oleh penulis -- menjelaskan DI MANA dan MENGAPA sebuah konsep digunakan dalam sistem nyata yang berjalan, tidak pernah logika bisnis atau kode sumber lengkap Profitina, yang tetap rahasia.

Backend Profitina menyimpan list terurut berisi sebutan merek yang dilacak untuk laporan peringkatnya, dan, berdampingan dengannya, sebuah dict berkunci identifier eksternal kanonis untuk lookup deduplikasi O(1) ketika sebutan baru datang -- persis bentuk _items/_by_isbn bab ini, dengan alasan yang persis sama: satu struktur menjaga urutan yang dibutuhkan laporan, yang lain menukar urutan demi kecepatan lookup. Utilitas pemasangan generik bersama (Pair bab ini, dalam semangatnya) muncul di mana pun backend perlu membawa skor hasil hitungan berdampingan dengan rekaman tempat skor itu dihitung, tanpa menulis kelas dua-atribut khusus untuk setiap jenis skor yang ditambahkannya.

10.10 Ringkasan Bab dan Poin-Poin Utama

- Parameter tipe generik Python (Pair[A, B], PEP 695) tidak pernah diperiksa oleh interpreter itu sendiri, di tahap mana pun -- berbeda dari Java, tempat compiler memeriksa sekali lalu menghapusnya; hanya alat eksternal seperti mypy yang menegakkannya, jika salah satunya dijalankan sama sekali.
- Fungsi generik berparameter tipe terbatas (def total_fines[T: LibraryItem](...)) tidak butuh jalan-keluar bergaya wildcard seperti padanan Java, karena type hint Python tidak membawa pembatasan invarian saat runtime yang perlu diakali.
- Library kini mempertahankan dict[str, LibraryItem] (_by_isbn) berdampingan dengan list[LibraryItem] (_items), diperbarui bersama pada setiap penyisipan dan penghapusan, menukar jaminan urutan yang tidak bisa diandalkan pada dict demi lookup ISBN O(1).
- Mendefinisikan __lt__ (dengan @total_ordering) memberi kelas urutan alami yang dipakai sorted() tanpa argumen key=; key= eksplisit tetap bisa menimpa default itu untuk satu pemanggilan, tanpa menyentuh __lt__.
- set Python biasa sama sekali tidak menjamin urutan dan bisa berbeda antar-eksekusi program yang identik (randomisasi hash string) -- distinct_types() mempertahankan tipe kembalian set[str] yang jujur, dan hanya kode tampilan demo yang membungkus hasilnya dalam sorted() agar log verifikasi tetap reproducible.

Sebuah kelas koleksi baru layak disebut "framework" pada hari ia berhenti menjadi sekadar list dengan method tambahan yang ditempel -- dan hari Library tumbuh dict yang harus tetap sinkron dengan list-nya, pada setiap penyisipan dan penghapusan, tanpa satu pun boleh tertinggal, adalah hari itu.

10.11 Pertanyaan Review

Pertanyaan-pertanyaan ini untuk belajar mandiri dan diskusi kelas; tidak dinilai.

1. total_fines() ditulis sebagai def total_fines[T: LibraryItem](items: list[T], ...). Jelaskan apa yang akan berubah, jika ada, tentang pemanggilan mana yang lolos pemeriksaan mypy jika batasnya dihapus sama sekali (def total_fines[T](...)).
2. Andaikan Library.remove_item(isbn) lupa memanggil del self._by_isbn[isbn], sehingga hanya menghapus item dari _items. Deskripsikan satu urutan pemanggilan konkret setelah titik itu yang akan mengembalikan jawaban salah (tetapi tidak jelas-jelas salah).
3. Jalankan python3.13 -c "print({'Book', 'DVD', 'ReferenceBook'})" tiga kali terpisah. Jelaskan, dengan kata-kata Anda sendiri, mengapa ketiga urutan yang tercetak tidak dijamin sama, dan mengapa itu adalah sifat hashing string, bukan set itu sendiri yang rusak.
4. __lt__ milik library_item.py mengurutkan berdasarkan title, tidak peduli huruf besar/kecil. Tuliskan pemanggilan sorted(items, key=...) alternatif yang mengurutkan berdasarkan calculate_fine(10) secara menurun, tanpa mengubah __lt__ sama sekali.
5. Jelaskan, dengan kata-kata Anda sendiri, mengapa __repr__ milik pair.py memakai pemformatan bergaya str() (f"({self._first}, {self._second})") alih-alih repr() pada tiap elemen, dan apa yang akan tercetak untuk sebuah Book seandainya memakai pemformatan !r.

Lampiran 10.A — Log Verifikasi Eksekusi

Berkas lengkap library_item.py, renewable.py, book.py, dvd.py, reference_book.py, item_not_found_error.py, pair.py, library_utils.py, library.py, dan library_demo.py (Code/Python/Chapter10/, disertakan bersama buku ini) dijalankan pada Python 3.13 sebelum bab ini ditulis, untuk memastikan kodenya benar, bukan sekadar "terlihat benar". Output direproduksi verbatim di bawah.

```
$ python3.13 library_demo.py
--- Borrow attempts (polymorphic dispatch, Chapter 9) ---
Clean Code .borrow() -> True
The Imitation Game .borrow() -> True
Encyclopedia .borrow() -> False  (ReferenceBook overrides borrow() to always
refuse -- never calls super().borrow())

--- __eq__/__hash__: value equality, not identity (Chapter 9) ---
book == copy_of_same_book        -> True  (same ISBN -> same real-world item)

--- NEW Chapter 10: find_by_isbn(), the dict-backed O(1) lookup ---
find_by_isbn("9780321573513") -> Algorithms

--- NEW Chapter 10: sorted_by_title(), via LibraryItem's __lt__ ---
  Algorithms
  Clean Code
  Encyclopedia of Computer Science
  The Imitation Game

--- NEW Chapter 10: distinct_types(), a set[str] (no duplicates) ---
distinct_types() -> ['Book', 'DVD', 'ReferenceBook']

--- NEW Chapter 10: total_fines(), a generic function with a bounded type
parameter ---
total_fines(all items, 10 days overdue) -> $20.50

--- NEW Chapter 10: Pair[A, B], a generic class ---
Pair[LibraryItem, float] -> ("Clean Code" by Robert C. Martin [ISBN
9780132350884, 2008] - ON LOAN (renewed 0x), 6.5)
Pair[str, int]           -> (Clean Code, 2008)

--- remove_item_or_raise on a missing ISBN (Chapter 6/9) ---
Caught: No library item found with ISBN: 0000000000000

--- Save/load round trip (type-tagged file format, Chapter 6/9) ---
Reloaded 4 item(s), distinct types: ['Book', 'DVD', 'ReferenceBook']
```

Referensi

[1] Python Software Foundation. "Generic classes" dan PEP 695 — "Type Parameter Syntax." Python 3 documentation dan Python Enhancement Proposals. https://docs.python.org/3/reference/compound_stmts.html#generic-classes dan https://peps.python.org/pep-0695/ (diakses Agustus 2026).

[2] Python Software Foundation. "functools.total_ordering." Python 3 documentation. https://docs.python.org/3/library/functools.html#functools.total_ordering (diakses Agustus 2026).

[3] Oracle Corporation. "Generics (Updated)" dan "Bounded Types." The Java Tutorials. https://docs.oracle.com/javase/tutorial/java/generics/index.html dan https://docs.oracle.com/javase/tutorial/java/generics/bounded.html (diakses Agustus 2026) — dikutip untuk perbandingan jalur Java pada Gambar 10.2.

[4] Python Software Foundation. "object.__hash__" dan "PYTHONHASHSEED." Python 3 documentation. https://docs.python.org/3/reference/datamodel.html#object.__hash__ dan https://docs.python.org/3/using/cmdline.html#envvar-PYTHONHASHSEED (diakses Agustus 2026) — dikutip untuk urutan set yang tidak deterministik yang dibahas pada Bagian 10.6.

BAB 11

Pemrograman GUI: Event & MVC

Setiap bab sebelumnya, Pustaka selalu berupa program konsol: sebuah skrip driver yang berjalan dari atas ke bawah sekali, mencetak hasilnya, lalu berhenti. Bab ini memberi Pustaka antarmuka berjendela yang nyata -- daftar item, formulir untuk menambah buku, tombol untuk meminjam/mengembalikan/menghapus -- dibangun dengan pustaka bawaan Python tkinter dan diorganisasi dengan pola Model-View-Controller (MVC), yang memecah program menjadi Model yang tidak tahu apa-apa tentang GUI, View yang tidak tahu apa-apa tentang logika perpustakaan, dan Controller yang menjadi satu-satunya kelas yang boleh menyentuh keduanya (Gambar 11.1).

Tujuan Pembelajaran — setelah bab ini Anda akan mampu:

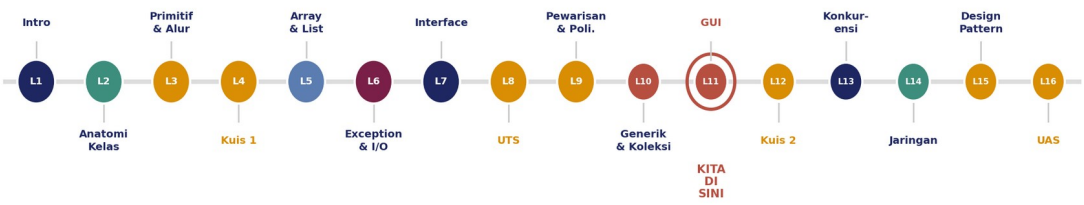
(1) Menjelaskan apa yang dipisahkan oleh pola Model-View-Controller, dan menyatakan kelas mana boleh bergantung pada kelas mana. (2) Membaca dan menulis event handler Tkinter yang dipasang lewat argumen command= sebuah widget, dan menjelaskan kapan Tkinter sungguh memanggilnya. (3) Membangun kelas View Tkinter yang tugasnya semata-mata membangun dan menyediakan widget, tanpa logika bisnis sama sekali. (4) Menelusuri satu klik tombol dari awal sampai akhir: View mengekspos klik -> handler Controller membaca View -> Controller mengubah Model -> Controller menyuruh View menyegarkan tampilannya. (5) Menjelaskan mengapa eksekusi verifikasi terskrip tanpa layar (memilih baris dan memicu tombol secara terprogram) membuktikan jalur kode yang sama persis dengan klik mouse sungguhan, dan bukan simulasi GUI.

11.1 Rekap: Pustaka Sejauh Ini

Bab 10 memberi Library indeks kedua (dict[str, LibraryItem] berkunci ISBN, berdampingan dengan list[LibraryItem]-nya), kelas generik Pair[A, B] yang bisa dipakai ulang (PEP 695), fungsi utilitas generik sungguhan, dan urutan alami lewat __lt__ -- tetapi setiap bab hingga Bab 10 masih berinteraksi dengan Library itu tidak lebih dari sekadar rangkaian pemanggilan print() yang dipilih sekali, saat skripnya ditulis. Tidak ada apa pun di library_demo.py yang membiarkan seseorang memilih item mana yang mau dipinjam selagi program berjalan; driver sudah memutuskan semuanya lebih dulu. Bentuk bab ini menjawab itu: sebuah jendela nyata, tempat seseorang mengklik baris list sungguhan dan tombol sungguhan, dan program merespons apa pun yang diklik orang itu, bukan skrip yang ditulis sebelumnya.

Peta Jalan 16 Kuliah OOP

Bab ini adalah Kuliah 11: membangun GUI berjendela nyata di atas Model yang sudah dibangun mata kuliah ini, lewat pola MVC.



Gambar 11.1 — Peta jalan 16-kuliah OOP. Bab ini adalah Kuliah 11: GUI berjendela nyata dibangun di atas Model Library/LibraryItem yang sama, yang sudah dibangun mata kuliah ini sejak Bab 5, lewat pola Model-View-Controller.

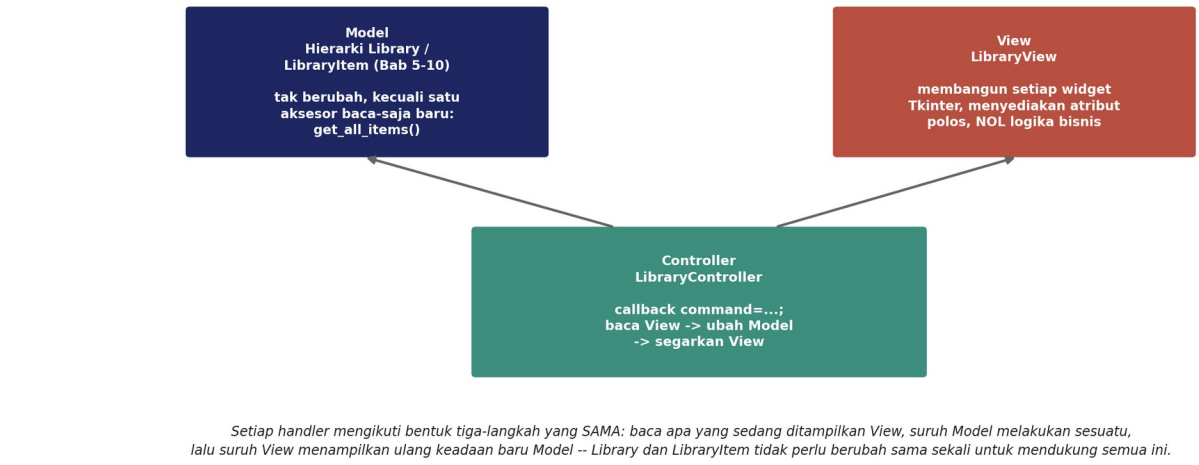
11.2 Dari Konsol ke Jendela: Pemrograman Berbasis Event

Sebuah skrip konsol seperti library_demo.py adalah program yang MEMEGANG KENDALI: ia memutuskan, baris demi baris, apa yang terjadi selanjutnya, dan memanggil input() (jika pernah dilakukannya) hanya akan menjeda urutan tetap yang sama, tidak pernah mengubah bentuknya. Program GUI membalik hubungan itu sepenuhnya -- begitu main() selesai membangun jendela dan memanggil root.mainloop(), kode program itu sendiri berhenti mengendalikan apa pun. Event loop milik Tkinter mengambil alih, duduk diam sampai pengguna melakukan sesuatu (mengklik tombol, mengetik di entry, menutup jendela), lalu memanggil balik tepat satu potongan kecil kode milik programmer untuk setiap event semacam itu. Gaya ini disebut PEMROGRAMAN BERBASIS EVENT: logika program adalah kumpulan callback, masing-masing pendek, masing-masing dipicu oleh sesuatu yang dilakukan pengguna, tanpa fungsi mana pun yang menggambarkan keseluruhan eksekusi dari awal sampai akhir seperti kode tingkat-atas sebuah skrip biasa dulu.

> **root.mainloop() tidak pernah kembali selama eksekusi biasa**
>
> Event loop Tkinter memanggil sebuah callback, menunggunya selesai, lalu kembali menunggu event berikutnya -- ia tidak menjalankan callback Anda bersamaan dengan hal lain. Inilah persis alasan setiap handler di LibraryController bab ini pendek dan langsung selesai: handler yang berjalan lama (misalnya panggilan jaringan yang lambat) akan membekukan seluruh jendela selama durasi itu, karena tidak ada apa pun -- bahkan menggambar ulang jendela -- yang bisa terjadi selagi event loop berada di dalam callback Anda. (Materi konkurensi Bab 13 adalah persis alat untuk handler yang sungguh perlu melakukan pekerjaan lambat tanpa membekukan UI; semua handler bab ini cukup cepat sehingga pertanyaan itu tidak pernah muncul.)

11.3 Pola Model-View-Controller

MVC memecah program GUI menjadi tiga kelas dengan tiga tugas terpisah, dan tepat satu hubungan yang diizinkan di antara mereka: Controller boleh bergantung pada Model maupun View, tetapi Model tidak boleh pernah meng-import atau merujuk apa pun tentang View, dan View tidak boleh pernah memanggil apa pun pada Model secara langsung. Model milik Pustaka -- hierarki Library dan LibraryItem -- ditulis sepanjang Bab 5 hingga 10 tanpa memikirkan GUI sama sekali, dan bab ini hanya perlu menambahkan tepat satu method padanya (get_all_items(), aksesor yang mengembalikan salinan) agar bisa dipakai dari View. Setiap method Model lain yang sudah ditulis Bab 5-10 -- add_item(), remove_item(), find_by_title(), borrow(), return_item() -- dipakai ulang sepenuhnya tanpa perubahan (Gambar 11.2).



Gambar 11.2 — Bentuk Bab 11 Pustaka: Model, View, dan Controller, dengan Controller sebagai satu-satunya kelas yang menyentuh keduanya.

Model-View-Controller, secara presisi

MODEL: data dan aturan tentangnya (Library, LibraryItem, Book, DVD, ReferenceBook) -- tidak tahu apa-apa tentang tombol, jendela, atau piksel. VIEW: widget yang terlihat dan tata letaknya (LibraryView) -- tidak tahu apa-apa tentang apa artinya meminjam atau menghapus sebuah item, hanya tahu cara menampilkan LibraryItem dan menyediakan widgetnya sendiri. CONTROLLER: perekat (LibraryController) -- membaca apa yang sedang ditampilkan View, menyuruh Model melakukan sesuatu, lalu menyuruh View menampilkan ulang keadaan baru Model. Sebuah View yang dibangun terhadap Library yang sama sekali berbeda akan terlihat identik piksel demi piksel; sebuah Model yang dipakai oleh View yang sama sekali berbeda akan berperilaku identik.

11.4 View: LibraryView

LibraryView membangun setiap widget Tkinter yang terlihat di dalam konstruktornya -- sebuah Listbox untuk item, empat Entry untuk menambah buku, empat ttk.Button, dan sebuah Label status -- dan menyediakan masing-masing sebagai atribut publik polos (Python tidak punya padanan field privat yang ditegakkan bahasanya, sehingga View sekadar mengandalkan disiplin konvensi-penamaan yang sama yang sudah dipakai kelas-kelas bab sebelumnya). Ia tidak memanggil apa pun pada Library, dan tidak memiliki method yang memutuskan apa artinya meminjam, menambah, atau menghapus; satu-satunya method substantif selain konstruksi adalah refresh(items), yang sekadar menggambar ulang daftar item apa pun yang diberikan kepadanya.

library_view.py (kutipan)

```
class LibraryView:
    def __init__(self, root: tk.Tk) -> None:
        self.item_listbox = tk.Listbox(list_frame, ...)
        self.borrow_button = ttk.Button(side, text="Borrow Selected")
        self.return_button = ttk.Button(side, text="Return Selected")
        self.remove_button = ttk.Button(side, text="Remove Selected")
        self.status_label = tk.Label(center, text="Ready.", anchor="w")
        # ... title_entry/author_entry/isbn_entry/year_entry, add_button ...
        self._current_items: list[LibraryItem] = []

    def selected_item(self) -> LibraryItem | None:
        selection = self.item_listbox.curselection()
        if not selection:
            return None
        return self._current_items[selection[0]]

    def refresh(self, items: list[LibraryItem]) -> None:
        previously_selected = self.selected_item()
        self._current_items = items
        self.item_listbox.delete(0, tk.END)
        for item in items:
            self.item_listbox.insert(tk.END, item.describe())
        if previously_selected is not None:
            for index, item in enumerate(items):
                if item is previously_selected:
                    self.item_listbox.selection_set(index)
                    break

    def set_status(self, message: str) -> None:
        self.status_label.config(text=message)
```

Listbox hanya menyimpan string tampilan -- View menyimpan daftar paralelnya sendiri untuk objek sungguhan

Berbeda dengan ListView<LibraryItem> milik JavaFX (yang bisa menyimpan objek sungguhan langsung, lewat cell factory yang memanggil describe() untuk tampilan), Listbox Tkinter hanya pernah menyimpan string polos. selected_item() menjembatani celah itu: _current_items dijaga sinkron dengan apa pun yang terakhir ditampilkan refresh(), sehingga sebuah INDEKS baris terpilih dari Listbox bisa dipetakan kembali ke objek LibraryItem sungguhan di baliknya -- method describe() yang sama yang sudah diekspos setiap LibraryItem (Bab 7-9) menyediakan string tampilannya, persis seperti pada jalur Java.

11.5 Controller: Memasang Event

LibraryController adalah satu-satunya kelas di bab ini yang meng-import baik Library maupun LibraryView. Konstruktornya memanggil _wire_events() (memasang satu callback command= per tombol) lalu _refresh_view() sekali, sehingga jendela terbuka sudah menampilkan keadaan awal Model. Setiap handler mengikuti bentuk tiga-langkah yang sama: baca seleksi atau field entry View saat ini, suruh Model melakukan sesuatu, lalu panggil _refresh_view() agar View menampilkan ulang apa pun keadaan Model sekarang (Gambar 11.3).

Penanganan Event, Dibandingkan: Dua Toolkit, Satu Gagasan

Kedua toolkit GUI memanggil kode Anda HANYA sebagai respons atas aksi pengguna yang nyata -- tetapi cara callback dipasang, dan apa yang diterimanya, berbeda.

	Java: JavaFX	Python: Tkinter
Memasang callback	button.setOnAction(e -> handler()) lambda yang mengimplemen-tasikan EventHandler<ActionEvent>	button.config(command=handler) sebuah callable polos tanpa argumen
Apa yang diterima callback	Sebuah objek ActionEvent (sering tak dipakai, tapi selalu diberikan)	Tidak ada -- callback command= Tkinter tidak menerima argumen sama sekali
Siapa menjalankan event loop	Application.launch() memulai thread rendering + event milik JavaFX sendiri	root.mainloop() memulai event loop Tk sendiri pada thread pemanggil

Tak satu pun toolkit memanggil kode handler Anda sebelum pengguna benar-benar mengklik -- keduanya mengembalikan kendali KE toolkit persis saat handler Anda selesai. "Event handler" bermakna gagasan yang sama di kedua pohon; objek yang diterimanya (atau tidak) dan siapa pemilik loop hanyalah detail toolkit.

Gambar 11.3 — Penanganan event, dibandingkan: setOnAction(...) milik JavaFX dan command=... milik Tkinter sama-sama memasang callback yang dipanggil toolkit hanya sebagai respons atas aksi pengguna yang nyata, tetapi berbeda dalam apa yang diterima callback dan siapa pemilik event loop.

library_controller.py (kutipan)

```
class LibraryController:
    def __init__(self, library: Library, view: LibraryView) -> None:
        self.library = library
        self.view = view
        self._wire_events()
        self._refresh_view()

    def _wire_events(self) -> None:
        self.view.add_button.config(command=self._handle_add_book)
        self.view.borrow_button.config(command=self._handle_borrow_selected)
        self.view.return_button.config(command=self._handle_return_selected)
        self.view.remove_button.config(command=self._handle_remove_selected)

    def _handle_borrow_selected(self) -> None:
        selected = self.view.selected_item()
        if selected is None:
            self.view.set_status("Borrow failed: no item selected.")
            return
        ok = selected.borrow()
        if ok:
            self.view.set_status(f"Borrowed: {selected.title}")
        else:
            self.view.set_status(
                f"Borrow refused: {selected.title} is unavailable "
                "(already on loan, or a reference-only item)."
            )
        self._refresh_view()

    def _refresh_view(self) -> None:
        self.view.refresh(self.library.get_all_items())
```

Callback command= Tkinter TIDAK menerima argumen apa pun -- berbeda dengan setOnAction(...) milik JavaFX

Lambda setOnAction(e -> handler()) milik JavaFX selalu diberi sebuah objek ActionEvent, bahkan ketika handler-nya tidak pernah memakainya (Bagian 11.5 buku jalur Java). command=self._handle_borrow_selected milik Tkinter tidak mengoper objek event apa pun sama sekali -- config(command=...) mengharapkan sebuah callable tanpa-argumen polos, titik, dan sebuah handler yang ditulis mengharapkan sebuah argumen (def _handle_borrow_selected(self, event):) akan memunculkan TypeError persis pertama kali Tkinter mencoba memanggilnya dengan nol argumen. Ini adalah kesalahan sungguhan yang mudah dibuat siapa pun yang datang dari JavaFX (atau dari API callback mana pun yang memang mengoper objek event) ke jalur Tkinter bab ini.

11.6 Titik Masuk Aplikasi: library_app.py

Fungsi main() di library_app.py adalah titik masuk program ini: ia membangun Model (sebuah Library yang sudah diisi empat item yang sama yang dipakai mata kuliah ini sejak Bab 7-10), membangun View dan Controller yang menghubungkannya ke Model itu, lalu memanggil root.mainloop(). Setelah pemanggilan itu, kendali berpindah ke event loop milik Tkinter sendiri, dan tidak ada apa pun lagi di main() yang berjalan selama sesi interaktif biasa -- setiap baris perilaku berikutnya datang dari handler Controller yang merespons klik sungguhan.

library_app.py (kutipan)

```
def main() -> None:
    library = Library()
    library.add_item(Book("Clean Code", "Robert C. Martin", "9780132350884",
2008))
    library.add_item(DVD("The Imitation Game", "9900000000001", 114))
    library.add_item(ReferenceBook("Encyclopedia of Computer Science",
                                   "Ralston & Reilly", "9780123456789", 2003))
    library.add_item(Book("Algorithms", "Robert Sedgewick", "9780321573513",
2011))

    root = tk.Tk()
    view = LibraryView(root)
    LibraryController(library, view)

    root.mainloop()

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

root.mainloop() dan event loop milik Tkinter sendiri

mainloop() adalah yang sungguh memulai pemrosesan event Tkinter: ia mulai menunggu event sistem-jendela/X11 (klik, penekanan tombol keyboard, permintaan tutup jendela) dan mengirimkan masing-masing ke callback command= widget mana pun (atau binding lain) yang didaftarkan untuknya. main() yang memanggil root.mainloop() lalu tidak menjalankan apa-apa lagi sesudahnya adalah normal dan diharapkan -- tidak ada baris kode dalam program ini yang menjalankan sesuatu setara while(true); Tkinter menyediakan loop itu sendiri, pada thread pemanggil, dan program hanya pernah bereaksi.

11.7 Menyatukan Semuanya: Memverifikasi GUI Sungguh Berjalan

Skrip bab konsol menghasilkan transkrip teks yang trivial untuk direproduksi: jalankan lagi, dapatkan baris print() yang identik. GUI tidak memiliki output teks setara secara default, sehingga library_app.py bab ini tambahan mendukung jalur verifikasi terskrip, digerbangi di belakang sebuah flag command-line --verify (Python tidak punya mekanisme system-property bawaan seperti JVM, sehingga sebuah flag command-line polos yang dibaca dari sys.argv adalah pengganti alami di sini -- perbedaannya hanya pada BAGAIMANA flag itu dioper, bukan pada apa yang sungguh dibuktikan verifikasinya), yang tidak pernah diambil eksekusi interaktif biasa. Ketika flag itu ada, root.after(200, ...) menjadwalkan run_scripted_verification(), yang menyimpan screenshot jendela yang baru dibuka, memilih baris list pertama SECARA TERPROGRAM (view.item_listbox.selection_set(0)), lalu memanggil view.borrow_button.invoke() -- API milik Tkinter sendiri untuk memicu tombol persis seperti klik mouse sungguhan, berjalan lewat callback command= yang identik dengan yang dikirim sebuah klik -- lalu menyimpan screenshot kedua sesudahnya.

library_app.py -- run_scripted_verification() (kutipan)

```
def run_scripted_verification(root: tk.Tk, view: LibraryView) -> None:
    save_snapshot(root, "gui_screenshot_1_initial.png")

    view.item_listbox.selection_set(0)
    view.borrow_button.invoke()   # jalur kode yang sama persis dengan klik
mouse sungguhan
    print(f'status now reads: "{view.status_label.cget("text")}"')

    save_snapshot(root, "gui_screenshot_2_after_borrow.png")
    root.destroy()

def save_snapshot(root: tk.Tk, file_name: str) -> None:
    root.update_idletasks()
    root.update()
    left = root.winfo_rootx()
    top = root.winfo_rooty()
    right = left + root.winfo_width()
    bottom = top + root.winfo_height()
    screenshot = ImageGrab.grab(bbox=(left, top, right, bottom))
    screenshot.save(file_name)
```

Button.invoke() tidak mensimulasikan atau mendeskripsikan sebuah klik; ia menjalankan callback command= yang identik dengan yang dipanggil Tkinter ketika mouse sungguh mengklik tombol itu, sehingga screenshot dan teks status yang dihasilkan adalah bukti sungguhan bahwa rantai handler LibraryController -- baca seleksi View, ubah Model lewat selected.borrow(), suruh View menyegarkan tampilan -- bekerja benar dari ujung ke ujung, bukan deskripsi tertulis tentang apa yang diharapkan penulis akan terjadi. Kedua screenshot direproduksi di Lampiran 11.A, berdampingan dengan log teks lengkap yang dihasilkan eksekusi ini.

11.8 Mengkompilasi dan Menjalankan Program Anda

Python tidak punya langkah kompilasi terpisah, tetapi bab ini membutuhkan interpreter spesifik dengan alasan berbeda dari sintaks PEP 695 Bab 10: Tkinter sendiri hanya bisa dipakai lewat paket

sistem python3-tk, dan, di lingkungan mata kuliah ini, paket itu dikompilasi khusus terhadap python3.12 -- baik python3 default (3.11) maupun python3.13 sama sekali tidak punya tkinter yang berfungsi. Ini adalah rekomendasi python3.12 yang sama yang sudah ditetapkan Bab 10 untuk sintaks PEP 695, kini dengan alasan kedua yang independen -- selalu jalankan kode bab ini dengan python3.12 secara eksplisit, jangan pernah dengan python3 polos.

```
$ python3.12 library_app.py

# eksekusi verifikasi terskrip (menghasilkan kedua screenshot di Lampiran 11.A):
$ python3.12 library_app.py --verify
```

python3.13 sama sekali tidak punya paket tkinter yang bisa diinstal di lingkungan ini

Bab 10 menetapkan bahwa python3.12 atau python3.13 sama-sama dibutuhkan untuk sintaks PEP 695; bab ini mempersempitnya lagi. python3-tk (paket apt yang menyediakan tkinter) dikompilasi khusus terhadap python3.12 di lingkungan mata kuliah ini, dan tidak ada paket python3.13-tk sama sekali yang bisa diinstal -- import tkinter di bawah python3.13 gagal dengan ModuleNotFoundError, bukan karena kode tkinter hilang, tetapi karena tidak pernah ada paket terkompilasi yang cocok yang tersedia untuk diinstal. Mulai bab ini, kode GUI jalur Python membutuhkan python3.12 secara spesifik, bukan sekadar "3.12 atau lebih baru".

Memverifikasi GUI sungguhan tanpa layar fisik

Kode bab ini dijalankan dalam lingkungan sandbox tanpa layar fisik terpasang, memakai Xvfb, sebuah virtual X11 display server: xvfb-run -a python3.12 library_app.py --verify memberi Tkinter sebuah display untuk digambar yang hanya ada di memori, membiarkan program berjalan, mengklik lewat event handler sungguhannya, dan menghasilkan screenshot genuine tanpa monitor. Ini diungkapkan di sini, bukan disamarkan: verifikasinya sepenuhnya nyata -- kode Tkinter yang sama, pengiriman event yang sama, pipeline rendering yang sama yang dipakai eksekusi desktop -- hanya display itu sendiri yang virtual, bukan disimulasikan. Kebutuhan itu soal sandbox tidak punya monitor sama sekali, bukan soal mekanisme screenshot itu sendiri: save_snapshot() (Bagian 11.7) menangkap lewat PIL.ImageGrab.grab() milik Pillow, yang tidak butuh display virtual apa pun di desktop biasa -- eksekusi --verify di mesin Windows, macOS, atau Linux sungguhan langsung men-screenshot jendela nyata yang terlihat.

11.9 OOP dalam Praktik: Profitina

Tentang kotak ini

Seperti di setiap bab, kotak ini menghubungkan materi kuliah dengan Profitina, platform AI-visibility (Generative Engine Optimization) Indonesia yang nyata dan aktif, dibangun oleh penulis -- menjelaskan DI MANA dan MENGAPA sebuah konsep digunakan dalam sistem nyata yang berjalan, tidak pernah logika bisnis atau kode sumber lengkap Profitina, yang tetap rahasia.

Dashboard admin internal Profitina mengikuti pemisahan MVC yang sama yang diajarkan bab ini: sebuah lapisan layanan backend (Model, dalam semangatnya) yang tidak tahu apa-apa tentang bagaimana dashboard mana pun menggambar sebuah chart, sebuah lapisan rendering (View) yang hanya tahu cara menggambar data apa pun yang diberikan kepadanya, dan sebuah lapisan koordinasi tipis (Controller) yang membaca apa yang diklik operator, meminta layanan backend melakukan sesuatu, dan menyuruh lapisan rendering menggambar ulang. Disiplin yang sama yang ditegakkan bab ini pada skala mainan -- Model tidak pernah meng-import apa pun tentang View -- adalah yang

membiarkan backend Profitina diuji, bahkan dipakai ulang, sepenuhnya terpisah dari dashboard tertentu mana pun yang dibangun di atasnya.

11.10 Ringkasan Bab dan Poin-Poin Utama

- Pola Model-View-Controller memecah program GUI menjadi Model (data dan aturan, tanpa pengetahuan GUI), View (widget yang terlihat, nol logika bisnis), dan Controller (satu-satunya kelas yang boleh bergantung pada keduanya).
- Hierarki Library dan LibraryItem milik Bab 5-10 hanya perlu tepat satu method baru (get_all_items(), aksesor yang mengembalikan salinan) agar bisa dipakai dari View GUI -- setiap pemanggilan borrow()/return_item()/remove_item_or_raise() dipakai ulang sepenuhnya tanpa perubahan.
- Pemrograman berbasis event membalik alur kendali skrip konsol: event loop milik Tkinter sendiri, bukan kode programmer, yang memutuskan kapan sebuah handler berjalan, memanggil balik tepat callable yang dipasang dengan command= untuk widget apa pun yang baru saja berinteraksi dengan pengguna.
- Callback command= Tkinter TIDAK menerima argumen apa pun, berbeda dengan setOnAction(...) milik JavaFX, yang selalu mengoper sebuah ActionEvent -- bentuk handler tiga-langkah yang sama (baca View, ubah Model, segarkan View) tetap berlaku pada kedua jalur, tetapi signature callback-nya berbeda.
- Button.invoke() menjalankan jalur kode event-handler yang identik dengan yang dikirim klik mouse sungguhan, membuat eksekusi verifikasi terskrip tanpa layar menjadi bukti sungguhan bahwa GUI-nya bekerja, bukan deskripsi perilaku yang diharapkan.

View yang tidak pernah meng-import aturan bisnis Model, dan Model yang tidak pernah meng-import apa pun tentang View, bukanlah cita-cita abstrak di bab ini -- itulah alasan Listbox milik library_view.py bisa ditukar dengan Treeview, atau library_view.py itu sendiri diganti dengan toolkit yang sama sekali berbeda, tanpa mengubah satu baris pun di library.py.

11.11 Pertanyaan Review

Pertanyaan-pertanyaan ini untuk belajar mandiri dan diskusi kelas; tidak dinilai.

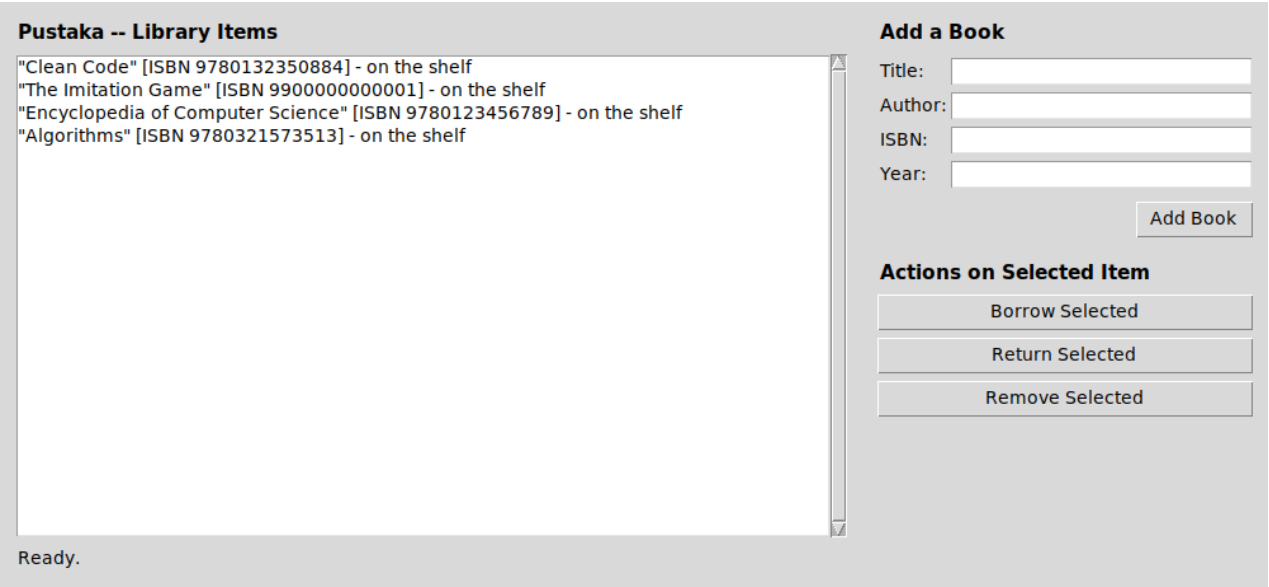
1. LibraryController._handle_remove_selected() menangkap ItemNotFoundError meskipun ISBN yang dioper ke remove_item_or_raise() datang langsung dari sebuah LibraryItem yang sudah ada di dalam list milik View sendiri. Jelaskan mengapa blok except itu tetap layak ada dalam kodenya, dan mengapa cabang di dalamnya seharusnya tidak pernah sungguh berjalan dalam praktik.
2. Andaikan LibraryView mengekspos library secara langsung (atribut publik) alih-alih hanya mengekspos widgetnya sendiri. Deskripsikan satu perubahan yang bisa dibuat LibraryController yang akan melanggar batas MVC yang ditetapkan bab ini, dan jelaskan apa yang akan salah jika dua View berbeda sama-sama mencoba melakukan ini.
3. view.borrow_button.invoke() dipakai dalam eksekusi verifikasi bab ini alih-alih memanggil langsung library.find_by_title(...).borrow() dari kode verifikasi. Jelaskan apa yang gagal dibuktikan oleh pemanggilan Model langsung semacam itu yang berhasil dibuktikan invoke().
4. Callback command= Tkinter tidak menerima argumen apa pun, sementara setOnAction(...) milik JavaFX selalu menerima sebuah ActionEvent. Tuliskan ulang _handle_borrow_selected

seolah-olah ia tetap perlu menerima sebuah argumen event (def _handle_borrow_selected(self, event=None):), dan jelaskan mengapa memberi parameter itu nilai default adalah yang membuat command=self._handle_borrow_selected tetap bekerja tanpa perubahan.

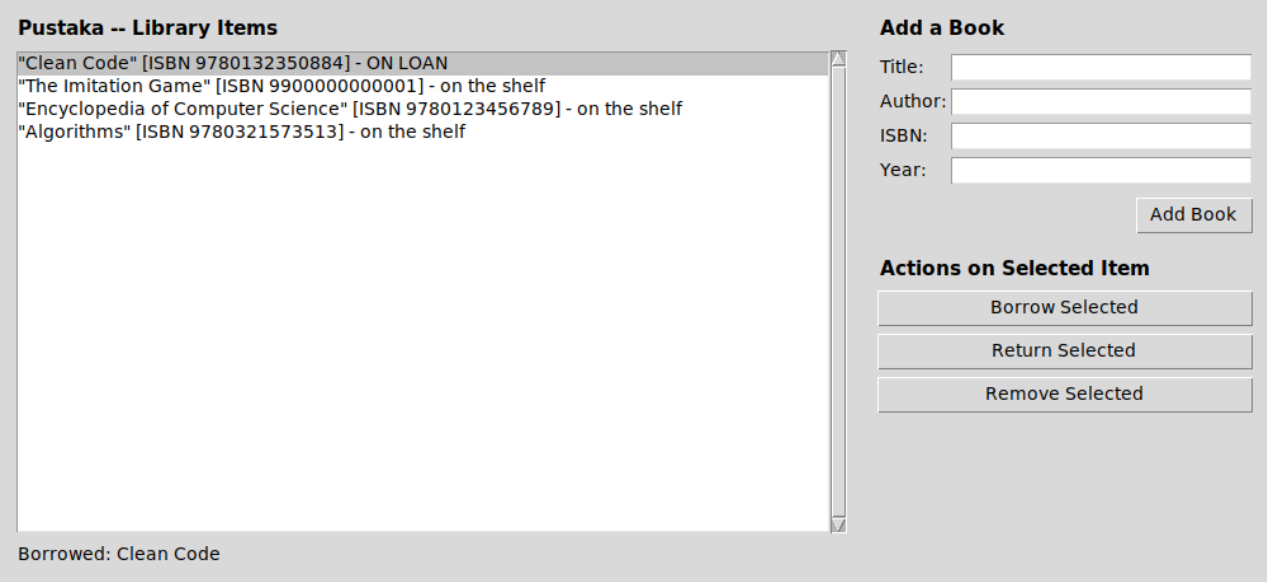
5. Tuliskan ulang _handle_borrow_selected() sehingga, alih-alih memanggil selected.borrow() secara langsung, ia lebih dulu memeriksa selected.is_on_loan dan menyetel pesan status "Already on loan." sebelum pernah memanggil borrow() sama sekali. Apakah ini akan mengubah antarmuka publik Model? Jelaskan jawaban Anda.

Lampiran 11.A — Log Verifikasi Eksekusi

Berkas lengkap library_item.py, book.py, dvd.py, reference_book.py, item_not_found_error.py, pair.py, library_utils.py, library.py, library_view.py, library_controller.py, dan library_app.py (Code/Python/Chapter11/, disertakan bersama buku ini) dijalankan di bawah Xvfb pada python3.12 (dengan tkinter 8.6) sebelum bab ini ditulis, memakai jalur terskrip --verify yang dideskripsikan Bagian 11.7. Kedua screenshot di bawah adalah output genuine, tak-diedit, dari eksekusi itu -- bukan mockup -- dan log teks di bawahnya direproduksi verbatim (Gambar 11.4 dan 11.5).



Gambar 11.4 — Screenshot 1: jendela segera setelah dibuka. Keempat item menunjukkan "on the shelf"; belum ada yang dipilih.



Gambar 11.5 — Screenshot 2: setelah eksekusi terskrip memilih baris 0 dan memicu Borrow Selected. "Clean Code" tersorot dan kini terbaca "ON LOAN"; label status terbaca "Borrowed: Clean Code".

```
$ xvfb-run -a python3.12 library_app.py --verify
--- Chapter 11 GUI verification run ---
Snapshot 1 saved: initial window, 4 items loaded, nothing selected.
Selected list row 0 (programmatically, same selection a click would make).
Fired Borrow Selected button -> status now reads: "Borrowed: Clean Code"
Snapshot 2 saved: after borrowing the selected item.
```

Referensi

[1] Python Software Foundation. "tkinter — Python interface to Tcl/Tk." Python 3 documentation. https://docs.python.org/3/library/tkinter.html (diakses Agustus 2026).

[2] Python Software Foundation. "An Introduction to Tkinter — Events and Bindings." Python 3 documentation. https://docs.python.org/3/library/tkinter.html#bindings-and-events (diakses Agustus 2026).

[3] G. E. Krasner and S. T. Pope. "A Description of the Model-View-Controller User Interface Paradigm in the Smalltalk-80 System." Journal of Object-Oriented Programming, 1(3), 1988.

[4] X.Org Foundation. "Xvfb — virtual framebuffer X server." X.Org manual pages. https://www.x.org/releases/X11R7.7/doc/man/man1/Xvfb.1.xhtml (diakses Agustus 2026).

BAB 12 · BAB LATIHAN

Soal Latihan: Kuis 2 — Meninjau Ulang Kuliah 1 Sampai 11

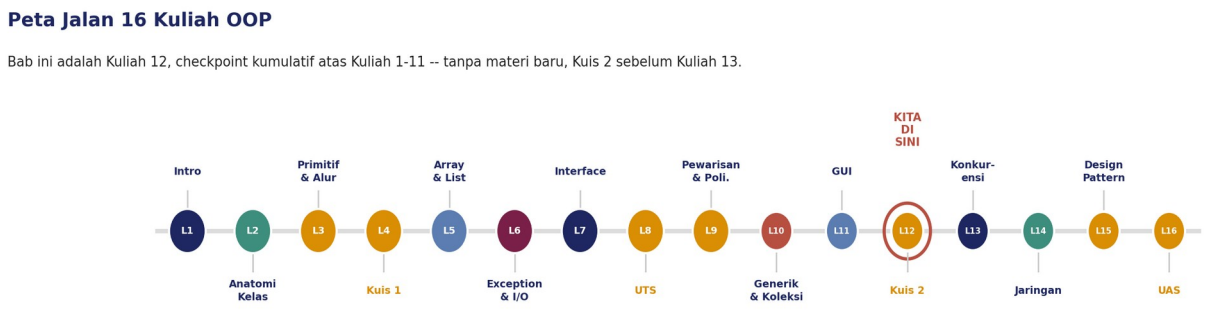
Tidak ada materi baru di sini -- delapan soal yang mencakup semuanya dari enkapsulasi hingga desain Model-View-Controller, meninjau ulang setiap kuliah sejak Kuliah 1 sebelum Kuis 2.

Tujuan Pembelajaran — setelah mengerjakan bab ini Anda akan mampu:

(1) Merancang kelas terenkapsulasi kecil yang menyimpan sepasang atribut dan menurunkan nilai lain darinya sesuai permintaan. (2) Menulis logika alur kontrol yang berperilaku benar pada setiap nilai batas, bukan hanya kasus yang jelas. (3) Membangun string terformat secara efisien dari sebuah list, menggunakan join yang idiomatis alih-alih konkatenasi berulang. (4) Mendefinisikan dan melempar exception kustom yang membiarkan state sebuah objek tidak tersentuh saat gagal, serta membaca dan menulis berkas log teks polos dengan aman. (5) Merancang ABC dengan method konkret yang dibangun di atas method abstrak, dan meng-extend kelas dengan pewarisan sungguhan yang memanggil implementasi induk lewat super() serta melakukan dispatch polimorfik atas list campuran. (6) Menulis fungsi generik yang dibatasi ke tipe yang bisa diurutkan, membangun indeks menggunakan dict, dan merancang desain Model-View-Controller di mana setiap kelas memiliki satu tanggung jawab yang jelas batasnya.

12.1 Rekap: Kuliah 1–11 Secara Singkat

Kuliah 1 sampai 8 membangun perkakas inti bahasa ini dan sudah ditinjau ulang secara lengkap oleh UTS di Bab 8: pergeseran dari pemikiran prosedural ke berorientasi objek (Kuliah 1), anatomi kelas dan enkapsulasi (Kuliah 2), tipe numerik dan alur kontrol (Kuliah 3), Kuis 1 (Kuliah 4), list dan pembangunan string (Kuliah 5), exception dan I/O berkas (Kuliah 6), abc.ABC dan typing.Protocol (Kuliah 7), serta UTS itu sendiri (Kuliah 8). Kuliah 9 membangun langsung di atas ABC dan Protocol Kuliah 7 dengan pewarisan sungguhan -- subclassing, super(), method overriding, dan dynamic dispatch yang di-resolve Python saat runtime lewat aturan duck-typing-nya sendiri. Kuliah 10 menambahkan generics Python (TypeVar, Generic) dan koleksi bawaannya, memungkinkan hierarki LibraryItem milik Pustaka disimpan, diurutkan, dan dicari dalam list, set, atau dict yang beranotasi tipe, bukan list biasa. Kuliah 11 membawa Library sepenuhnya keluar dari konsol, membangun LibraryView dan LibraryController di atas event loop milik Tkinter sendiri dengan pola Model-View-Controller -- pola yang sama yang diminta Soal 8 di bawah untuk Anda rancang bagi aplikasi yang jauh lebih kecil (Gambar 12.1).



Gambar 12.1 — Bab ini berada di Kuliah 12 dalam peta jalan 16-kuliah OOP: sebuah checkpoint, bukan topik baru, Kuis 2 sebelum Kuliah 13.

12.2 Cara Menggunakan Bab Latihan Ini

Apa bab ini -- dan bukan apa

Ini adalah Bab Latihan: bab praktik, bukan bab materi baru. Setiap soal di bawah berakar pada materi spesifik dari salah satu dari delapan kuliah sejak Kuliah 1 yang mengajarkan konten baru (lihat Gambar 12.2). Setiap soal disertai PETUNJUK -- dorongan ke arah cara berpikir yang tepat -- tapi sengaja BUKAN solusi lengkap atau kode sumber. Kedelapan soal ini berdiri sendiri, independen dari studi kasus Pustaka yang berjalan di seluruh bab biasa -- persis seperti soal-soal Kuis 1 di Bab 4 dan soal-soal UTS di Bab 8.

Asal Setiap Soal Latihan

Setiap soal di Bagian 12.3 dapat ditelusuri kembali ke salah satu dari delapan kuliah sejak Kuliah 1 yang mengajarkan materi baru.

#	Soal	Berasal dari
1	Kelas Rectangle	K2 -- atribut, konstruktor, enkapsulasi
2	Kategori BMI	K3 -- alur kendali, kondisi batas
3	Pemformat Baris CSV	K5 -- pembangunan string, list
4	Penjaga Saldo Negatif	K6 -- kelas exception kustom, berkas log transaksi
5	Hierarki Shape	K7 -- ABC, method konkret di atas method abstrak
6	Vehicle & Car	K9 -- pewarisan, polimorfisme, super().describe()
7	max_of Generik	K10 -- TypeVar terbatas, indeks berbasis dict
8	MVC Counter Mini	K11 -- Model-View-Controller, desain berbasis event

Gambar 12.2 — Setiap soal di Bagian 12.3 dapat ditelusuri kembali ke salah satu dari delapan kuliah sejak Kuliah 1 yang mengajarkan materi baru.

Sebelum Anda mulai

Coba setiap soal sungguh-sungguh dalam berkas .py baru sebelum membaca petunjuknya. Jika buntu, baca hanya petunjuknya -- bukan solusi -- lalu coba lagi. Ini persis disiplin yang akan dihargai oleh Kuis 2 di Bagian 12.4: asesmennya open-book, tapi itu bukan pengganti penalaran Anda sendiri.

12.3 Soal Latihan

12.3.1 Review Atribut, Konstruktor & Enkapsulasi

Soal 1 *[Mudah]*. Definisikan kelas Rectangle(width, height), hanya menyimpan kedua dimensi, dengan get_width(), get_height(), get_area() (width * height), dan get_perimeter() (2 * (width + height)).

Petunjuk

Simpan hanya atribut width dan height di __init__. Baik get_area() maupun get_perimeter() harus menghitung hasilnya sesuai permintaan dari kedua atribut itu, bukan menyimpan area dan perimeter sebagai atribut yang bisa jadi tidak sinkron jika width atau height pernah diubah -- persis disiplin yang sama dengan Temperature di Bab 8.

12.3.2 Review Tipe Numerik & Alur Kontrol

Soal 2 *[Mudah]*. Tulis fungsi bmi_category(weight_kg, height_m) yang menghitung bmi = weight_kg / (height_m ** 2) dan mengembalikan "Underweight" untuk bmi < 18,5, "Normal" untuk bmi < 25,0, "Overweight" untuk bmi < 30,0, dan "Obese" jika tidak. Uji pada 18,5, 18,49, 25,0, 24,99, 30,0, dan 29,99.

> **Petunjuk**
>
> Urutkan perbandingan Anda dari batas terendah ke tertinggi, menggunakan elif, dan kembali begitu satu cocok. Kasus 18,49/24,99/29,99 sengaja ada untuk menangkap kesalahan < versus <= persis pada nilai batas, yang tidak akan pernah terungkap hanya dengan angka bulat.

12.3.3 List & Pembangunan String

Soal 3 *[Sedang]*. Tulis fungsi format_csv_row(fields) yang menggabungkan field dengan koma menjadi satu baris CSV, kecuali field yang mengandung koma harus dibungkus tanda kutip ganda terlebih dahulu (versi sederhana dari aturan quoting CSV yang sesungguhnya). Contoh: ["Ann", "Budi, Jr.", "30"] menjadi Ann,"Budi, Jr.",30. Bangun hasilnya secara idiomatis, bukan dengan konkatenasi string berulang dalam sebuah loop.

> **Petunjuk**
>
> Bangun list baru di mana setiap field dibiarkan apa adanya atau dibungkus tanda kutip ganda yang di-escape (berdasarkan apakah "," muncul di field itu), lalu panggil ",".join(...) sekali pada list itu -- persis idiom join-di-akhir yang sudah dipakai format_roster() di Bab 8.

12.3.4 Penanganan Exception & I/O Berkas

Soal 4 *[Sedang]*. Definisikan kelas exception kustom NegativeBalanceError(Exception), dan kelas Account(opening_balance) dengan method withdraw(self, amount) yang melempar NegativeBalanceError (pesannya menyertakan baik jumlah yang diminta maupun saldo saat ini) jika amount melebihi saldo saat ini, dan jika tidak mengurangi amount dari saldo serta menambahkan satu baris "WITHDRAW,amount,balance_after" ke berkas log transaksi.

> **Petunjuk**
>
> Periksa kondisi saldo tidak cukup dan lempar exception SEBELUM menyentuh atribut saldo atau membuka berkas -- penarikan yang gagal harus membiarkan keduanya sepenuhnya tidak tersentuh. Setelah pemeriksaan lolos, perbarui atribut saldo terlebih dahulu, lalu gunakan with open(path, "a") as f: untuk menambahkan baris log -- persis pola yang dipakai save_to_file di Bab 6 untuk Pustaka dan append_score_record di Bab 8 untuk skor.

12.3.5 ABC & Protocol

Soal 5 *[Sedang]*. Definisikan kelas abstrak Shape(ABC) dengan method abstrak area(self), dan method KONKRET describe(self) yang dibangun sepenuhnya dari area() (mis. mengembalikan sesuatu seperti "This shape covers 28.27 square units."). Lalu definisikan dua subclass konkret: Circle(radius) dan Square(side).

> **Petunjuk**
>
> describe() sebaiknya berada di kelas abstrak itu sendiri, bukan di salah satu subclass -- ia sudah sepenuhnya bisa diimplementasikan dalam istilah area() apa pun yang akhirnya disediakan tiap subclass, persis seperti Employee.annual_pay() di Bab 8 yang merupakan method konkret dibangun di atas monthly_pay() yang abstrak.

12.3.6 Pewarisan & Polimorfisme

Soal 6 *[Sedang]*. Definisikan kelas Vehicle(make, model) dengan method describe(self) yang mengembalikan "make model". Lalu definisikan subclass Car(make, model, door_count) yang meng-OVERRIDE describe() untuk memanggil super().describe() dan menambahkan jumlah pintu, mis. "Toyota Corolla (4 doors)". Terakhir, bangun sebuah list berisi campuran objek Vehicle biasa dan Car, lalu panggil describe() pada masing-masing dalam sebuah loop.

> **Petunjuk**
>
> Car.describe() harus memanggil super().describe() alih-alih memformat ulang make dan model sendiri -- itulah pewarisan yang menggunakan ulang logika induk, bukan menduplikasinya. Loop atas list campuran itu adalah dispatch polimorfik: meskipun setiap elemen secara konseptual adalah Vehicle, setiap pemanggilan describe() di-resolve saat runtime ke kelas mana pun objek sesungguhnya berasal -- duck typing Python membuat ini otomatis, tapi memanggil super() tetap disiplin yang harus Anda terapkan sendiri.

12.3.7 Generics & Collections

Soal 7 *[Sulit]*. Tulis fungsi generik max_of(items: list[T]) -> T (dengan T sebuah TypeVar yang dibatasi ke tipe yang bisa diurutkan) yang mengembalikan elemen terbesar dari items, melempar ValueError untuk list kosong. Lalu tulis fungsi group_by_first_letter(words: list[str]) -> dict[str, list[str]] yang mengelompokkan kata ke dalam dict berdasarkan huruf pertamanya (dikapitalkan).

> **Petunjuk**
>
> Python tidak memiliki batasan saat-kompilasi seperti <T extends Comparable<T>> milik Java, tapi mendeklarasikan T = TypeVar("T", bound=...) mendokumentasikan persyaratan urutan yang sama bagi siapa pun yang membaca signature-nya, meskipun Python hanya memeriksanya saat runtime, bukan saat kompilasi. Untuk group_by_first_letter, gunakan dict.setdefault(key, []).append(word), atau collections.defaultdict(list), agar Anda tidak perlu pemeriksaan 'apakah key ini sudah ada di dict' terpisah sebelum menambahkan ke list sebuah grup.

12.3.8 GUI & MVC

Soal 8 *[Sulit]*. Rancang aplikasi counter MVC mini: kelas Model CounterModel dengan increment(), decrement(), dan sebuah property value; kelas View yang menampilkan nilai saat ini dalam sebuah label serta menyediakan tombol "+" dan tombol "-"; dan kelas Controller yang menjadi SATU-SATUNYA kelas yang memegang referensi ke Model maupun View, menghubungkan command setiap tombol ke method Model yang sesuai lalu me-refresh label View. Sketsakan atribut dan signature method ketiga kelas itu -- Anda tidak perlu GUI yang sungguh berjalan untuk menjawab soal ini.

Petunjuk

Persis seperti LibraryView/LibraryController di Bab 11: Model harus nol import terhadap tkinter, View harus nol logika bisnis (tidak ada increment yang terjadi di dalam View), dan hanya Controller yang boleh meng-import dan bergantung pada keduanya. Jika command callback tombol mana pun mengubah count internal CounterModel secara langsung alih-alih memanggil increment()/decrement(), maka View itu telah merusak enkapsulasi persis seperti yang diperingatkan Bab 11.

12.4 Format Kuis 2: Open-Book, Individu, Timed

Kuis 2 adalah penilaian take-home open-book, individu, dan timed (kurang lebih 120 menit) yang mencakup persis delapan jenis soal yang direview di bab ini -- sebuah checkpoint kumulatif atas semuanya sejak Kuliah 1, bukan hanya materi sejak UTS. Penilaian dilakukan per-soal, bukan semua-atau-tidak-sama-sekali: nilai parsial diberikan untuk kode yang berjalan dan menangani kasus umum dengan benar sekalipun satu edge case terlewat.

Open-book berarti referensi, bukan kolaborator

Anda boleh berkonsultasi dengan buku teks, catatan Anda sendiri, dan slide kuliah saat menjawab. Anda TIDAK BOLEH berkolaborasi dengan teman sekelas atau mengumpulkan kode yang bukan benar-benar milik Anda sendiri -- kuis open-book yang timed menguji apakah Anda bisa MENERAPKAN apa yang sudah Anda pelajari, tanpa pengawasan, dalam batas waktu.

Kriteria	Yang dinilai	Bobot
Kebenaran output	Apakah kelas atau fungsi menghasilkan hasil yang persis sesuai harapan untuk setiap kasus uji yang diberikan, termasuk nilai batas?	45%
Desain kelas, ABC & generik	Apakah atribut hanya dijangkau lewat method, dan apakah pemisahan ABC (Soal 5) serta batasan TypeVar (Soal 7) digunakan dengan benar?	25%
Disiplin exception, I/O & MVC	Apakah exception Soal 4 membawa pesan yang berguna dan membiarkan state akun tidak tersentuh saat gagal, apakah I/O berkas-nya menggunakan pernyataan with dengan benar, dan apakah sketsa Soal 8 menjaga Model, View, dan Controller terpisah dengan bersih?	15%
Kejelasan kode & berjalan bersih	Apakah kode mudah dibaca, penamaannya masuk akal, dan bebas dari kode mati -- serta berjalan tanpa error?	15%

12.5 OOP dalam Praktik: Profitina

Tentang kotak ini

Seperti di setiap bab, kotak ini menghubungkan materi kuliah dengan Profitina, platform AI-visibility (Generative Engine Optimization) Indonesia yang nyata dan aktif, dibangun oleh penulis -- menjelaskan DI MANA dan MENGAPA sebuah konsep digunakan dalam sistem nyata yang berjalan, tidak pernah logika bisnis atau kode sumber lengkap Profitina, yang tetap rahasia.

Kedelapan soal bab ini, digabungkan, mendekati versi mini dari apa yang sungguh dibutuhkan satu fitur di backend Python/FastAPI milik Profitina: satu atau dua kelas terenkapsulasi kecil (Soal 1), logika yang diuji-batas dengan cermat (Soal 2), penanganan teks yang efisien untuk sebuah laporan atau respons API (Soal 3), operasi yang harus gagal dengan aman tanpa merusak state yang sudah ada, dicatat secara persisten ke disk (Soal 4), sebuah kontrak bersama yang diimplementasikan oleh beberapa bentuk data konkret (Soal 5), hierarki yang menggunakan ulang perilaku induk alih-alih menduplikasinya (Soal 6), utilitas generik yang bisa dipakai lintas banyak tipe data yang tidak berkaitan ditambah indeks berbasis dict untuk pencarian cepat (Soal 7), dan pemisahan Model-View-Controller yang membuat logika render sebuah dashboard tidak peduli terhadap layanan backend yang

memasoknya (Soal 8, pemisahan yang sama yang digunakan dashboard admin Profitina sendiri). Kuis yang hanya menguji satu dari kemampuan ini akan melewatkan betapa seringnya fitur nyata membutuhkan beberapa di antaranya bekerja bersama dalam satu potongan kode kecil yang sama.

12.6 Ringkasan Bab

- Bab ini tidak memperkenalkan materi baru -- ini adalah checkpoint kumulatif yang mencakup Kuliah 1 sampai 11.
- Delapan soal berdiri sendiri mencakup seluruh rentang yang direview: enkapsulasi (K2), batas alur kontrol (K3), pembangunan string dan list (K5), exception kustom dan I/O berkas (K6), ABC (K7), pewarisan dan polimorfisme (K9), generics dan collections (K10), serta desain Model-View-Controller (K11).
- Setiap soal menyertakan petunjuk, bukan solusi -- mengerjakan penalaran dan kodenya sendiri adalah intinya.
- Kuis 2 adalah penilaian take-home open-book, individu, dan timed: referensi diperbolehkan, tapi kodenya harus milik Anda sendiri, dan kebenaran pada setiap kasus uji yang diberikan (termasuk nilai batas) adalah porsi terbesar nilai.

Kode yang berjalan tidak sama dengan kode yang benar -- dan desain yang terlihat rapi di atas kertas tidak sama dengan desain yang sungguh menjaga Model, View, dan Controller-nya agar tidak saling bergantung.

Lampiran 12.A — Checklist Self-Check (Tidak Dinilai)

Sebelum mengumpulkan jawaban apa pun -- pada soal bab ini atau pada Kuis 2 itu sendiri -- jalankan lewat checklist ini. Butuh waktu beberapa menit dan menangkap sebagian besar kesalahan yang paling sering dilihat penilai.

- Apakah setiap fungsi dan kelas berjalan tanpa error pada setiap kasus uji yang diberikan?
- Apakah atribut hanya dijangkau lewat method, tanpa kode di luar kelas yang menjangkau langsung ke dalamnya?
- Sudahkah Anda menguji nilai batas persis Soal 2 (18,49, 24,99, 29,99), bukan hanya angka bulat yang bisa menyembunyikan bug < versus <=?
- Apakah withdraw() pada Soal 4 membiarkan atribut saldo dan berkas log sepenuhnya tidak tersentuh ketika melempar NegativeBalanceError?
- Apakah method abstrak dan method konkret Soal 5 berada pada kelas abstrak yang SAMA, dengan method konkret memanggil yang abstrak alih-alih menduplikasi logika di setiap subclass?
- Apakah Car.describe() pada Soal 6 memanggil super().describe() alih-alih memformat ulang make dan model sendiri?
- Apakah max_of() pada Soal 7 mendokumentasikan persyaratan urutannya dengan TypeVar berbatas, dan apakah group_by_first_letter() menghindari pemeriksaan keanggotaan terpisah sebelum menambahkan ke sebuah grup?
- Sudahkah Anda membaca ulang kode Anda sendiri seolah-olah Anda penilai skeptis yang mencari satu input yang bisa merusaknya?

Referensi

[1] Format penilaian bab latihan ini (open-book, individu, timed) dimodelkan dari Kuis 2 nyata mata kuliah ini (EF234302 Pemrograman Berorientasi Objek), sebuah proyek kelompok membangun sistem prediksi teks T9. Tiga bagian pertama proyek itu berpusat pada tree rekursif bercabang banyak kustom, binary search atas array terurut, dan pencarian Map multi-nilai -- kelas materi Struktur Data yang sama yang sudah ditempatkan UTS nyata pembaruan ini (lihat Referensi [1] Bab 8) di luar cakupan OOP sendiri, sehingga bagian-bagian itu pun tidak digunakan ulang di sini. Bagian keempatnya, meskipun demikian, membangun GUI keypad ponsel menggunakan pola Model-View-Controller -- pola yang sama yang sungguh diajarkan pembaruan ini di Bab 11 -- sehingga Soal 8 di atas mengambil tema GUI/MVC-nya langsung dari bagian itu, sementara ketujuh soal lainnya dirancang baru untuk menguji tujuh area kuliah lain yang perlu dicakup Kuis 2 ini.

[2] Python Software Foundation. "typing — TypeVar and Generic." Python 3 documentation. https://docs.python.org/3/library/typing.html (diakses Agustus 2026).

[3] Python Software Foundation. "abc — Abstract Base Classes." Python 3 documentation. https://docs.python.org/3/library/abc.html (diakses Agustus 2026).

BAB 13

Konkurensi: Thread, Race Condition, dan Sinkronisasi

Setiap bab sejauh ini menjalankan satu bagian kode pada satu waktu, dalam satu urutan, yang sepenuhnya ditentukan oleh program itu sendiri. Sistem nyata -- termasuk kios publik Pustaka sendiri -- tidak memiliki kemewahan itu: beberapa patron bisa bertindak pada perpustakaan di saat yang persis sama. Bab ini menunjukkan, dengan bug nyata yang bisa Anda reproduksi sendiri, mengapa itu mengubah segalanya tentang bagaimana state bersama harus ditulis -- termasuk satu kejutan jujur tentang betapa sulitnya bug khusus ini untuk diamati secara alami di Python (Gambar 13.1).

Tujuan Pembelajaran — setelah bab ini Anda akan mampu:

(1) Menjelaskan mengapa operasi sesederhana count += 1 tidak atomik, dan mendeskripsikan secara konkret bagaimana dua thread bisa berselang-seling sehingga sebuah update hilang. (2) Membuat dan memulai banyak threading.Thread untuk menjalankan pekerjaan secara konkuren, dan meng-join()-nya untuk menunggu semuanya selesai. (3) Menggunakan threading.Lock untuk menegakkan mutual exclusion pada state bersama, dan menjelaskan persis apa yang dihalangi ketika sebuah thread memperoleh lock. (4) Menggunakan concurrent.futures.ThreadPoolExecutor untuk menjalankan banyak tugas singkat tanpa mengelola satu objek Thread secara manual per tugas. (5) Membaca hasil eksekusi program nyata dan membedakan race condition yang sekadar mungkin secara teoretis dari yang benar-benar teramati, dengan angka pasti yang menyertainya.

13.1 Rekap: Pustaka Sejauh Ini

Bab 9 menggeneralisasi Library agar bisa menyimpan LibraryItem mana pun secara polimorfik. Bab 10 menambahkan Generics dan koleksi bawaan Python. Bab 11 memberi Library antarmuka berjendela nyata yang dibangun di atas pola Model-View-Controller. Bab 12 adalah checkpoint, bukan materi baru. Setiap bab itu, dan setiap bab sebelumnya, berjalan pada satu thread: satu call stack, satu urutan eksekusi, sepenuhnya bisa diprediksi dari membaca kode sumber dari atas ke bawah. Bab ini adalah pertama kalinya hal itu berhenti berlaku.

Gambar 13.1 — Peta jalan 16-kuliah OOP. Bab ini adalah Kuliah 13: menjalankan lebih dari satu bagian logika Library pada saat yang sama, dengan aman.

13.2 Mengapa Konkurensi: Thread, State Bersama, dan GIL

Sebuah threading.Thread adalah jalur eksekusi independen di dalam program yang sama yang sedang berjalan, berbagi objek dan memori yang sama dengan setiap thread lain yang dibuat program itu. CPython (implementasi Python acuan yang dipakai mata kuliah ini) juga memiliki Global Interpreter Lock (GIL), yang hanya mengizinkan satu thread mengeksekusi bytecode Python pada satu instan mana pun -- tetapi GIL TIDAK membuat sebuah pernyataan Python multi-langkah menjadi atomik, dan tidak menghilangkan kebutuhan akan sinkronisasi eksplisit, seperti yang didemonstrasikan secara konkret oleh Bagian 13.3. Rencana deployment nyata Pustaka (terlepas dari GUI Bab 11) membayangkan beberapa kios publik, masing-masing membiarkan patron berbeda meminjam item pada saat yang sama -- dan jika kios-kios itu masing-masing didukung oleh thread-nya sendiri, semuanya bisa berakhir memanggil method pada objek bersama yang persis sama pada instan yang persis sama. Itulah situasi yang dirancang untuk diungkap oleh kelas baru bab ini, BorrowCounter: satu hitungan bersama berapa banyak item yang telah dipinjam di seluruh perpustakaan hari ini, diperbarui dari banyak thread sekaligus.

borrow_counter.py -- kontrak minimal untuk state bersama

```
@runtime_checkable
class BorrowCounter(Protocol):
    def increment(self) -> None: ...
    def get_count(self) -> int: ...
```

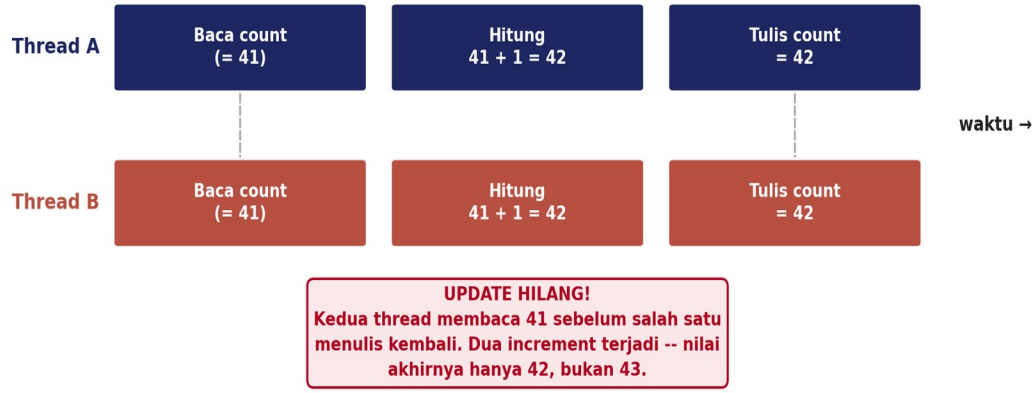
Dua kelas memenuhi Protocol ini secara struktural, persis seperti Book dan DVD di Bab 7 memenuhi Renewable. Salah satunya punya bug yang tak terlihat hanya dengan membaca kode sumber -- ia hanya muncul ketika thread sungguhan benar-benar menjalankannya.

13.3 Race Condition: Bug Nyata yang Bisa Direproduksi

self._count += 1 terlihat seperti satu operasi tunggal dan tak terbagi dalam kode sumber Python. Ternyata bukan. CPython melakukannya sebagai tiga langkah terpisah pada level bytecode: membaca nilai _count saat ini, menambahkan satu, dan menulis nilai baru kembali. GIL melindungi setiap instruksi bytecode individual, bukan satu pernyataan Python secara utuh -- sehingga tidak ada yang mencegah interpreter berpindah ke thread lain di antara tiga langkah itu, dan ketika itu terjadi, satu increment hilang secara diam-diam (Gambar 13.2).

Race Condition: Bagaimana Satu Increment Bisa Hilang

count++ / count += 1 sebenarnya tiga langkah -- baca, tambah satu, tulis kembali -- dan dua thread bisa berselang-seling di langkah-langkah itu.



Gambar 13.2 — Dua thread sama-sama membaca count sebagai 41 sebelum salah satu menulis kembali; keduanya menghitung 42 dan menulis 42. Salah satu dari dua increment tidak pernah terjadi, dan tidak ada apa pun dalam program yang memunculkan error untuk mengatakannya.

unsafe_borrow_counter.py -- implementasi naif yang buggy

```
class UnsafeBorrowCounter:
    def __init__(self) -> None:
        self._count = 0

    def increment(self) -> None:
        current = self._count
        time.sleep(0)  # melebarkan jendela race -- lihat catatan di bawah
        self._count = current + 1

    def get_count(self) -> int:
        return self._count
```

Catatan kejujuran: pengujian buku ini sendiri perlu melebarkan jendela untuk melihat race-nya

Sebuah self._count += 1 biasa, dijalankan lintas 20 thread dan 20 juta total increment selama pengujian buku ini sendiri, menghasilkan NOL update yang hilang pada mesin sandbox tempat buku ini ditulis -- interval pemeriksaan eval-loop CPython 3.12 sekadar tidak cukup sering jatuh tepat di dalam urutan baca/tambah/tulis pada lingkungan itu. Race condition-nya tetap sepenuhnya nyata (ini adalah perilaku CPython yang terdokumentasi, dikonfirmasi oleh Referensi [2] jalur Python di bawah, bukan klaim yang direka-reka untuk buku ini) -- ia hanya terlalu jarang untuk muncul secara andal dalam demo singkat pada satu mesin itu. time.sleep(0) di atas disisipkan secara sengaja, di antara pembacaan dan penulisan, untuk melebarkan jendela interleaving itu sehingga bahaya nyata yang SAMA menjadi terlihat sesuai permintaan, persis seperti kamera slow-motion mengungkap tabrakan nyata yang terjadi terlalu cepat untuk dilihat langsung. Ini TIDAK merekayasa bahaya yang sebelumnya tidak ada, dan diskusi Profitina di Bagian 13.9 sendiri menjelaskan mengapa bahaya yang mendasarinya tetap penting dalam kode produksi yang tidak memiliki penundaan artifisial semacam itu.

13.4 Sinkronisasi: Mutual Exclusion dengan threading.Lock

Sebuah objek threading.Lock hanya bisa diperoleh oleh satu thread pada satu waktu; sebuah blok with self._lock: memperoleh lock saat masuk dan melepaskannya secara otomatis saat keluar, bahkan jika sebuah exception dimunculkan di dalamnya. Selagi satu thread memegang lock itu, setiap thread lain yang mencoba memperoleh objek Lock yang SAMA akan terblokir -- ia hanya menunggu -- sampai lock itu dilepaskan. Itu mengubah 'baca, tambah satu, tulis kembali' menjadi operasi yang berperilaku atomik dari sudut pandang thread lain mana pun, meskipun di baliknya tetap tiga langkah pada level bytecode -- perbaikan yang sama dengan keyword synchronized pada jalur Java, hanya dieja sebagai objek eksplisit alih-alih keyword pada method itu sendiri.

safe_borrow_counter.py -- implementasi yang diperbaiki

```
class SafeBorrowCounter:
    def __init__(self) -> None:
        self._count = 0
        self._lock = threading.Lock()

    def increment(self) -> None:
        with self._lock:
            self._count += 1

    def get_count(self) -> int:
        with self._lock:
            return self._count
```

get_count() juga butuh lock, bukan cuma increment()

Mengunci hanya increment() akan menghentikan dua increment berselang-seling SATU SAMA LAIN, tapi thread yang memanggil get_count() selagi thread lain sedang di tengah increment tetap bisa membaca nilai yang sedang berubah -- bahaya yang lebih halus namun tetap nyata, disebut masalah visibility. Memperoleh Lock yang sama pada kedua method menutup kedua celah sekaligus.

13.5 Thread Pool: ThreadPoolExecutor

Membuat dan memulai satu objek threading.Thread per tugas, seperti yang dilakukan dua demo pertama Bagian 13.7, tidak dapat diskalakan melampaui segelintir tugas -- setiap thread membawa overhead memori dan penjadwalan OS yang nyata. concurrent.futures.ThreadPoolExecutor sebaliknya mengelola pool tetap dan bisa dipakai ulang dari thread pekerja: submit() menyerahkan sebuah tugas, dan thread pool mana pun yang paling dulu bebas akan mengambilnya, tanpa objek Thread baru dibuat per tugas (Gambar 13.3).

Mutual Exclusion, Dibandingkan: Dua Cara Menyatakan Hal yang Sama

Kedua bahasa memungkinkan Anda menandai "hanya satu thread boleh menjalankan ini pada satu waktu" -- mekanisme dan sintaksnya berbeda.

	Java	Python
Mendeklarasikan mutual exclusion	synchronized void increment() { ... } sebuah keyword pada method itu sendiri	with self._lock: self._count += 1 objek threading.Lock yang eksplisit
Apa yang dipegang selagi di dalam	Lock intrinsik (monitor) milik objek -- bawaan setiap objek	Objek Lock apa pun yang Anda buat dan pilih untuk diperoleh
Thread pool untuk banyak tugas	ExecutorService + Executors.newFixed-ThreadPool(n)	concurrent.futures.ThreadPoolExecutor(max_workers=n)

Tak satu pun mekanisme menghilangkan bahaya yang mendasarinya secara ajaib -- keduanya bekerja dengan membuat setiap thread LAIN menunggu gilirannya pada satu baris kode yang menyentuh nilai bersama, sehingga "baca, tambah satu, tulis kembali" menjadi efektif satu langkah dari sudut pandang thread lain mana pun.

Gambar 13.3 — Mutual exclusion dan thread pool, dibandingkan lintas kedua jalur yang dipakai mata kuliah ini.

Menyerahkan pekerjaan ke thread pool tetap

```
with ThreadPoolExecutor(max_workers=4) as pool:
    futures = [
        pool.submit(borrow_worker, counter, ITERATIONS)
        for _ in range(KIOSK_COUNT)
    ]
    for future in futures:
        future.result()
```

Blok with tidak kembali sampai setiap tugas yang diserahkan selesai

Keluar dari blok with ThreadPoolExecutor(...) as pool: memanggil pool.shutdown(wait=True) secara implisit, yang memblokir sampai setiap tugas yang sudah diserahkan selesai -- dan memanggil future.result() pada setiap Future juga memunculkan kembali (re-raise) exception apa pun yang dimunculkan tugas itu, alih-alih membiarkannya lenyap diam-diam di sebuah thread latar belakang.

13.6 Konkurensi dan Library: BorrowCounter sebagai Studi Kasus

BorrowCounter sengaja dibuat sebagai kelas kecil yang berdiri sendiri, bukan perubahan pada Library itu sendiri -- persis keputusan yang sama yang diambil Bab 11 ketika menambahkan get_all_items() alih-alih menulis ulang internal Library untuk sebuah GUI. Pelajaran yang diajarkan bab ini (state mutable bersama butuh sinkronisasi eksplisit begitu lebih dari satu thread bisa menjangkaunya) berlaku pada atribut Library sendiri persis sebanyak ia berlaku pada satu int milik BorrowCounter; BorrowCounter hanya mengisolasi pelajaran itu dari setiap perhatian lain yang sudah dicakup mata kuliah ini, sehingga Bagian 13.7 bisa mendemonstrasikannya secara terisolasi.

borrow_worker.py -- mensimulasikan satu kios patron

```
def borrow_worker(counter: BorrowCounter, iterations: int) -> None:
    for _ in range(iterations):
        counter.increment()
```

13.7 Program Driver: Tiga Demo, Berturut-turut

concurrency_demo.py mensimulasikan 8 kios patron. Demo 1 menjalankan UnsafeBorrowCounter dengan jumlah iterasi yang LEBIH KECIL (2.000 per kios) daripada Demo 2-3, karena setiap increment unsafe kini tidur sebentar agar race-nya terlihat andal (lihat catatan kejujuran Bagian 13.3) -- sehingga eksekusi unsafe pada beban kerja penuh 200.000-per-kios akan memakan waktu terlalu lama. Demo 2 menjalankan beban kerja penuh 200.000-per-kios terhadap SafeBorrowCounter dengan 8 Thread yang dibuat manual; Demo 3 menjalankannya sekali lagi terhadap SafeBorrowCounter, tapi lewat ThreadPoolExecutor 4-worker alih-alih 8 Thread yang dikelola manual.

concurrency_demo.py -- menjalankan demo dengan thread manual (diringkas)

```
def run_with_manual_threads(counter, iterations: int) -> int:
    kiosks = [
        threading.Thread(target=borrow_worker, args=(counter, iterations))
        for _ in range(KIOSK_COUNT)
    ]
    for kiosk in kiosks:
        kiosk.start()
    for kiosk in kiosks:
        kiosk.join()
    return counter.get_count()
```

start() memulai thread; memanggil fungsi target secara langsung tidak

kiosk.start() meminta sistem operasi untuk memulai thread eksekusi yang sungguh baru, yang kemudian memanggil fungsi target itu dengan sendirinya. Memanggil borrow_worker(counter, iterations) secara langsung, tanpa lewat Thread.start(), hanya akan mengeksekusi loop pada thread SAAT INI, berurutan, tanpa konkurensi sama sekali -- kesalahan umum yang mudah dilakukan dan menghasilkan kode yang berjalan bahkan menghasilkan angka yang benar, karena alasan yang salah.

13.8 Mengompilasi dan Menjalankan Program Anda

Python tidak memiliki langkah kompilasi, tetapi pelajaran Bab 11 tentang tkinter yang secara eksplisit membutuhkan python3.12 tidak berlaku di sini -- modul threading dan concurrent.futures bab ini adalah bagian dari pustaka standar pada versi Python 3 mana pun yang didukung. python3.12 tetap dipakai di bawah semata-mata demi konsistensi dengan jalur Python buku ini yang lain.

```
$ python3.12 concurrency_demo.py
```

13.9 OOP dalam Praktik: Profitina

> **Tentang kotak ini**
>
> Seperti di setiap bab, kotak ini menghubungkan materi kuliah dengan Profitina, platform AI-visibility (Generative Engine Optimization) Indonesia yang nyata dan aktif, dibangun oleh penulis -- menjelaskan DI MANA dan MENGAPA sebuah konsep digunakan dalam sistem nyata yang berjalan, tidak pernah logika bisnis atau kode sumber lengkap Profitina, yang tetap rahasia.

Backend Python/FastAPI Profitina melayani banyak permintaan konkuren sekaligus, dan beberapa counter miliknya sendiri -- rate limit, tally permintaan yang sedang berjalan, statistik cache hit/miss -- menghadapi persis bahaya bab ini: sebuah counter bersama yang naif diperbarui dari banyak thread penangan permintaan tanpa sinkronisasi bisa diam-diam under-count di bawah beban produksi nyata, bahaya mendasar yang sama yang didemonstrasikan UnsafeBorrowCounter di sini (dengan penundaan artifisial hanya karena pengujian buku ini sendiri kebetulan tidak memicunya secara alami dalam demo singkat -- kode produksi di bawah beban nyata, berkelanjutan, dan tak terduga tidak memiliki jaminan keamanan semacam itu). Perbaikan pada kode produksi adalah gagasan yang sama yang diajarkan bab ini pada skala yang jauh lebih kecil: baik penguncian eksplisit di sekitar state bersama, atau (lebih sering, pada skala Profitina) sebuah primitif aman-konkurensi khusus dari pustaka standar platform atau framework async-nya yang sudah menyelesaikan masalah persis ini sekali, dengan benar, sehingga fitur individual tidak perlu menyelesaikannya lagi.

13.10 Ringkasan Bab dan Poin-Poin Utama

- self._count += 1 sebenarnya tiga langkah -- baca, tambah satu, tulis kembali -- dan TIDAK atomik, bahkan di bawah GIL CPython, yang melindungi instruksi bytecode individual alih-alih satu pernyataan Python secara utuh.
- Pengujian bab ini sendiri menemukan race alaminya ternyata jarang pada satu mesin sandbox, dan mengungkap secara eksplisit mengapa serta perubahan artifisial apa (tidur singkat) yang dipakai agar bisa diamati secara andal -- lihat Bagian 13.3.
- Sebuah threading.Lock, diperoleh dengan blok with, menegakkan mutual exclusion: setiap thread lain yang mencoba memperoleh Lock yang SAMA terblokir sampai lock itu dilepaskan.
- Sebuah ThreadPoolExecutor menjalankan banyak tugas singkat lewat sekumpulan kecil thread pekerja tetap yang bisa dipakai ulang, alih-alih satu objek Thread per tugas -- dan memanggil .result() pada setiap Future memunculkan kembali exception apa pun yang dijumpai sebuah tugas.
- BorrowCounter mengisolasi pelajaran bab ini dari segala sesuatu yang lain yang sudah dilakukan Pustaka -- tapi bahaya yang sama berlaku pada state mutable bersama apa pun yang bisa dijangkau lebih dari satu thread, termasuk atribut Library sendiri.

Bug yang tidak pernah muncul di bawah satu thread, dan yang tidak dikomentari apa pun oleh linter mana pun, bukanlah bug yang tidak ada -- ia adalah bug yang sekadar belum pernah ditanyai satu pertanyaan yang mengungkapkannya.

13.11 Pertanyaan Review

Pertanyaan-pertanyaan ini untuk belajar mandiri dan diskusi kelas; tidak dinilai.

1. SafeBorrowCounter mengunci baik increment() maupun get_count(). Jelaskan, dengan kata-kata Anda sendiri, apa yang bisa salah jika hanya increment() yang memperoleh lock dan get_count() dibiarkan sebagai method biasa yang tidak dikunci.
2. Tulis ulang borrow_worker() sehingga, alih-alih memanggil counter.increment() dalam loop, ia membangun variabel total lokalnya sendiri dan hanya memanggil counter.increment() sekali di paling akhir (dalam loop dari 0 sampai total). Apakah versi ini masih bisa kehilangan update terhadap UnsafeBorrowCounter? Mengapa atau mengapa tidak?
3. Demo 3 concurrency_demo.py menggunakan ThreadPoolExecutor(max_workers=4) untuk 8 kios. Jelaskan apa yang terjadi pada tugas borrow_worker ke-5, ke-6, ke-7, dan ke-8 selagi keempat thread pool sibuk dengan 4 yang pertama.
4. Misalkan dua objek BorrowCounter yang berbeda (bukan objek yang sama) masing-masing diakses oleh thread khususnya sendiri, tanpa ada thread yang pernah menyentuh objek yang lain. Apakah Lock dibutuhkan di sini? Jelaskan mengapa atau mengapa tidak, dengan merujuk kembali pada apa yang sesungguhnya dibatasi oleh sebuah Lock.
5. Catatan kejujuran Bagian 13.3 menjelaskan bahwa race condition-nya tidak muncul dengan andal tanpa perubahan artifisial pada mesin pengujian buku ini. Jelaskan, dengan kata-kata Anda sendiri, mengapa race condition tetap bisa dianggap bahaya produksi yang nyata meskipun tidak muncul pada setiap eksekusi tunggal, atau bahkan pada setiap mesin.

Lampiran 13.A — Log Verifikasi Eksekusi

Berkas borrow_counter.py, unsafe_borrow_counter.py, safe_borrow_counter.py, borrow_worker.py, dan concurrency_demo.py yang lengkap (Code/Python/Chapter13/, disediakan bersama buku ini) dieksekusi pada python3.12 sebelum bab ini ditulis, untuk memastikan kode itu benar, bukan sekadar 'terlihat benar'. Output direproduksi verbatim di bawah ini dari satu eksekusi nyata.

Angka update-hilang yang persis akan berbeda jika Anda menjalankannya sendiri

Race condition, menurut definisinya, adalah soal timing -- menjalankan ulang concurrency_demo.py sangat mungkin menunjukkan angka update hilang yang BERBEDA dari eksekusi yang ditangkap di bawah ini. Yang TIDAK diharapkan berubah adalah hasil kualitatifnya: Demo 1 (unsafe) secara andal jatuh di bawah hitungan yang diharapkan di seluruh eksekusi berulang selama pengujian buku ini sendiri, sementara Demo 2 dan Demo 3 (keduanya safe) cocok dengan hitungan yang diharapkan persis, setiap saat.

```
$ python3.12 concurrency_demo.py
Simulating 8 patron kiosks.

--- Demo 1: UnsafeBorrowCounter (no synchronization) ---
(2000 borrow operations per kiosk -- fewer than Demos 2-3, since each unsafe
increment sleeps briefly to make the race visible; see
unsafe_borrow_counter.py.)
Mathematically correct final count: 16000
Actual final count:                 2012
Lost updates:                       13988  <-- a real race condition, not a typo

--- Demo 2: SafeBorrowCounter (threading.Lock) ---
(200000 borrow operations per kiosk.) Mathematically correct final count:
1600000
Actual final count:                 1600000
Matches expected:                   True

--- Demo 3: SafeBorrowCounter via a thread pool (ThreadPoolExecutor) ---
Actual final count:                 1600000
Matches expected:                   True
```

Referensi

[1] Python Software Foundation. "threading — Thread-based parallelism." Dokumentasi Python 3. https://docs.python.org/3/library/threading.html (diakses Agustus 2026).

[2] Python Software Foundation. "Thread State and the Global Interpreter Lock." Python/C API Reference Manual. https://docs.python.org/3/c-api/init.html#thread-state-and-the-global-interpreter-lock (diakses Agustus 2026).

[3] Python Software Foundation. "concurrent.futures — Launching parallel tasks." Dokumentasi Python 3. https://docs.python.org/3/library/concurrent.futures.html (diakses Agustus 2026).

BAB 14

Networking: Socket, Klien, dan Server

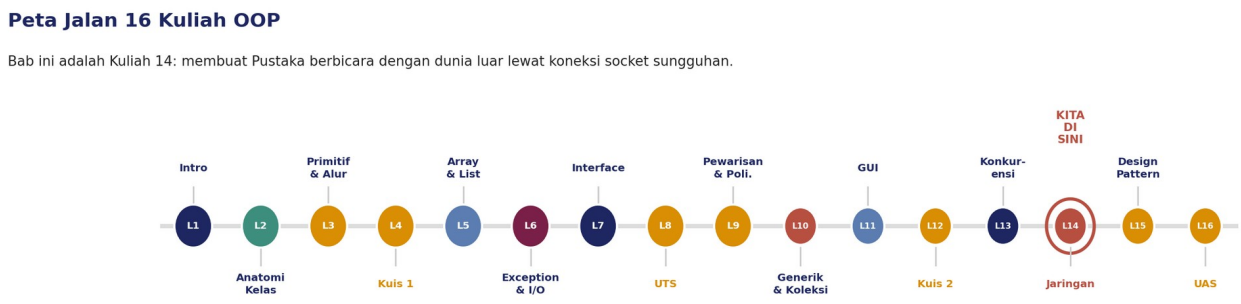
Setiap bab sejauh ini berjalan sepenuhnya pada satu mesin, hanya berbicara dengan dirinya sendiri. Bab ini membuat Pustaka berbicara dengan dunia luar: server TCP nyata yang mendengarkan pada sebuah port, klien TCP nyata yang terhubung kepadanya lewat jaringan, dan protokol request/response nyata yang berjalan di antara keduanya -- dibangun, secara sengaja, di atas threading.Thread yang persis sama yang diperkenalkan mata kuliah ini untuk alasan yang sama sekali berbeda di Bab 13 (Gambar 14.1).

Tujuan Pembelajaran — setelah bab ini Anda akan mampu:

(1) Menjelaskan model client-server: sebuah server yang mendengarkan pada port tetap, dan klien mana pun yang masing-masing membuka koneksinya sendiri kepadanya. (2) Membuka socket yang mendengarkan, meng-accept() koneksi masuk, dan membaca dari serta menulis ke data sebuah socket menggunakan makefile(). (3) Merancang protokol request/response berbasis baris yang minimal, dan mengimplementasikan baik sisi server maupun sisi klien-nya. (4) Menjelaskan mengapa menyerahkan setiap koneksi yang diterima ke threading.Thread-nya sendiri memungkinkan sebuah server melayani banyak klien sekaligus, dan menghubungkan keputusan itu secara eksplisit kembali ke pelajaran konkurensi Bab 13. (5) Membaca log eksekusi client-server nyata dan membedakan apa yang dijamin protokol dari apa yang kebetulan dihasilkan satu eksekusi tertentu.

14.1 Rekap: Pustaka Sejauh Ini

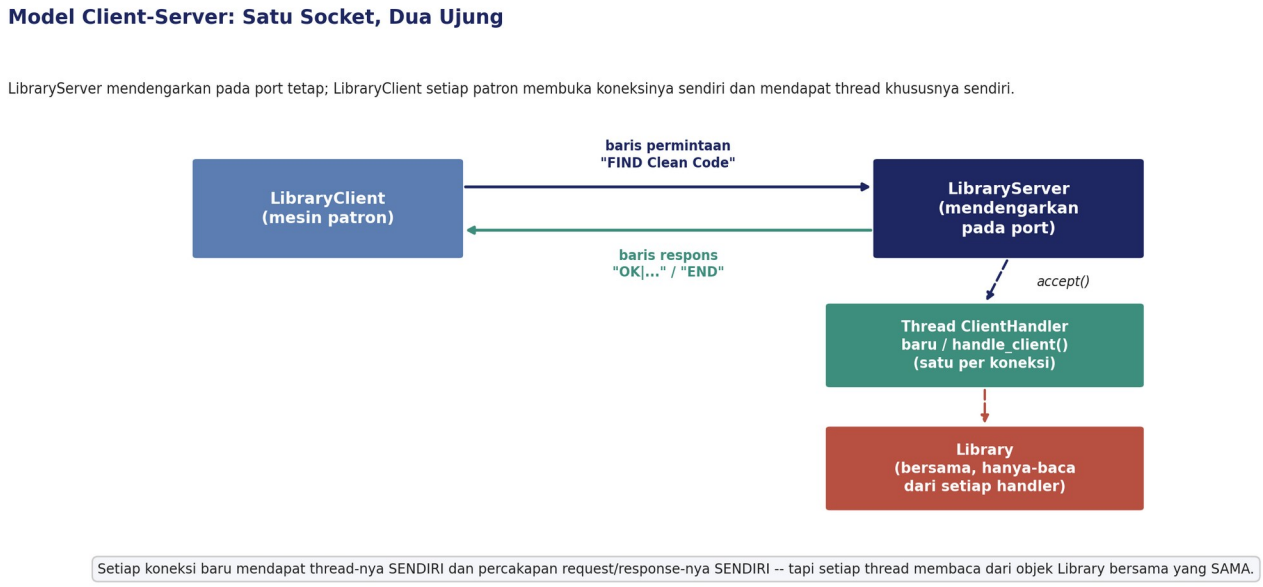
Bab 11 memberi Library antarmuka berjendela nyata. Bab 13 membuat kode terkait-Library berjalan pada lebih dari satu thread sekaligus, menggunakan studi kasus BorrowCounter yang kecil dan berdiri sendiri untuk mengungkap race condition nyata dan memperbaikinya dengan dua cara berbeda. Setiap bab itu, meski begitu, tetap berjalan pada satu mesin: proses mana pun yang menjalankan kode Pustaka adalah satu-satunya proses yang terlibat. Bab ini adalah pertama kalinya hal itu berhenti berlaku -- sebuah proses library_server.py dan sebuah proses library_client.py, berjalan sebagai dua program terpisah (berpotensi pada dua mesin terpisah), yang hanya pernah berbicara satu sama lain lewat koneksi jaringan.



Gambar 14.1 — Peta jalan 16-kuliah OOP. Bab ini adalah Kuliah 14: membuat Pustaka berbicara dengan dunia luar lewat koneksi socket sungguhan.

14.2 Model Client-Server

Server adalah program yang dimulai lebih dulu, membuka socket yang terikat pada port tetap, lalu menunggu: ia tidak tahu sebelumnya siapa yang akan terhubung, atau berapa banyak klien yang akan ada. Klien adalah program yang sudah tahu alamat dan port server, dan memulai koneksinya. Begitu koneksi terbentuk, sebuah objek socket yang terhubung ada pada KEDUA ujung -- server memegang satu per klien yang terhubung, dan klien memegang satu untuk koneksinya sendiri -- dan setiap sisi bisa membaca dari serta menulis ke socket itu persis seperti Bab 6 membaca dari dan menulis ke sebuah berkas (Gambar 14.2).



Gambar 14.2 — LibraryServer mendengarkan pada port tetap; LibraryClient setiap patron membuka koneksinya sendiri dan mendapat thread khususnya sendiri, tapi setiap thread membaca dari objek Library bersama yang sama.

> **Socket adalah jalan dua arah, bukan pipa satu arah**
>
> Socket terhubung yang sama yang dipakai klien untuk MENGIRIM baris permintaan juga menjadi objek tempat ia MEMBACA respons server -- memanggil makefile("w") dan makefile("r") padanya memberikan dua pandangan pada satu koneksi yang mendasarinya, bukan dua koneksi terpisah.

14.3 Membuka Socket yang Mendengarkan

library_server.py -- membuka socket dan menerima koneksi (diringkas)

```
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as server_socket:
    server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    server_socket.bind((HOST, PORT))
    server_socket.listen()
    print(f"LibraryServer listening on port {PORT}...")
    while True:
        conn, addr = server_socket.accept()   # memblokir sampai patron
terhubung
        client_thread = threading.Thread(
            target=handle_client, args=(conn, addr, library))
        client_thread.start()
```

accept() memblokir -- dan loop ini tidak pernah berakhir dengan sendirinya

server_socket.accept() tidak kembali sampai seorang klien benar-benar terhubung; thread pemanggil hanya menunggu. Loop while True: ini sengaja dan sesuai bagaimana server nyata berperilaku: ia terus menerima koneksi baru selama proses itu berjalan, dan biasanya dihentikan dari luar (Ctrl+C, atau process manager), bukan dari dalam kodenya sendiri.

14.4 Menangani Satu Klien: Protokol Berbasis Baris

Protokol Pustaka lewat kabel sengaja dibuat sederhana: klien mengirim tepat satu baris perintah, dan server mengirim kembali satu blok respons. FIND <judul> mencari satu item dan menjawab dengan satu baris OK|... atau ERROR|.... LIST menjawab dengan satu baris ITEM|... per item di perpustakaan, diikuti baris sentinel END sehingga klien tahu bloknya selesai. QUIT mengakhiri percakapan dengan baris BYE.

library_server.py -- satu thread khusus per koneksi (diringkas)

```
def handle_client(conn: socket.socket, addr, library: Library) -> None:
    with conn:
        reader = conn.makefile("r", encoding="utf-8", newline="\n")
        writer = conn.makefile("w", encoding="utf-8", newline="\n")
        for line in reader:
            request = line.rstrip("\n")
            if request.upper() == "LIST":
                for item in library.get_all_items():
                    writer.write(f"ITEM|{item.describe()}\n")
                writer.write("END\n")
                writer.flush()
            # cabang FIND dan QUIT dihilangkan -- lihat Code/Python/Chapter14/
```

conn.makefile(...) membungkus socket mentah dalam objek mirip-berkas

makefile("r", ...) dan makefile("w", ...) memberikan pembacaan dan penulisan mode-teks berbasis-baris lewat socket -- padanan Python untuk membungkus stream Socket Java dalam BufferedReader/PrintWriter -- dan blok with conn: menutup koneksi yang mendasarinya dengan bersih apa pun cara fungsi ini keluar, disiplin yang sama yang diajarkan Bab 6 untuk berkas, diterapkan di sini pada koneksi jaringan.

14.5 Satu Thread Per Klien: Networking Bertemu Bab 13

library_server.py menyerahkan setiap koneksi yang diterima ke threading.Thread yang benar-benar baru yang menjalankan handle_client(), lalu segera kembali ke accept() untuk koneksi BERIKUTNYA -- ia tidak pernah menunggu percakapan satu klien selesai sebelum menerima yang lain. Ini bukan materi baru; ini adalah threading.Thread Bab 13, diterapkan pada jenis pekerjaan konkuren yang sungguh baru. Yang BARU di sini adalah state bersama yang disentuh setiap thread itu: objek Library yang sama, dibaca (tidak pernah ditulis) dari setiap handler sekaligus. Ringkasan Bab pada Bagian 14.9 menjelaskan secara eksplisit mengapa pola khusus itu -- banyak pembaca, tanpa penulis, selagi server berjalan -- aman tanpa mesin Lock Bab 13 sama sekali, dan persis apa yang harus berubah agar itu berhenti benar (Gambar 14.3).

Socket, Dibandingkan: Dua Cara Menyatakan Hal yang Sama

Kedua bahasa mengekspos konsep socket TCP yang sama di baliknya -- nama kelas dan urutan pemanggilan persisnya berbeda.

	Java	Python
Membuka socket yang mendengarkan	new ServerSocket(port)	socket.socket(...) lalu .bind((host, port))
Menerima SATU koneksi	serverSocket.accept() mengembalikan Socket	server_socket.accept() mengembalikan (conn, addr)
Membaca/menulis teks lewatnya	BufferedReader / PrintWriter membungkus stream milik Socket	conn.makefile("r"/"w") membungkus socket mentah
Satu thread per klien	new Thread(new ClientHandler(...)) .start()	threading.Thread(target=handle_client, args=...).start()

Tak satu pun bahasa membuat koneksi socket andal dengan sendirinya -- keduanya tetap butuh disiplin try-with-resources / context-manager yang sudah diajarkan Bab 6 dan Bab 13, sehingga koneksi yang putus atau klien yang lenyap di tengah permintaan tetap menutup setiap stream dan socket dengan bersih, bukan membocorkannya.

Gambar 14.3 — API socket dibandingkan lintas kedua jalur yang dipakai mata kuliah ini.

State bersama hanya-baca tidak otomatis aman-thread selamanya -- hanya selama ia tetap hanya-baca

Setiap handler di sini hanya memanggil get_all_items() dan find_by_title() -- keduanya hanya MEMBACA field Library. Jika versi mendatang protokol ini pernah menambahkan perintah BORROW yang memanggil add_item() atau remove_item() dari dalam handle_client(), seluruh pelajaran Bab 13 akan berlaku lagi segera: banyak thread yang memutasi list dan dict yang sama tanpa sinkronisasi akan berisiko kehilangan update persis seperti yang didemonstrasikan BorrowCounter.

14.6 Sisi Klien: library_client.py

library_client.py -- mengirim satu perintah, membaca satu blok respons (diringkas)

```
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as sock:
    sock.connect((host, port))
    writer = sock.makefile("w", encoding="utf-8", newline="\n")
    reader = sock.makefile("r", encoding="utf-8", newline="\n")
    writer.write(command + "\n")
    writer.flush()   # tidak seperti PrintWriter(..., autoFlush=True) Java,
                     # writer makefile() Python butuh flush() eksplisit
    for line in reader:
        text = line.rstrip("\n")
        print(text)
        if text == "END" or text.startswith("OK|") \
           or text.startswith("ERROR|") or text == "BYE":
            break   # respons protokol sederhana ini selalu tepat satu blok
```

sock.connect((host, port)) adalah pemanggilan sisi-KLIEN

Berbeda dari socket yang mendengarkan, yang menunggu koneksi secara pasif, memanggil connect() secara aktif MEMULAI upaya koneksi ke alamat itu -- ia berhasil (pemanggilan kembali) atau memunculkan OSError (server tak terjangkau, menolak koneksi, atau tidak ada).

14.7 Mengompilasi dan Menjalankan: Server dan Klien, Dua Program Terpisah

Berbeda dari program driver tunggal setiap bab sebelumnya, demo bab ini benar-benar membutuhkan dua program berjalan PADA SAAT YANG SAMA: library_server.py harus sudah berjalan sebelum library_client.py bisa terhubung kepadanya. Tidak ada yang perlu dikompilasi di sini -- tapi kedua program, seperti berkas Python lain mana pun di jalur buku ini, dijalankan dengan python3.12 semata demi konsistensi dengan jalur Python yang lain.

```
$ python3.12 library_server.py &
LibraryServer listening on port 5052...
$ python3.12 library_client.py localhost 5052 LIST
$ python3.12 library_client.py localhost 5052 FIND Clean Code
```

14.8 OOP dalam Praktik: Profitina

Tentang kotak ini

Seperti di setiap bab, kotak ini menghubungkan materi kuliah dengan Profitina, platform AI-visibility (Generative Engine Optimization) Indonesia yang nyata dan aktif, dibangun oleh penulis -- menjelaskan DI MANA dan MENGAPA sebuah konsep digunakan dalam sistem nyata yang berjalan, tidak pernah logika bisnis atau kode sumber lengkap Profitina, yang tetap rahasia.

Arsitektur Profitina sendiri adalah sistem client-server pada skala yang jauh lebih besar: backend Python/FastAPI-nya memainkan persis peran library_server.py -- proses yang mendengarkan pada sebuah port dan menjawab permintaan dari banyak klien berbeda (browser, frontend-nya sendiri, tugas terjadwal) -- sementara setiap pemanggil itu memainkan peran library_client.py, membuka

koneksi, mengirim permintaan, dan membaca kembali respons. Protokol berbasis baris yang dirancang tangan di bab ini adalah, pada skala Profitina, HTTP dan JSON alih-alih FIND/LIST/QUIT -- tapi bentuk yang mendasarinya (alamat mendengarkan yang tetap, pertukaran request/response, satu handler per koneksi masuk) adalah pola persis yang diajarkan bab ini dari prinsip dasar.

14.9 Ringkasan Bab dan Poin-Poin Utama

- Server membuka socket yang mendengarkan terikat pada port tetap dan memanggil accept() dalam loop, yang memblokir sampai klien terhubung; klien memanggil connect((host, port)) untuk secara aktif memulai koneksi.
- Kedua ujung koneksi yang terbentuk membaca dari dan menulis ke data socket yang SAMA -- makefile("r") dan makefile("w") adalah dua pandangan pada satu koneksi dua-arah, bukan dua koneksi terpisah.
- Protokol bab ini sengaja dibuat sederhana: satu baris perintah masuk, satu blok respons keluar, dengan END eksplisit atau sentinel satu-baris sehingga klien tahu kapan respons selesai.
- library_server.py menyerahkan setiap koneksi yang diterima ke threading.Thread-nya sendiri -- alat yang sama yang diperkenalkan Bab 13, kini diterapkan pada I/O jaringan alih-alih kios yang disimulasikan -- sehingga banyak patron bisa dilayani pada saat yang sama.
- Setiap handler di sini hanya MEMBACA Library bersama; itulah persis mengapa belum diperlukan Lock -- begitu handler mana pun mulai menulis ke state bersama, seluruh pelajaran Bab 13 berlaku lagi.

Server yang hanya pernah bisa berbicara dengan satu klien pada satu waktu sebenarnya bukan server -- ia adalah program yang kebetulan menerima satu koneksi sekali.

14.10 Pertanyaan Review

Pertanyaan-pertanyaan ini untuk belajar mandiri dan diskusi kelas; tidak dinilai.

1. Jelaskan, dengan kata-kata Anda sendiri, mengapa server_socket.accept() harus dipanggil lagi segera di dalam loop while True:, alih-alih hanya sekali sebelum loop dimulai.
2. Misalkan handle_client() TIDAK membungkus koneksi dalam blok with conn:. Apa yang bisa salah jika klien terputus tiba-tiba di tengah respons LIST?
3. Protokol bab ini tidak punya cara bagi klien untuk memberi tahu server 'saya masih terhubung tapi tidak punya apa pun untuk dikatakan sekarang.' Jelaskan apa yang akan terjadi, pada sisi server, jika seorang klien terhubung lalu sama sekali tidak pernah mengirim baris apa pun.
4. Bagian 14.5 menjelaskan bahwa setiap handler hanya membaca Library, tidak pernah menulis kepadanya. Rancang (dengan kata-kata, bukan kode) sebuah perintah BORROW yang AKAN perlu menulis ke Library, dan jelaskan secara spesifik alat Bab 13 mana yang akan Anda gunakan agar aman.
5. library_client.py keluar setelah menerima tepat satu blok respons. Apakah membiarkan satu koneksi mengirim BANYAK perintah, satu demi satu, sebelum terputus, adalah perubahan sisi-klien atau sisi-server? Jelaskan kode sisi mana yang perlu berubah dan mengapa.

Lampiran 14.A — Log Verifikasi Eksekusi

Berkas book.py, dvd.py, item_not_found_error.py, library.py, library_demo.py, library_item.py, library_utils.py, pair.py, reference_book.py, renewable.py, library_server.py, dan library_client.py yang lengkap (Code/Python/Chapter14/, disediakan bersama buku ini) dieksekusi pada python3.12 sebelum bab ini ditulis -- dengan library_server.py berjalan sebagai proses yang benar-benar terpisah, dan beberapa pemanggilan library_client.py terhubung kepadanya lewat TCP/IP loopback nyata (bukan simulasi), termasuk dua klien yang terhubung PADA SAAT YANG SAMA untuk memastikan desain satu-thread-per-koneksi benar-benar melayani patron konkuren dengan benar. Output direproduksi verbatim di bawah ini dari satu eksekusi nyata.

```
$ python3.12 library_server.py &
LibraryServer listening on port 5052...

$ python3.12 library_client.py localhost 5052 LIST
ITEM|"Clean Code" [ISBN 9780132350884] - on the shelf
ITEM|"The Imitation Game" [ISBN 9900000000001] - on the shelf
ITEM|"Encyclopedia of Computer Science" [ISBN 9780123456789] - on the shelf
END

$ python3.12 library_client.py localhost 5052 FIND Clean Code
OK|"Clean Code" [ISBN 9780132350884] - on the shelf

$ python3.12 library_client.py localhost 5052 FIND Nonexistent Book
ERROR|Not found: Nonexistent Book

$ python3.12 library_client.py localhost 5052 QUIT
BYE
```

Dua klien yang terhubung pada instan yang sama sama-sama dilayani dengan benar

Pengujian bab ini sendiri juga meluncurkan dua proses library_client.py pada momen yang pada dasarnya sama (satu meminta FIND Clean Code, yang lain FIND The Imitation Game) terhadap satu library_server.py yang sedang berjalan; keduanya menerima respons yang benar, memastikan desain satu-thread-per-koneksi benar-benar melayani patron konkuren secara independen, bukan sekadar satu per satu dengan penyamaran.

Referensi

[1] Python Software Foundation. "socket — Low-level networking interface." Dokumentasi Python 3. https://docs.python.org/3/library/socket.html (diakses Agustus 2026).

[2] Python Software Foundation. "Socket Programming HOWTO." Dokumentasi Python 3. https://docs.python.org/3/howto/sockets.html (diakses Agustus 2026).

[3] Python Software Foundation. "threading — Thread-based parallelism." Dokumentasi Python 3, diakses Agustus 2026 (referensi silang Bab 13).

BAB 15

Design Pattern: Menamai Apa yang Sudah Dilakukan Pustaka

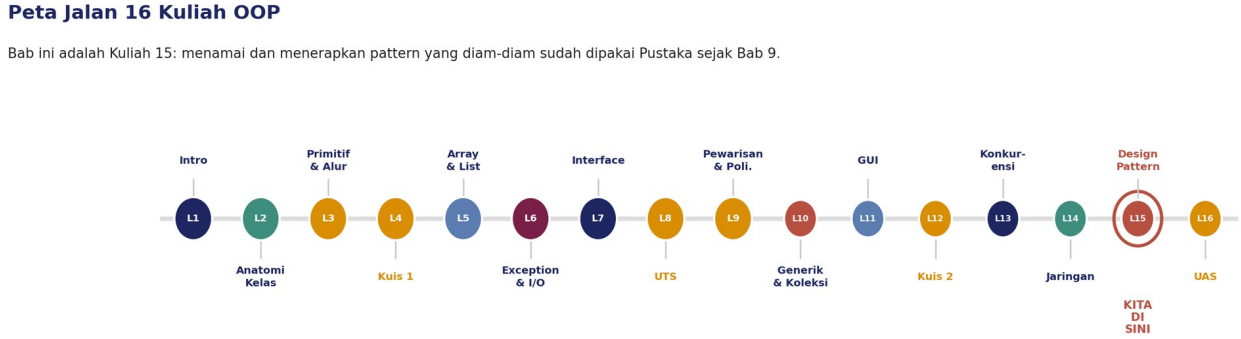
Setiap bab sejauh ini diam-diam meraih sebuah bentuk yang dapat dipakai ulang: rekonstruksi bertanda-tipe di Bab 9/10, View yang menggambar ulang dirinya sendiri ketika data Model berubah di Bab 11. Bab ini memberi tiga bentuk berulang itu namanya -- Factory Method, Observer, Singleton -- menerapkan dua di antaranya sebagai refactor Library sendiri yang nyata dan diungkapkan secara terbuka, dan memakai yang ketiga sebagai kisah peringatan tentang bahaya konkurensi yang sudah pernah ditemui mata kuliah ini, dalam penyamaran baru (Gambar 15.1).

Tujuan Pembelajaran — setelah bab ini Anda akan mampu:

(1) Menjelaskan apa itu design pattern, dan mengapa menamai sebuah bentuk struktural yang berulang memudahkan Anda mengenali, mendiskusikan, dan memakainya ulang secara sengaja. (2) Menerapkan pattern Factory Method untuk memusatkan logika rekonstruksi objek yang sebelumnya tersebar secara inline. (3) Menerapkan pattern Observer agar satu objek dapat mengumumkan perubahan state ke objek lain tanpa bergantung pada tipe konkretnya. (4) Menjelaskan mengapa inisialisasi lazy pada pattern Singleton naif adalah race condition yang nyata, dan mengapa inisialisasi eager menghilangkannya sepenuhnya, bukan sekadar mempersempitnya. (5) Membaca log eksekusi nyata yang mendemonstrasikan refactor design pattern yang memperbaiki bug yang sudah ada sebelumnya, dan log kedua yang mendemonstrasikan race condition yang direproduksi sesuai permintaan.

15.1 Rekap: Pustaka Sejauh Ini

Bab 13 memberi kode terkait-Library eksekusi konkuren yang aman; Bab 14 membuat Pustaka berbicara dengan dunia luar lewat koneksi socket nyata. Kedua bab itu, dengan caranya masing-masing, sudah memakai design pattern tanpa berhenti untuk menamainya: dispatch handle_client() Bab 14 adalah bentuk mirip-Command, dan pemisahan Model-View Bab 11 adalah relasi Observer buku teks dalam segala hal kecuali namanya. Bab ini berhenti meminjam pattern secara implisit dan mulai menerapkannya secara eksplisit -- pada Library yang SAMA yang telah dibangun mata kuliah ini sejak Bab 9, bukan pada contoh mainan baru.



Gambar 15.1 — Peta jalan 16-kuliah OOP. Bab ini adalah Kuliah 15: menamai dan menerapkan pattern yang diam-diam sudah dipakai Pustaka sejak Bab 9.

15.2 Mengapa Design Pattern?

Design pattern bukan library, framework, atau fitur bahasa -- ia adalah BENTUK bernama dan dapat dipakai ulang untuk menyelesaikan masalah desain yang berulang, tidak bergantung pada basis kode tertentu mana pun. "Factory Method" menamai bentuk "pusatkan kelas konkret mana yang dibangun, di satu tempat, alih-alih menyebarkan keputusan itu ke setiap pemanggil." "Observer" menamai bentuk "biarkan satu objek mengumumkan bahwa sesuatu berubah, tanpa objek itu perlu tahu siapa, jika ada, yang mendengarkan." Tak satu pun nama mengubah apa yang dilakukan kode; keduanya mengubah seberapa cepat dua developer bisa sepakat tentang APA sebuah potongan kode itu, begitu keduanya tahu nama pattern-nya. Bab ini secara sengaja menerapkan kedua pattern pada logika yang sudah dimiliki Library -- rekonstruksi tanda-tipe inline Bab 9/10, dan kebutuhan penyegaran-GUI implisit Bab 11 -- khusus agar "sebelum" dan "sesudah"-nya bisa dibandingkan secara langsung, alih-alih memperkenalkan pattern bersamaan dengan fungsionalitas yang benar-benar baru di mana perbandingannya akan lebih sulit dilihat.

> **Design pattern lebih sering dikenali belakangan daripada direncanakan lebih dulu**
>
> Katalog asli Gang of Four tahun 1994 sendiri disusun dengan mengamati bentuk-bentuk yang terus berulang di sistem berorientasi objek yang sudah bekerja, bukan dengan menciptakan bentuk lebih dulu lalu mencari penggunaannya belakangan. Pemisahan MVC Bab 11 sudah ada selama tiga bab sebelum bab ini pernah memakai kata "Observer" -- menamainya sekarang tidak mengubah kodenya; itu mengubah kosakata apa yang Anda miliki untuk mendiskusikannya.

15.3 Factory Method: Memusatkan "Kelas Mana yang Dibangun"

Sejak Bab 9/10, Library.load_from_file() membaca baris bertanda-tipe (BOOK|..., REFBOOK|..., DVD|...) dan merekonstruksi subtipe LibraryItem konkret yang cocok, secara inline, di dalam method privat _from_line(). Keputusan itu -- "diberikan tanda ini, kelas mana yang saya bangun?" -- persis yang dinamai pattern Factory Method: sebuah fungsi (atau kelas) khusus yang seluruh tugasnya adalah membangun objek dari tipe yang dipilih saat runtime, sehingga pemanggil LAIN mana pun yang kelak butuh keputusan yang sama bisa memakai ulang, bukan mengimplementasikan ulang tag-switch yang sama.

library_item_factory.py -- logika rekonstruksi, diekstrak (lengkap)

```
from book import Book
from dvd import DVD
from library_item import LibraryItem
from reference_book import ReferenceBook

def create_from_line(line: str) -> LibraryItem:
    parts = line.rstrip("\n").split("|")
    tag = parts[0]
    if tag == "BOOK":
        return Book(parts[1], parts[2], parts[3], int(parts[4]))
    elif tag == "REFBOOK":
        return ReferenceBook(parts[1], parts[2], parts[3], int(parts[4]))
    elif tag == "DVD":
        return DVD(parts[1], parts[2], int(parts[3]))
    raise ValueError(f"Unknown item type tag: {tag}")
```

Berbeda dari jalur Java, yang meniru bentuk kelas-utilitas-tanpa-state LibraryUtils (konstruktor privat, hanya method statik), jalur Python mengekspresikan Factory Method sebagai fungsi tingkat-modul biasa -- Python tidak perlu kelas sama sekali hanya untuk menampung fungsi tanpa state, dan memaksakan satu di sini hanya akan meniru sintaks Java, bukan maksudnya. Ini adalah REFACTOR dari perilaku yang sudah ada, bukan fitur baru: logika tag-switch itu sendiri tidak berubah dari Bab 9/10, hanya lokasinya yang berpindah. _to_line() -- serialisasi, arah sebaliknya -- sengaja tetap di dalam Library, karena Factory Method secara spesifik tentang PEMBUATAN; menulis kembali objek yang sudah ada ke sebuah baris adalah tanggung jawab yang berbeda, dan prinsip bab ini sendiri (satu modul, satu tanggung jawab yang jelas) menentang penggabungan keduanya hanya karena mereka adalah kebalikan satu sama lain.

> **Bug nyata yang terungkap oleh refactor ini: find_by_isbn() setelah reload**
>
> Mengekstrak create_from_line() mengharuskan penulisan ulang load_from_file() untuk memanggilnya, dan penulisan ulang load_from_file() mengungkap cacat nyata yang sebelumnya tidak terdeteksi, ada tanpa berubah sejak Bab 9/10: load_from_file() LAMA memanggil self._items.append(self._from_line(line)) secara langsung -- mengisi daftar items, tapi tak pernah dict _by_isbn. Setiap item yang pernah dimuat ulang dari berkas tersimpan karena itu tak terlihat oleh find_by_isbn(), meski find_by_title() bekerja dengan benar (menyembunyikan bug itu di setiap demo bab sebelumnya, yang tak satu pun kebetulan memanggil find_by_isbn pada item yang dimuat ulang). load_from_file() BARU kini menyalurkan setiap item yang direkonstruksi lewat add_item() -- dan add_item() sudah mengisi KEDUA koleksi. Lihat Lampiran 15.A untuk verifikasi sebelum/sesudah yang nyata atas perbaikan ini.

15.4 Observer: Mengumumkan Perubahan Tanpa Tahu Siapa yang Mendengarkan

GUI Bab 11 perlu menyegarkan daftar item yang ditampilkannya setiap kali koleksi Library yang mendasarinya berubah -- item ditambahkan atau dihapus -- tanpa Library pernah mengimpor apa pun yang terkait-GUI; Library tidak boleh tahu tkinter, atau teknologi presentasi spesifik mana pun, ada. Pattern Observer persis ini: Library memegang daftar pendengar yang memenuhi kontrak struktural yang sempit, dan memanggil method notifikasi masing-masing setelah setiap perubahan berhasil,

membiarkan setiap pendengar bebas melakukan apa pun yang diinginkannya dengan notifikasi itu -- mencatatnya ke konsol, menggambar ulang jendela, atau mengabaikannya sepenuhnya.

library_observer.py -- kontrak notifikasi (lengkap)

```
from typing import Protocol, runtime_checkable

from library_item import LibraryItem

@runtime_checkable
class LibraryObserver(Protocol):
    def on_item_added(self, item: LibraryItem) -> None: ...
    def on_item_removed(self, item: LibraryItem) -> None: ...
```

library.py -- mendaftarkan observer dan memberitahu mereka (diringkas)

```
def __init__(self) -> None:
    self._items: list[LibraryItem] = []
    self._by_isbn: dict[str, LibraryItem] = {}
    self._observers: list[LibraryObserver] = []

def add_observer(self, observer: LibraryObserver) -> None:
    self._observers.append(observer)

def _notify_item_added(self, item: LibraryItem) -> None:
    for observer in self._observers:
        observer.on_item_added(item)

def add_item(self, item: LibraryItem) -> None:
    self._items.append(item)
    self._by_isbn[item.isbn] = item
    self._notify_item_added(item)   # BARU pada bab ini
```

Karena typing.Protocol Python memberi LibraryObserver kontrak STRUKTURAL -- gaya sama yang sudah dipakai Renewable Bab 7 dan BorrowCounter Bab 13 -- ConsoleLogObserver memenuhinya cukup dengan mendefinisikan method yang cocok, tanpa pewarisan eksplisit atau kata kunci `implements` yang diperlukan, berbeda dari jalur Java yang memakai `class ConsoleLogObserver implements LibraryObserver`. Karena load_from_file() kini menyalurkan setiap item yang direkonstruksi lewat add_item() (Bagian 15.3), observer yang terdaftar dengan benar terpicu juga untuk item yang dimuat dari disk, tidak hanya item yang ditambahkan secara interaktif -- manfaat sampingan dari memperbaiki bug find_by_isbn dengan cara yang sama. Demo bab ini sendiri hanya mendaftarkan satu observer konkret, ConsoleLogObserver, yang sekadar mencetak baris ke konsol; presenter GUI di masa depan bisa mendaftarkan observer kedua yang menggambar ulang Listbox sebagai gantinya, dan kode Library sendiri tidak perlu berubah sama sekali untuk mendukungnya.

Observer adalah persis yang membuat desain Library terbuka untuk Bab 11 tanpa Bab 11 mengubah Library

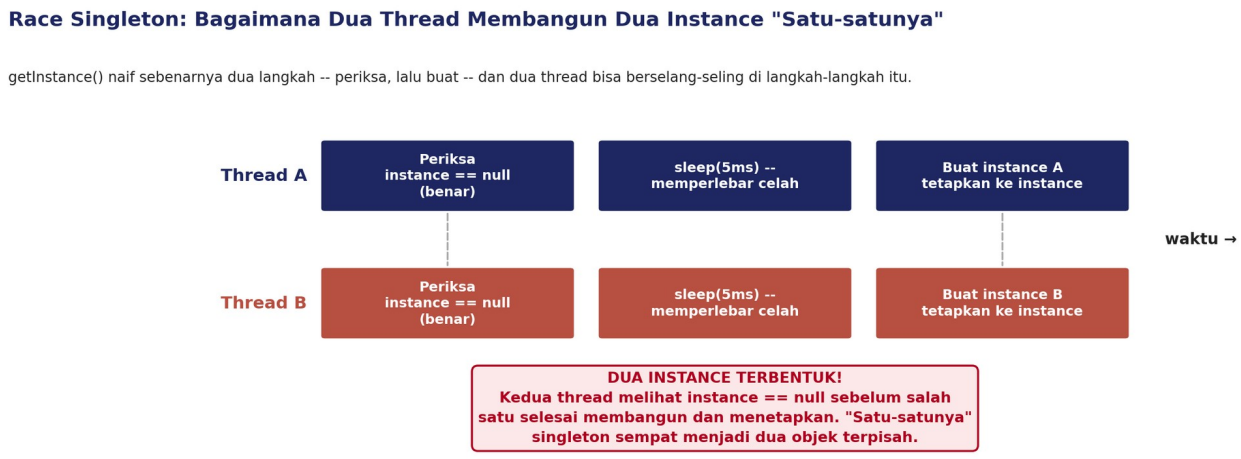
Ini adalah inti dari pattern-nya: LibraryObserver didefinisikan dalam istilah yang sudah dipahami Library (sebuah LibraryItem), sehingga Library bisa bergantung padanya tanpa bergantung pada BAGAIMANA observer tertentu bereaksi. Menambahkan jenis observer kesepuluh nanti tidak memerlukan perubahan apa pun pada add_item(), remove_item(), atau load_from_file() -- hanya kelas baru yang memenuhi Protocol LibraryObserver dan satu pemanggilan add_observer().

15.5 Singleton: Kisah Peringatan

Singleton menjanjikan "tepat satu instance kelas ini pernah ada, dan setiap orang yang memintanya mendapat yang sama." Implementasi naif dan lazy muncul di tak terhitung banyak tutorial: periksa apakah instance sudah ada, dan jika belum, buat. Urutan check-then-act itu, secara struktural, persis bahaya yang sama yang didemonstrasikan race lost-update Bab 13 -- kecuali di sini race-nya adalah atas PEMBUATAN objek, bukan increment sebuah counter, dan Global Interpreter Lock CPython TIDAK melindungi celah ini: sebuah thread bisa dialihkan tepat pada instan di antara pemeriksaan None dan penetapan atribut (Gambar 15.2, 15.3 dan 15.4).

naive_singleton.py -- versi klasik tanpa sinkronisasi (lengkap)

```
class NaiveSingleton:
    _instance: "NaiveSingleton | None" = None

    def __init__(self) -> None:
        pass

    @classmethod
    def get_instance(cls) -> "NaiveSingleton":
        if cls._instance is None:              # (1) periksa
            cls._instance = NaiveSingleton()    # (2) buat dan tetapkan
        return cls._instance
```



Gambar 15.2 — Dua thread sama-sama mengamati instance == null sebelum salah satu selesai membangun dan menetapkan: "satu-satunya" singleton sempat menjadi dua objek terpisah.

Catatan kejujuran: race ini persis tidak muncul dengan sendirinya dalam pengujian buku ini sendiri

Dua puluh thread yang berlomba melalui get_instance() dengan konstruktor tak dimodifikasi dan tanpa jeda hanya menghasilkan SATU instance berbeda, setiap kali, di lima eksekusi terpisah -- bahkan dengan starting gate threading.Barrier yang melepaskan kedua puluh thread sekaligus, overhead interpreter biasa saja rupanya cukup untuk menyerialisasi thread melewati celah check-then-act yang kecil di lingkungan ini. Ini kelas temuan yang sama yang diungkapkan Bab 13 untuk race Python-nya (loop increment biasa tidak konsisten kehilangan update tanpa jeda pelebar eksplisit), dikonfirmasi ulang secara independen di sini untuk bentuk bahaya yang berbeda. Sesuai disiplin kejujuran baku buku ini, temuan negatif itu diungkapkan di sini alih-alih diam-diam diakali: sumber naive_singleton.py sendiri membawa time.sleep(0.005) singkat di antara pemeriksaan dan pembuatan, melebarkan bahaya nyata yang sudah ada agar dapat diamati sesuai permintaan -- ia tidak mengarang bahaya yang belum ada di sana. Dengan perubahan itu diterapkan (dan starting gate threading.Barrier yang sudah ada), race-nya bereproduksi dengan andal di tiga eksekusi berulang: dua puluh instance NaiveSingleton berbeda, setiap kali. Lihat Lampiran 15.A untuk output nyata yang tertangkap.

safe_singleton.py -- inisialisasi eager menghilangkan race sepenuhnya (lengkap)

```
class SafeSingleton:
    _instance: "SafeSingleton"

    def __init__(self) -> None:
        pass

    @classmethod
    def get_instance(cls) -> "SafeSingleton":
        return cls._instance

SafeSingleton._instance = SafeSingleton()
```

Singleton, Dibandingkan: Naif vs. Aman

Kedua kelas mengekspos bentuk getInstance() yang sama -- yang berbeda hanya KAPAN instance tunggalnya dibangun.

	NaiveSingleton	SafeSingleton
Kapan instance dibangun	Malas (lazy) -- pertama kali getInstance() dipanggil (ada celah check-then-act)	Segera (eager) -- sekali, saat kelas itu sendiri dimuat/diimpor, sebelum thread mana pun memanggil getInstance()
Jendela race untuk dua thread	Ada -- keduanya bisa melihat "belum dibangun" sekaligus	Tidak ada -- tak ada pemeriksaan untuk diperlombakan sama sekali
Biaya jika tak pernah benar-benar dipakai	Tidak ada -- dibangun hanya saat pemakaian pertama	Satu instance tetap dibangun walau getInstance() tak pernah dipanggil

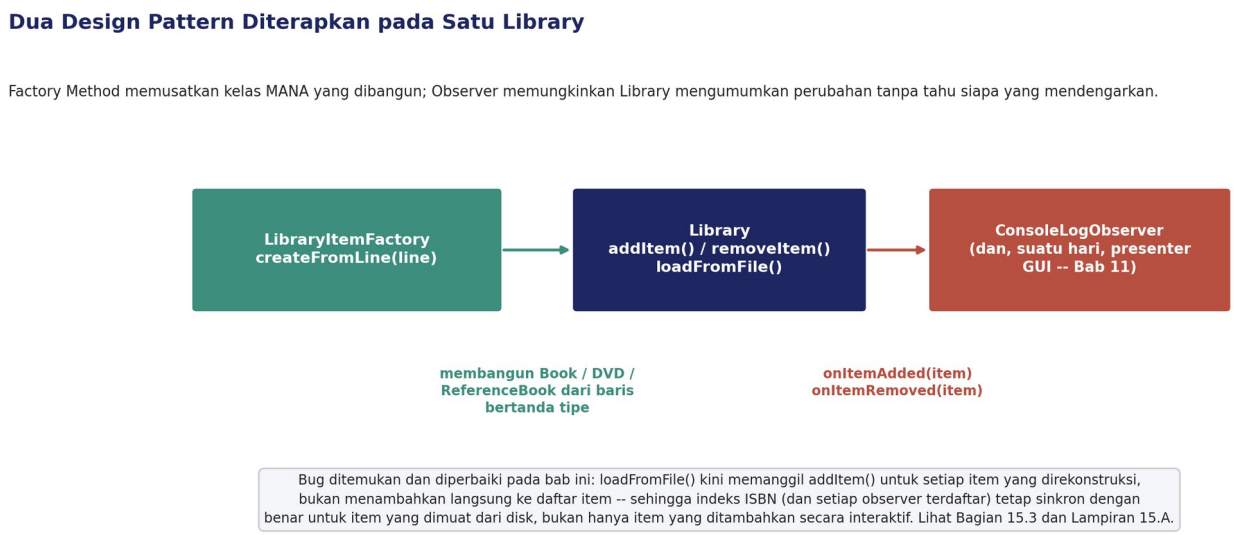
Ini kelas bahaya yang sama dengan race lost-update Bab 13 -- urutan check-then-act yang terlihat aman dibaca dari atas ke bawah, tapi tidak atomik. Perbaikannya di sini bersifat arsitektural (menghilangkan celahnya sama sekali), bukan menambal dengan synchronized/Lock.

Gambar 15.3 — NaiveSingleton vs. SafeSingleton: perbedaannya adalah KAPAN satu instance-nya dibangun, bukan bagaimana get_instance() terlihat dari luar.

Mengapa inisialisasi eager bebas-race, bukan sekadar kurang mungkin ber-race

Sistem impor modul Python menjamin kode tingkat-atas sebuah modul -- termasuk penetapan SafeSingleton._instance = SafeSingleton() tepat setelah badan kelasnya -- berjalan tepat sekali, pertama kali modul itu diimpor, sebelum thread mana pun di luar impor itu mungkin memanggil get_instance(). Tak ada celah check-then-act di sini untuk diselang-selingi dua thread; ini adalah perbaikan arsitektural, bukan tambalan Lock pada versi naif.

15.6 Design Pattern dan Library: Sebuah Studi Kasus



Gambar 15.4 — Factory Method dan Observer, keduanya diterapkan pada Library yang sama yang telah dibangun mata kuliah ini sejak Bab 9: library_item_factory memusatkan pembuatan, Library memberi tahu observer terdaftar pada setiap perubahan.

patterns_demo.py menjalankan kedua pattern yang diterapkan bersama-sama terhadap satu instance Library: tiga item ditambahkan (masing-masing memicu notifikasi Observer), satu dihapus (memicu notifikasi lain), library disimpan ke berkas dan dimuat ulang ke Library baru yang juga memiliki observer terdaftar (memicu notifikasi untuk setiap item yang dimuat dari disk, dan mendemonstrasikan rekonstruksi yang digerakkan Factory Method), dan akhirnya find_by_isbn() dipanggil pada item yang dimuat ulang untuk mengonfirmasi langsung perbaikan bug Bagian 15.3. Setiap program demo Bab 9 hingga Bab 14 SENDIRI -- library_demo.py khususnya -- dijalankan ulang tanpa modifikasi terhadap library.py baru bab ini dan menghasilkan output identik seperti sebelumnya, memastikan kedua refactor bab ini mengubah struktur internal Library tanpa mengubah perilaku yang dapat diamati untuk pemanggil mana pun yang sudah ada (lihat Lampiran 15.A).

15.7 Menjalankan Program

```
$ python3 -m py_compile *.py
$ python3 patterns_demo.py
$ python3 singleton_race_demo.py
$ python3 library_demo.py
```

15.8 OOP dalam Praktik: Profitina

> **Tentang kotak ini**
>
> Seperti di setiap bab, kotak ini menghubungkan materi kuliah dengan Profitina, platform AI-visibility (Generative Engine Optimization) Indonesia yang nyata dan aktif, dibangun oleh penulis -- menjelaskan DI MANA dan MENGAPA sebuah konsep digunakan dalam sistem nyata yang berjalan, tidak pernah logika bisnis atau kode sumber lengkap Profitina, yang tetap rahasia.

Basis kode Profitina sendiri memakai kedua pattern yang diterapkan bab ini secara nyata, pada skala produksi. Beberapa pipeline pemasukan-data-nya memusatkan "fungsi parser mana yang menangani format sumber ini" di belakang satu fungsi factory, bentuk yang sama seperti create_from_line() -- sehingga menambahkan dukungan untuk sumber data baru berarti menambahkan satu parser baru dan satu cabang factory baru, bukan memburu setiap titik pemanggilan yang dulu tahu formatnya secara langsung. Secara terpisah, pipeline event internal Profitina sendiri memungkinkan beberapa subsistem independen (logging, caching, notifikasi) bereaksi terhadap "sebuah laporan selesai dibuat" tanpa kode pembuatan-laporan mengimpor satu pun dari mereka secara langsung -- persis decoupling yang sama yang diberikan Observer kepada Library sehubungan dengan GUI Bab 11.

15.9 Ringkasan Bab dan Poin-Poin Utama

- Design pattern adalah bentuk bernama dan dapat dipakai ulang untuk masalah desain yang berulang -- menamainya tidak mengubah apa yang dilakukan kode, tapi membuat jauh lebih cepat bagi dua developer untuk sepakat tentang APA sebuah potongan kode itu.
- Factory Method memusatkan "kelas konkret mana yang saya bangun?" di satu tempat; bab ini menerapkannya dengan mengekstrak logika rekonstruksi tanda-tipe Library yang sudah ada ke library_item_factory.create_from_line(), refactor murni dari perilaku Bab 9/10.
- Refactor itu mengungkap dan memperbaiki bug nyata yang sebelumnya tidak terdeteksi: load_from_file() tak pernah memperbarui indeks _by_isbn, diam-diam merusak find_by_isbn() untuk item mana pun yang pernah dimuat ulang dari disk sejak Bab 9/10.
- Observer memungkinkan Library mengumumkan perubahan add_item()/remove_item() ke pendengar terdaftar tanpa tahu atau peduli apa yang dilakukan pendengar itu -- persis decoupling yang akan dibutuhkan presenter GUI Bab 11 di masa depan.
- Inisialisasi lazy check-then-act Singleton naif adalah race condition nyata, secara struktural identik dalam bentuk dengan race lost-update Bab 13 dan TIDAK dilindungi GIL; inisialisasi eager menghilangkan bahayanya secara arsitektural alih-alih mengunci di sekelilingnya.

Menamai sebuah pattern tidak dengan sendirinya membuat kode lebih baik -- menerapkannya untuk menghilangkan penyebaran logika duplikat yang nyata, atau kopling nyata yang seharusnya tidak ada, itulah yang membuat kode lebih baik; namanya hanya cara Anda dan developer lain sepakat tentang perbaikan mana yang baru saja Anda buat.

15.10 Pertanyaan Review

Pertanyaan-pertanyaan ini untuk belajar mandiri dan diskusi kelas; tidak dinilai.

1. Bagian 15.3 menyatakan _to_line() (serialisasi) tetap di dalam Library sementara create_from_line() (rekonstruksi) berpindah ke library_item_factory. Jelaskan, dengan kata-

kata Anda sendiri, mengapa kedua tanggung jawab ini TIDAK digabungkan ke modul yang sama meski satu adalah kebalikan persis dari yang lain.

2. Jelaskan, secara tepat, mengapa bug yang dijelaskan Bagian 15.3 tak terlihat di setiap program demo dari Bab 9 hingga Bab 14, dan pemanggilan spesifik mana yang akan mengungkapnya di bab-bab sebelumnya itu.
3. Misalkan LibraryObserver konkret kedua ditambahkan yang on_item_added()-nya melempar exception. Berdasarkan implementasi _notify_item_added() yang ditunjukkan Bagian 15.4, apa yang akan terjadi pada observer LAIN yang terdaftar, dan pada add_item() itu sendiri? Apakah ini desain yang baik, dan jika tidak, apa yang akan Anda ubah?
4. Bagian 15.5 mengungkapkan bahwa race Singleton naif tidak bereproduksi secara alami tanpa jeda tambahan, bahkan dengan starting gate threading.Barrier yang sudah ada. Jelaskan mengapa ini TIDAK berarti bahaya yang mendasarinya palsu atau aman diabaikan dalam kode nyata.
5. Rancang (dengan kata-kata, bukan kode) sebuah skenario di mana inisialisasi eager SafeSingleton akan menjadi pilihan yang benar-benar lebih buruk daripada inisialisasi lazy NaiveSingleton, meski ada race condition. Apa yang harus benar tentang konstruktor singleton itu agar trade-off itu penting?

Lampiran 15.A — Log Verifikasi Eksekusi

Berkas book.py, dvd.py, item_not_found_error.py, library.py, library_demo.py, library_item.py, library_utils.py, pair.py, reference_book.py, renewable.py, library_item_factory.py, library_observer.py, console_log_observer.py, naive_singleton.py, safe_singleton.py, singleton_race_demo.py, dan patterns_demo.py yang lengkap (Code/Python/Chapter15/, disediakan bersama buku ini) dieksekusi pada Python 3.12 sebelum bab ini ditulis. Output direproduksi verbatim di bawah ini dari eksekusi nyata.

```
$ python3 patterns_demo.py
--- Observer: notified on add_item() ---
[observer] added:   Clean Code
[observer] added:   The Imitation Game
[observer] added:   Encyclopedia of Computer Science

--- Observer: notified on remove_item() ---
[observer] removed: The Imitation Game

--- Factory Method: load_from_file() reconstructs items via library_item_factory
---
[observer] added:   Clean Code
[observer] added:   Encyclopedia of Computer Science

--- Bug found and fixed this chapter: find_by_isbn() after reload ---
reloaded.size()                          -> 2
reloaded.find_by_isbn("9780132350884")     -> Clean Code  (was None before this
chapter's refactor -- see Section 15.3)
```

Verifikasi sebelum/sesudah atas perbaikan bug find_by_isbn

Pengujian sekali-pakai yang dijalankan terhadap library.py LAMA (pra-Bab-15) mengonfirmasi bug itu secara empiris: setelah putaran simpan/muat, find_by_isbn("1111111111111") mengembalikan None sementara find_by_title("Test Book") dengan benar mengembalikan item. Pengujian yang identik dijalankan terhadap library.py BARU bab ini mengembalikan item yang benar dari find_by_isbn() setelah putaran yang sama -- memastikan penulisan ulang load_from_file() yang digerakkan Factory Method benar-benar memperbaiki cacat itu, bukan sekadar memindahkannya.

```
$ python3 singleton_race_demo.py
--- Demo 1: NaiveSingleton (unsynchronized lazy init) ---
Distinct NaiveSingleton instances observed: 20

--- Demo 2: SafeSingleton (eager init) ---
Distinct SafeSingleton instances observed:  1
```

Dua puluh thread, dua puluh instance NaiveSingleton berbeda -- setiap kali

Hasil persis ini (20 instance berbeda untuk NaiveSingleton, 1 untuk SafeSingleton) bereproduksi identik di tiga eksekusi berulang begitu pelebaran time.sleep(0.005) yang dijelaskan Bagian 15.5 diterapkan -- memastikan race-nya nyata dan dapat diamati dengan andal, bukan kebetulan sekali jadi, begitu jendela check-then-act diberi ruang untuk dijalankan.

Referensi

[1] Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley. (Katalog asli "Gang of Four" yang mendefinisikan Factory Method, Observer, Singleton, dan 20 pattern lainnya.)

[2] Python Software Foundation. "typing — Support for type hints" (Protocol, runtime_checkable). Python 3.12 documentation, diakses Agustus 2026 (referensi silang Bab 7/13).

[3] Python Software Foundation. "threading — Thread-based parallelism" (Barrier, Lock, Global Interpreter Lock). Python 3.12 documentation, diakses Agustus 2026 (referensi silang Bab 13; Barrier dipakai starting gate singleton_race_demo.py bab ini).

BAB 16 · BAB LATIHAN

UAS: Proyek Capstone Client-Server Berbasis GUI dan Socket

Tidak ada materi baru di sini — satu proyek terintegrasi yang menggabungkan pemrograman GUI, konkurensi, jaringan, dan design pattern menjadi satu aplikasi client-server yang bekerja secara nyata.

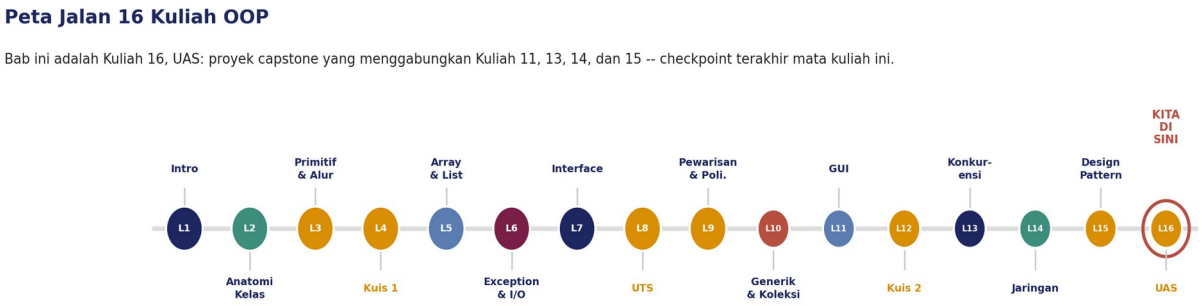
Tujuan Pembelajaran — setelah menyelesaikan bab ini, Anda akan mampu:

(1) Memutuskan dan menentukan cakupan topik aplikasi client-server, serta membagi pekerjaan dalam tim kecil dengan spesifikasi dan deskripsi tugas masing-masing anggota. (2) Menghasilkan diagram komponen yang mencakup Server, Client, konektivitas basis data, GUI, dan testing/dokumentasi, serta menulis dokumentasi interface untuk setiap komponen dalam pengertian umum — bukan berarti harus berupa Protocol atau ABC Python yang formal. (3) Merancang dan mendeskripsikan protokol komunikasi antara client dan server, termasuk flowchart, diilustrasikan dengan contoh nyata. (4) Merancang test case dua arah, di mana setiap sisi pasangan client-server merancang pengujian untuk fungsionalitas sisi lainnya. (5) Membawa proyek melalui fase Spesifikasi, Desain, dan Implementasi, dengan dokumentasi level pengguna, kriteria performa, instalasi, dan troubleshooting. (6) Mendeteksi dan menangani server yang down atau client yang down secara baik pada kedua sisi, menggunakan exception handling, bukan membiarkan salah satu proses crash tanpa keterangan.

16.1 Rekap: Kuliah 13–15 secara Ringkas

Kuliah 1 hingga 11 membangun perangkat inti bahasa dan sudah direview secara penuh oleh Kuis 2 di Bab 12: pergeseran dari pemikiran prosedural ke berorientasi objek, anatomi kelas dan enkapsulasi, primitif dan alur kendali, Kuis 1, list dan struktur data sederhana, exception dan file I/O, ABC dan Protocol, UTS, pewarisan dan polimorfisme, typing bergaya generik dan koleksi yang dibangun di atasnya, serta pemrograman GUI dengan Model-View-Controller. Kuliah 13 membawa Library melampaui satu thread eksekusi, memperkenalkan modul threading, Lock, dan race lost-update yang dapat muncul dari urutan read-modify-write yang naif di bawah akses konkuren — dengan GIL CPython diungkapkan secara jujur sebagai pelindung operasi bytecode tunggal, bukan urutan multi-langkah. Kuliah 14 membuka Library ke jaringan, membangun pasangan client-server di atas modul socket Python, dengan protokol eksplisit dan diungkapkan secara jujur yang mengatur apa yang boleh dikatakan setiap sisi dan kapan. Kuliah 15 menamai dua bentuk yang sudah dimiliki Library — Factory Method dan Observer, yang terakhir diekspresikan sebagai Protocol — dan menggunakan bentuk ketiga, Singleton, sebagai kisah peringatan tentang jenis race yang sama seperti Kuliah 13, kali ini pada konstruksi objek, bukan saldo rekening.

Bab final ini tidak menambahkan topik kelima yang baru. Sebagai gantinya, Anda diminta membangun satu aplikasi terintegrasi yang menggabungkan Kuliah 11, 13, 14, dan 15 sekaligus — sebuah sistem client-server berbasis GUI dan socket, dibangun dan dipertanggungjawabkan sebagai proyek capstone tim kecil (Gambar 16.1).



Gambar 16.1 — Bab ini berada di Kuliah 16, UAS, di ujung peta jalan 16 kuliah OOP: sebuah capstone, bukan topik baru.

16.2 Cara Menggunakan Bab Latihan Ini

Apa bab ini — dan apa yang bukan

Ini adalah Bab Latihan: bab praktik, bukan bab materi baru. Berbeda dari Bab 4, 8, dan 12, proyek final ini bukan sekumpulan soal terpisah dan independen — ini adalah SATU aplikasi capstone terintegrasi, yang komponen-komponennya dapat ditelusuri kembali ke empat kuliah tertentu (lihat Gambar 16.2). Panduan di bawah ini disusun berdasarkan aspek (memilih topik, mendokumentasikan protokol, merancang pengujian, menangani kegagalan), masing-masing dengan petunjuk menuju praktik yang baik, bukan solusi lengkap yang sudah dikerjakan atau kode sumber referensi. Bab ini, seperti Bab 4, 8, dan 12 sebelumnya, tidak menyertakan subfolder code/ — inti dari sebuah capstone adalah Anda merancang dan membangunnya sendiri.

Asal Setiap Komponen Capstone

Proyek akhir ini adalah satu aplikasi terintegrasi -- setiap komponen di Bagian 16.3 dapat ditelusuri ke salah satu dari empat kuliah.

#	Komponen Capstone	Berasal dari
1	Lapisan GUI	K11 -- Model-View-Controller, komponen Tkinter/PyQt berbasis event
2	Server Konkuren	K13 -- threading, lock, menghindari race lost-update
3	Protokol Client-Server	K14 -- modul socket, stream, protokol komunikasi yang diungkapkan secara jujur
4	Desain yang Dapat Digunakan Ulang	K15 -- Factory Method, Observer, Singleton diterapkan pada kelas Anda sendiri

Gambar 16.2 — Keempat komponen proyek capstone ini masing-masing dapat ditelusuri kembali ke Kuliah 11, 13, 14, atau 15.

Sebelum Anda mulai

Baca Bagian 16.3 secara penuh — termasuk panduan penanganan kegagalan — sebelum menulis satu baris kode pun atau membentuk tim Anda. Aplikasi client-server yang dirancang tanpa protokol dan rencana kegagalan sejak awal jauh lebih sulit diperbaiki belakangan, persis seperti peringatan Kuliah 14.

16.3 Proyek Capstone Final

Bekerjalah dalam tim beranggotakan maksimal tiga mahasiswa. Bangun aplikasi client-server berbasis GUI dan socket pada topik apa pun yang benar-benar diminati tim Anda — sebuah game multipemain kecil dengan chat, sebuah klon media sosial yang ringan, sebuah alat kolaborasi real-time, sebuah portal informasi interaktif, atau, sesuai semangat asesmen asli yang menjadi model bab ini, "apa pun yang menurut Anda keren." Topiknya jauh lebih tidak penting dibandingkan apakah tim Anda dapat mendemonstrasikan setiap keterampilan di bawah ini di atasnya.

16.3.1 Memilih Topik Capstone Anda

Pilih topik yang dapat tim Anda deskripsikan dalam satu kalimat, dan yang jelas membutuhkan client, server, dan suatu bentuk state persisten atau bersama — topik di mana program konsol satu proses jelas merupakan alat yang salah. Tentukan judul proyek dan pitch satu paragraf sebelum melanjutkan ke desain komponen.

> **Petunjuk**
>
> Topik yang menyenangkan untuk dideskripsikan belum tentu cocok secara otomatis — periksa dahulu apakah topik itu secara alami membutuhkan setidaknya dua proses independen yang berkomunikasi lewat socket, setidaknya satu GUI, dan suatu perilaku yang dapat gagal lewat jaringan. Jika ide Anda bekerja baik-baik saja sebagai satu program konsol bergaya Library, itu belum menjadi topik client-server.

16.3.2 Komponen Proyek & Pembagian Kerja

Hasilkan diagram komponen yang mengidentifikasi, minimal: Server, Client, konektivitas basis data atau persistensi, lapisan GUI, dan testing/dokumentasi sebagai komponen tersendiri — bukan sebagai renungan belakangan. Untuk tim beranggotakan dua atau tiga orang, tetapkan kepemilikan yang jelas atas satu atau lebih komponen kepada setiap anggota, dan minta setiap anggota menulis spesifikasi singkat dan deskripsi tugasnya sendiri untuk bagian yang dimilikinya.

> **"Spesifikasi dan dokumentasi interface" adalah istilah umum di sini**
>
> Dokumentasi interface tidak selalu berarti Protocol atau ABC Python yang formal. Artinya: untuk setiap komponen yang bergantung padanya anggota tim lain atau bagian sistem lain, tuliskan apa yang diharapkannya sebagai input, apa yang dijaminkannya sebagai output, dan apa yang dijamin (atau tidak dijamin) ketika terjadi kegagalan — dalam bentuk apa pun (dokumen singkat, sekumpulan docstring, sebuah tabel) yang mengomunikasikan hal ini dengan jelas kepada rekan tim yang belum membaca kode Anda.

16.3.3 Desain Protokol: Contoh "Ketuk-Ketuk"

Rancang dan deskripsikan, dengan flowchart, protokol komunikasi yang akan diikuti client dan server Anda — secara persis, sisi mana yang bicara lebih dulu, apa arti setiap pesan, dan bagaimana pertukaran itu berakhir. Deskripsi protokol yang hanya mendaftar jenis pesan, tanpa urutan, meninggalkan justru ambiguitas yang menyebabkan dua sisi yang dibangun secara independen mengalami deadlock atau salah memahami satu sama lain.

```
Server: "ketuk, ketuk"
Client: "siapa di sana?"
Server: "<seseorang>"
Client: "<seseorang> siapa?"
Server: "<sesuatu yang lucu>"
Server: "mau lagi?"
  -> Client menjawab "ya": pertukaran diulang dari awal
  -> Client menjawab "tidak": server menutup koneksi
```

Mengapa contoh sederhana ini penting

Setiap baris di atas tidak ambigu tentang giliran siapa yang berbicara dan apa arti sebuah jawaban -- persis properti yang dibutuhkan protokol Anda sendiri, apa pun topik Anda. Deskripsi protokol tanpa tingkat presisi ini belum menjadi protokol, hanya daftar nama pesan.

16.3.4 Desain Test Case Dua Arah

Karena client Anda menggunakan fungsionalitas server, dan server Anda, pada gilirannya, bergantung pada menerima permintaan yang terbentuk dengan baik dari client, desain test case harus berjalan dalam dua arah: anggota tim pemilik client sebaiknya merancang test case yang menguji perilaku server, dan anggota tim pemilik server sebaiknya merancang test case yang menguji perilaku client. Tidak ada sisi yang boleh hanya menguji kodenya sendiri secara terisolasi.

Rangkaian pengujian satu arah menyembunyikan bug nyata

Menguji hanya "apakah server saya merespons permintaan yang terbentuk dengan baik" tidak pernah menguji apa yang dilakukan server Anda terhadap permintaan yang cacat, pesan yang dikirim di luar urutan protokol, atau koneksi yang terputus di tengah pertukaran -- persis mode kegagalan yang diminta Bagian 16.3.6 untuk ditangani secara sengaja.

16.3.5 Fase Desain, Implementasi, dan Dokumentasi

Bawa proyek melalui tiga fase yang terlihat: Spesifikasi (apa yang harus dilakukan sistem, dan protokolnya), Desain (diagram komponen, tanggung jawab kelas, dan dokumentasi interface), dan Implementasi (kode yang benar-benar bekerja). Selain kode, hasilkan empat jenis dokumentasi: panduan level pengguna (cara menjalankan dan menggunakan aplikasi), pernyataan kriteria performa (apa arti "bekerja dengan benar" dan "bekerja dengan baik" untuk sistem Anda), panduan instalasi/cara menjalankan (termasuk paket apa saja yang harus diambil `pip install`), dan panduan troubleshooting yang mencakup mode kegagalan di Bagian 16.3.6.

Rencana pengujian, jamak

Rencana pengujian Anda sebaiknya mendeskripsikan lebih dari satu bentuk pengujian: pengujian berbasis stub (client atau server palsu yang menggantikan yang asli), pengujian multithreaded (beberapa client mengakses satu server secara konkuren -- lihat Kuliah 13), dan, jika memungkinkan bagi setup tim Anda, pengujian terdistribusi di lebih dari satu mesin.

16.3.6 Menangani Kegagalan: Server Down, Client Down

Aplikasi client-server yang hanya bekerja saat kedua sisi sehat dan jaringan sempurna belumlah selesai. Client Anda harus mendeteksi dan merespons secara masuk akal ketika server tidak dapat dijangkau atau down di tengah sesi -- bukan hang tanpa henti atau crash dengan ConnectionError yang

tidak ditangani. Secara simetris, server Anda harus mendeteksi dan merespons secara masuk akal ketika sebuah client terputus secara tak terduga -- bukan membocorkan thread atau sumber daya socket, dan tidak membuat sisa server crash bagi client lain yang masih terhubung.

> **Ini persis peringatan Kuliah 14, pada skala proyek tim**
>
> Kuliah 14 membangun client dan server Library dengan try/except eksplisit di sekitar setiap operasi socket, justru karena panggilan jaringan dapat gagal dengan cara yang tidak pernah dialami panggilan fungsi lokal. Proyek capstone yang melewatkan ini mengulangi persis kesalahan yang disebutkan namanya di Bagian 14.6 kuliah tersebut.

16.4 Format UAS: Proyek Tim, Open-Book

UAS adalah proyek tim beranggotakan maksimal tiga mahasiswa, open-book, dinilai berdasarkan deliverable yang dideskripsikan di Bagian 16.3 secara keseluruhan: diagram komponen Anda, spesifikasi dan dokumentasi interface, deskripsi protokol dan flowchart, test case dua arah, dokumentasi per fase, dan aplikasi client-server yang bekerja dan dapat didemonstrasikan dengan penanganan kegagalan yang disengaja dan diungkapkan secara jujur.

Integritas akademik

Dengan mengumpulkan proyek ini, setiap anggota tim menegaskan bahwa pekerjaan yang dikumpulkan adalah upaya genuin tim mereka sendiri, bahwa kode, pustaka, atau referensi eksternal apa pun yang digunakan diungkapkan secara jujur dan bukan disajikan sebagai karya asli, dan bahwa kontribusi yang diklaim masing-masing anggota secara akurat mencerminkan pekerjaan yang benar-benar mereka lakukan.

Kriteria	Yang dinilai	Bobot
Desain	Diagram komponen, spesifikasi dan dokumentasi interface, serta kejelasan protokol/flowchart.	20%
Implementasi	Aplikasi client-server yang bekerja sesuai desain, termasuk lapisan GUI, konkurensi, dan jaringan.	50%
Analisis & evaluasi	Test case dua arah, rencana pengujian (berbasis stub, multithreaded, terdistribusi jika memungkinkan), dan perilaku penanganan kegagalan yang diungkapkan secara jujur.	25%
Kesimpulan	Ringkasan yang jelas dan jujur tentang apa yang dibangun tim, apa yang berhasil, dan apa yang akan dilakukan tim secara berbeda dengan waktu lebih banyak.	5%

16.5 OOP dalam Praktik: Profitina

Tentang kotak ini

Seperti pada setiap bab, kotak ini menghubungkan materi kuliah dengan Profitina, sebuah platform AI-visibility (Generative Engine Optimization) Indonesia yang nyata dan aktif, dibangun oleh penulis — mendeskripsikan DI MANA dan MENGAPA sebuah konsep digunakan dalam sistem nyata yang berjalan, bukan logika bisnis atau kode sumber lengkap Profitina yang bersifat rahasia.

Profitina sendiri, pada intinya, persis berbentuk seperti bab ini pada skala produksi: sebuah client yang menghadap GUI berkomunikasi lewat jaringan dengan sebuah server yang memegang logika bisnis dan data yang sesungguhnya, dibangun dari komponen dengan spesifikasinya sendiri, diuji secara dua arah, dan direkayasa sejak hari pertama untuk mendeteksi dan pulih dari dependensi yang down, bukan mengasumsikan jaringan selalu sempurna. Proyek capstone yang berhasil menerapkan keempat unsur bab ini dengan benar — GUI, konkurensi, jaringan, dan beberapa design pattern yang dinamai dengan baik — telah, dalam skala mini, membangun bentuk sistem yang sama dengan yang dibangun secara profesional oleh penulis mata kuliah ini.

16.6 Ringkasan Bab

- Bab ini tidak memperkenalkan materi baru — ini adalah capstone yang menggabungkan Kuliah 11 (GUI & MVC), Kuliah 13 (Konkurensi), Kuliah 14 (Jaringan), dan Kuliah 15 (Design Pattern) menjadi satu proyek terintegrasi.
- Tim beranggotakan maksimal tiga mahasiswa merancang dan membangun aplikasi client-server berbasis GUI dan socket, pada topik pilihan mereka, didemonstrasikan dengan diagram komponen, dokumentasi interface, deskripsi protokol dengan flowchart, dan test case dua arah.
- Dokumentasi mencakup empat jenis: level pengguna, kriteria performa, instalasi, dan troubleshooting — di samping fase Spesifikasi, Desain, dan Implementasi.
- Penanganan kegagalan dinilai, bukan opsional: baik server down maupun client down harus dideteksi dan ditangani secara baik, pada kedua sisi, menggunakan exception handling yang diungkapkan secara jujur.
- Bobot penilaian memberi bobot terberat pada Implementasi (50%), diikuti Analisis & Evaluasi (25%), Desain (20%), dan Kesimpulan (5%).

Enam belas kuliah lalu, mata kuliah ini dimulai dengan satu pertanyaan: apa yang berubah ketika sebuah program diorganisasikan di sekitar objek, bukan prosedur? Setiap bab sejak itu telah menjadi satu jawaban lagi atas pertanyaan tersebut, dalam potongan yang semakin besar — dan proyek final ini adalah kali pertama semua potongan itu harus bekerja bersama, sekaligus, saling berkomunikasi lewat jaringan, persis seperti perangkat lunak yang sesungguhnya.

Lampiran 16.A — Daftar Periksa Mandiri (Tidak Dinilai)

Sebelum tim Anda mengumpulkan tugas, jalankan seluruh proyek melalui daftar periksa ini. Butuh waktu satu sore dan menangkap sebagian besar hal yang paling sering ditandai penilai pada capstone semacam ini.

- Apakah diagram komponen mencakup Server, Client, konektivitas basis data/persistensi, GUI, dan testing/dokumentasi, dengan setiap komponen dimiliki oleh anggota tim yang disebutkan namanya?
- Apakah ada spesifikasi tertulis untuk setiap komponen yang bergantung padanya anggota tim lain — input, output, dan jaminan kegagalan — bukan sekadar docstring pada sebuah signature fungsi?
- Apakah deskripsi protokol menyertakan flowchart, dan apakah secara tidak ambigu menyatakan giliran siapa yang berbicara di setiap langkah, seperti contoh Ketuk-Ketuk?
- Apakah pemilik client merancang test case untuk server, dan apakah pemilik server merancang test case untuk client — bukan hanya pengujian yang masing-masing tulis untuk kodenya sendiri?
- Apakah rencana pengujian mencakup pengujian berbasis stub, pengujian multithreaded dengan lebih dari satu client konkuren, dan (jika memungkinkan) pengujian terdistribusi di lebih dari satu mesin?
- Apakah client mendeteksi dan menangani server yang down atau down di tengah sesi, tanpa hang atau crash tanpa keterangan?
- Apakah server mendeteksi dan menangani client yang terputus secara tak terduga, tanpa membocorkan sumber daya atau memengaruhi client lain yang masih terhubung?
- Apakah keempat deliverable dokumentasi (level pengguna, performa, instalasi, troubleshooting) semuanya ada, dan apakah rekan tim yang baru bergabung hari ini dapat menjalankan sistem hanya dari dokumentasi tersebut?

Referensi

[1] Format asesmen dan rubrik penilaian bab ini dimodelkan secara dekat dari UAS nyata mata kuliah ini (EF234302 Pemrograman Berorientasi Objek): proyek tim beranggotakan maksimal tiga mahasiswa, open-book, membangun aplikasi client-server berbasis GUI dan socket pada topik pilihan tim, dinilai berdasarkan Desain (20%), Implementasi (50%), Analisis & Evaluasi (25%), dan Kesimpulan (5%). Berbeda dari UTS yang asli (lihat Referensi [1] Bab 8), yang konten pohon rekursif dan berbasis list-nya berada di luar cakupan OOP dari rebuild ini dan membutuhkan soal pengganti yang dirancang baru, konten wajib UAS yang asli — diagram komponen yang mencakup Server/Client/GUI/konektivitas basis data/testing, spesifikasi dan dokumentasi interface, protokol komunikasi yang diungkapkan secara jujur diilustrasikan dengan contoh Ketuk-Ketuk yang sama seperti di atas, desain test case dua arah, dokumentasi per fase, dan penanganan kegagalan server-down/client-down yang disengaja — sesuai secara langsung dengan persis apa yang benar-benar diajarkan rebuild ini di Kuliah 11, 13, 14, dan 15, dan digunakan kembali di sini sebagian besar apa adanya. Rincian administratif dari dokumen asli (alamat email pengumpulan, tanggal, konvensi penamaan berkas, dan Ikrar Integritas

Akademiknya sendiri yang bernuansa religius) telah digeneralisasi atau diparafrasakan untuk audiens buku ajar, bukan direproduksi kata demi kata.

[2] Python Software Foundation, "socket — Low-level networking interface," dokumentasi Python 3.12, diakses Agustus 2026.

[3] Python Software Foundation, "threading — Thread-based parallelism," dokumentasi Python 3.12, diakses Agustus 2026.