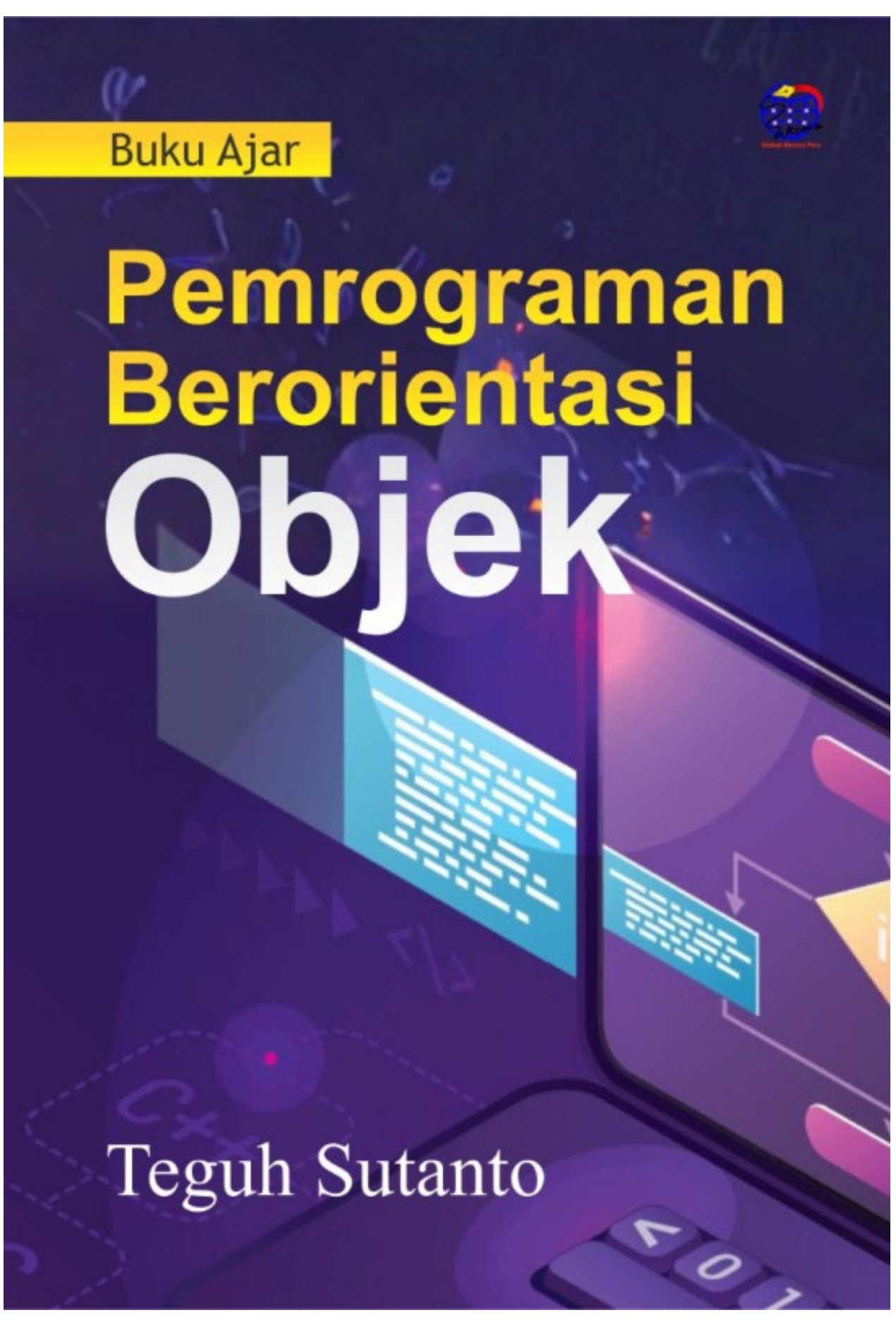
Buku Ajar

Pemrograman Berorientasi Objek



Teguh Sutanto

UNDANG-UNDANG REPUBLIK INDONESIA
NOMOR 28 TAHUN 2014
TENTANG HAK CIPTA

PASAL 113
KETENTUAN PIDANA
SANKSI PELANGGARAN

1. Setiap Orang yang dengan tanpa hak melakukan pelanggaran hak ekonomi sebagaimana dimaksud dalam Pasal 9 ayat (1) huruf i untuk Penggunaan Secara Komersial dipidana dengan pidana penjara paling lama 1 (satu) tahun dan/atau pidana denda paling banyak Rp100.000.000 (seratus juta rupiah).
2. Setiap Orang yang dengan tanpa hak dan/atau tanpa izin Pencipta atau pemegang Hak Cipta melakukan pelanggaran hak ekonomi Pencipta sebagaimana dimaksud dalam Pasal 9 ayat (1) huruf c, huruf d, huruf f, dan/atau huruf h untuk Penggunaan Secara Komersial dipidana dengan pidana penjara paling lama 3 (tiga) tahun dan/atau pidana denda paling banyak Rp500.000.000,00 (lima ratus juta rupiah).
3. Setiap Orang yang dengan tanpa hak dan/atau tanpa izin Pencipta atau pemegang Hak Cipta melakukan pelanggaran hak ekonomi Pencipta sebagaimana dimaksud dalam Pasal 9 ayat (1) huruf a, huruf b, huruf e, dan/atau huruf g untuk Penggunaan Secara Komersial dipidana dengan pidana penjara paling lama 4 (empat) tahun dan/atau pidana denda paling banyak Rp1.000.000.000,00 (satu miliar rupiah).
4. Setiap Orang yang memenuhi unsur sebagaimana dimaksud pada ayat (3) yang dilakukan dalam bentuk pembajakan, dipidana dengan pidana penjara paling lama 10 (sepuluh) tahun dan/atau pidana denda paling banyak Rp4.000.000.000,00 (empat miliar rupiah).

Teguh Sutanto

Buku Ajar
PEMROGRAMAN BERORIENTASI OBJEK

Global Aksara Pers

Buku Ajar
Pemrograman Berorientasi Objek

Diterbitkan pertama kali dalam bahasa Indonesia
oleh Penerbit Global Aksara Pers

ISBN: **978-623-462-991-0**
ix + 189 hal; 17,6 x 25 cm
Cetakan Pertama, Desember 2025

Penulis : Teguh Sutanto
Penyunting : Dr. Muhamad Basyrul Muvid, M.Pd
Desain Sampul : Hamim Thohari M
Layouter : Ilil N. Maghfiroh

Diterbitkan oleh:

CV. Global Aksara Pers
Anggota IKAPI, Jawa Timur, 2021,
No. 282/JTI/2021
Jl. Wonocolo Utara V/18 Surabaya
+628977416123/+628573269334
globalaksarapers@gmail.com

Kata Pengantar

Halo mahasiswa Indonesia yang luar biasa!

Selamat datang di dunia Pemrograman Berorientasi Objek (PBO) — sebuah paradigma yang tidak hanya mengubah cara kita menulis kode, tetapi juga cara kita memahami dan memodelkan dunia nyata ke dalam sistem digital.

Puji syukur Alhamdulillah penulis panjatkan ke hadirat Tuhan Yang Maha Esa atas segala rahmat dan karunia-Nya sehingga buku ajar yang berjudul " Pemrograman Berorientasi Objek " dapat disusun dan diselesaikan dengan baik.

Buku ini hadir sebagai panduan praktis dan konseptual bagi mahasiswa, dosen, maupun praktisi yang ingin memahami dan menguasai Pemrograman Berorientasi Objek (PBO) secara menyeluruh. Disusun secara sistematis dari konsep dasar hingga implementasi nyata, buku ini tidak hanya menjelaskan teori, tetapi juga dilengkapi dengan analogi kehidupan sehari-hari, studi kasus, latihan praktikum, dan worksheet interaktif dalam berbagai bahasa pemrograman seperti Python, Java, dan PHP.

Buku ini disusun dengan semangat untuk menjembatani kesenjangan antara teori dan praktik, antara kampus dan industri, antara konsep abstrak dan aplikasi nyata. Sebagai mahasiswa, Anda mungkin pernah merasa bahwa PBO itu rumit, membingungkan, atau bahkan “terlalu teknis”. Tapi percayalah, dengan pendekatan yang fun, naratif, dan aplikatif, Anda akan menemukan bahwa PBO bisa dipelajari dengan cara yang menyenangkan dan relevan.

Di dalam buku ini, Anda akan menemukan:

- Penjelasan konsep PBO dengan analogi kehidupan sehari-hari.
- Studi kasus nyata seperti sistem perpustakaan dan simulasi kendaraan.
- Latihan praktikum dan worksheet interaktif dalam Python, Java, dan PHP.
- Integrasi PBO dengan GUI, file handling, dan design pattern — semua dalam satu alur pembelajaran yang terstruktur.

Kami percaya bahwa mahasiswa bukan hanya pembelajar, tetapi juga calon inovator. Oleh karena itu, buku ini tidak hanya mengajarkan “cara membuat program”, tetapi juga “cara berpikir sebagai seorang software engineer”.

Terima kasih telah memilih buku ini sebagai bagian dari perjalanan akademik Anda. Semoga buku ini menjadi teman belajar yang menyenangkan,

membuka wawasan baru, dan mempersiapkan Anda untuk tantangan dunia teknologi yang terus berkembang.

Selamat belajar, bereksperimen, dan berkarya!

Surabaya, Oktober 2025
Teguh Sutanto
Penulis

Daftar Isi

BAB 1
Pendahuluan Pemrograman dan Evolusi OOP

Tujuan Pembelajaran

Sebelum kita masuk ke dunia pemrograman berorientasi objek (OOP), mari kita tetapkan dulu apa yang akan dicapai. Setelah menyelesaikan bab ini, Anda diharapkan mampu:

1. Menjelaskan bagaimana pemrograman berkembang dari era awal hingga sekarang.
2. Mengenali perbedaan paradigma pemrograman (prosedural, fungsional, dan OOP).
3. Memahami alasan mengapa OOP muncul dan menjadi sangat penting.
4. Menggambarkan konsep dasar OOP (kelas, objek, atribut, metode, konstruktor).
5. Mengaitkan OOP dengan kehidupan sehari-hari melalui analogi sederhana.

Kalau tujuan ini tercapai, maka kita sudah punya fondasi kokoh untuk mempelajari bab-bab berikutnya.

A. Sejarah Singkat Pemrograman: Dari Biner ke OOP

Mari kita mulai dengan perjalanan waktu. Bayangkan komputer pertama yang besar, berat, dan hanya mengerti bahasa biner: 0 dan 1. Programmer pada masa itu benar-benar menulis instruksi berupa deretan angka. Ibarat Anda harus memberi tahu teman cara membuat kopi dengan berkata: *"ambil 0011 gram kopi, masukkan ke 0101 air panas..."* — sulit, bukan?

Kemudian lahir **Assembly Language**, sedikit lebih manusiawi dengan simbol seperti MOV (pindahkan), ADD (tambah). Tapi tetap saja, itu seperti bahasa teknisi yang masih rumit bagi orang awam.

Lalu berkembang **bahasa tingkat tinggi** seperti FORTRAN, COBOL, dan BASIC. Ini sudah lebih mirip bahasa manusia, jadi programmer bisa fokus ke logika, bukan ke mesin.

Namun, saat aplikasi semakin kompleks, pendekatan prosedural mulai kewalahan. Bayangkan Anda menulis ribuan baris kode dalam satu file besar.

Seperti punya buku resep raksasa tanpa bab, tanpa daftar isi. Susah mencari, susah memperbaiki.

Di titik inilah muncul **Object Oriented Programming** (OOP). Bahasa seperti *Simula* dan *Smalltalk* memperkenalkan ide bahwa program bisa dimodelkan seperti dunia nyata: ada objek dengan sifat dan perilaku. Ide ini kemudian populer lewat C++, Java, hingga Python.

B. Paradigma Pemrograman: Cara Pandang yang Berbeda

Paradigma pemrograman bisa dianggap sebagai **cara pandang** dalam menulis program.

- **Tidak Terstruktur** → Paradigma dimana programmer dapat membuat program dengan tidak struktur dalam arti alur program bisa lompat sana dan lompat sini.
- **Paradigma Prosedural**: Fokus pada urutan langkah. Seperti resep masakan: potong bawang → tumis → masukkan garam. Cocok untuk masalah kecil dan linear.
- **Paradigma Fungsional** : Semua dipandang sebagai fungsi matematika. Ibarat mesin cuci: masukkan pakaian kotor, keluar pakaian bersih, tanpa efek samping di luar proses.
- **Paradigma Berorientasi Objek (OOP)**: Dunia dilihat sebagai kumpulan objek. Setiap objek punya identitas, atribut (data), dan metode (aksi). Ibarat kehidupan nyata: mobil punya warna, kecepatan, dan bisa berjalan atau berhenti.

Contoh sederhananya:

- Dalam prosedural, Anda menulis fungsi tambahKecepatan(mobil, 20).
- Dalam OOP, Anda cukup bilang mobil.tambahKecepatan(20).

Perbedaan kecil ini membawa dampak besar: kode jadi terasa lebih alami, seolah kita benar-benar “berbicara” dengan objek dalam program.

C. Mengapa OOP Lahir?

Bayangkan Anda membangun sistem perpustakaan besar. Awalnya, dengan pendekatan prosedural, Anda membuat fungsi tambahBuku(), hapusBuku(), pinjamBuku(). Semua data buku disimpan dalam satu daftar panjang (Lavin, 2006).

Masalah muncul saat sistem berkembang: ada buku digital, ada aturan denda baru, ada fitur pencarian canggih. Akhirnya, fungsi jadi bertambah panjang, daftar kode makin rumit, dan tiap perubahan bisa menimbulkan bug tak terduga.

OOP hadir sebagai solusi. Dengan OOP, Anda bisa membuat **kelas Buku**, yang punya atribut (judul, penulis, ISBN) dan metode (pinjam, kembalikan). Jika nanti ada **kelas Ebook**, Anda tinggal membuatnya sebagai turunan dari Buku, lalu menambahkan fitur khusus seperti download().

Dengan cara ini, perubahan tidak merusak sistem lain. Program tumbuh lebih teratur, ibarat taman yang diperluas dengan menambahkan petak baru, bukan dengan menumpuk tanaman sembarangan (Dennis, Wixom, & Tegarden, 2021).

D. Kelebihan Pemrograman Berorientasi Objek (OOP)

Bayangkan Anda sedang membangun sebuah kota. Kalau Anda mencoba membangun semuanya sekaligus—jalan, rumah, jembatan, sekolah—tanpa ada perencanaan modular, tentu akan sangat kacau. Akan lebih mudah kalau Anda membaginya menjadi bagian-bagian: blok perumahan, kawasan industri, jalan utama, fasilitas umum. Begitu pula dalam dunia pemrograman, OOP hadir untuk membantu kita memecah sistem yang rumit menjadi potongan-potongan kecil yang lebih mudah dipahami (Rangisetti, 2024).

Mari kita bahas satu per satu kelebihan OOP dengan contoh dan analogi yang dekat dengan kehidupan sehari-hari.

1. Modularitas: Kode seperti LEGO

Salah satu kekuatan utama OOP adalah **modularitas**. Program tidak lagi berupa satu "benang kusut" yang panjang, melainkan tersusun dari modul-modul kecil yang disebut kelas. Setiap kelas punya tanggung jawab tertentu, mirip balok LEGO dengan bentuk unik. Anda bisa menggabungkan balok-balok itu untuk membangun sesuatu yang besar, atau membongkarnya dan menyusunnya kembali dengan cara berbeda.

Contoh sederhana: dalam aplikasi e-commerce, kita bisa punya kelas User, kelas Product, dan kelas Order. Masing-masing berdiri sendiri, tapi juga bisa saling berinteraksi. Jika suatu hari aturan pemesanan berubah, Anda cukup mengutak-atik kelas Order saja tanpa harus mengacaukan kode User atau Product.

Dengan modularitas, pemrograman terasa lebih teratur, seperti punya laci-laci khusus untuk setiap jenis barang, bukan satu kardus besar berisi segalanya.

2. Enkapsulasi: Rahasia di Balik Kapsul

Pernahkah Anda minum obat kapsul? Anda tidak perlu tahu isi detailnya apa, yang penting kapsul itu bisa menyembuhkan. Konsep ini sama persis dengan **enkapsulasi** dalam PBO Sebuah objek menyimpan data (atribut) dan menyembunyikannya dari dunia luar. Orang lain hanya bisa berinteraksi lewat “pintu resmi” yaitu metode (fungsi dalam kelas).

Bayangkan sebuah kelas BankAccount. Saldo di dalam rekening tidak boleh sembarangan diubah. Kalau tidak ada enkapsulasi, siapa pun bisa menuliskan saldo = -1000, dan itu jelas tidak masuk akal. Dengan enkapsulasi, Anda harus melewati metode deposit() atau withdraw() yang sudah dilengkapi aturan dan validasi. Dengan begitu, data internal tetap aman, ibarat brankas yang hanya bisa dibuka dengan kunci tertentu.

3. Reusability: Cetak Biru yang Bisa Dipakai Lagi

Mengapa arsitek membuat cetak biru rumah? Karena cetak biru itu bisa dipakai berulang kali untuk membangun rumah lain, mungkin dengan sedikit variasi. Begitu pula OOP: Anda bisa menulis kode sekali lalu memakainya lagi di berbagai tempat.

Ada dua cara utama: **inheritance (pewarisan)** dan **composition (komposisi)**.

- Dengan inheritance, misalnya Anda punya kelas Vehicle lalu menurunkannya menjadi Car dan Motorcycle. Semua hal umum seperti kecepatan atau jumlah_roda sudah tersedia di Vehicle. Anda tinggal menambahkan hal-hal spesifik di kelas turunannya.
- Dengan composition, Anda merangkai kelas dari kelas lain. Contohnya, kelas Car punya objek Engine di dalamnya. Mesin bisa dipakai ulang di banyak jenis kendaraan tanpa harus menyalin ulang kodenya.

Reusability ini membuat hidup programmer lebih efisien—tidak perlu menulis ulang roda setiap kali mau membuat kendaraan baru.

4. Maintainability: Mudah Dirawat dan Diperbaiki

Bayangkan Anda memiliki sebuah gedung. Kalau instalasi listriknya dibuat modular, ketika ada kerusakan cukup memperbaiki bagian panel listrik tertentu, bukan membongkar seluruh bangunan. Prinsip serupa berlaku dalam OOP: kode lebih mudah dipelihara.

Misalnya, sistem perpustakaan awalnya menyimpan data buku dalam sebuah *list*. Setelah berkembang, Anda ingin menyimpannya di database. Kalau kelas Library sudah dirancang dengan baik, Anda cukup mengganti bagian internal tanpa mengubah cara pemanggilannya. Metode seperti add_book() atau find_book() tetap sama, sehingga kode lain yang menggunakan Library tidak perlu ikut diutak-atik.

Inilah mengapa maintainability sangat penting dalam proyek nyata: perubahan kebutuhan tidak bisa dihindari, dan OOP membantu kita beradaptasi tanpa stres.

5. Mapping Dunia Nyata: Kode yang Bicara Bahasa Manusia

OOP memungkinkan kita menulis kode yang “berbicara” dengan bahasa dunia nyata. Kalau Anda membuat aplikasi untuk bioskop, Anda bisa punya kelas Film, Tiket, Penonton. Stakeholder yang bukan programmer pun bisa paham maksudnya.

Misalnya, manajer bioskop berkata: “Tiket VIP harus bisa mengakses lounge khusus.” Anda bisa langsung menambahkan atribut atau metode akses_lounge() pada kelas TiketVIP. Tidak ada terjemahan rumit, karena kode sudah selaras dengan istilah sehari-hari.

Dengan cara ini, komunikasi antara programmer, manajer, dan pengguna akhir menjadi lebih mudah. OOP bukan sekadar teknik, tetapi juga jembatan antara dunia bisnis dan dunia teknologi.

E. Contoh Nyata: Sistem Perpustakaan

Mari kita buat gambaran sederhana.

- **Kelas Buku** → atribut: judul, penulis, ISBN; metode: pinjam(), kembalikan().
- **Kelas Anggota** → atribut: nama, nomor anggota; metode: pinjamBuku(), kembalikanBuku().
- **Kelas Peminjaman** → merekam hubungan antara anggota dan buku.

Kalau sistem bertambah rumit (misalnya menambahkan fitur e-book), kita bisa membuat Ebook sebagai subclass dari Buku. OOP memungkinkan ekspansi alami tanpa mengacak-acak kode lama.

F. Catatan Penting: OOP Bukan Obat Segalanya

Meski hebat, OOP bukan solusi untuk semua kasus.

- Kalau hanya butuh skrip kecil, OOP bisa terasa berlebihan.
- Terlalu banyak abstraksi bisa membuat kode malah lebih sulit dibaca.
- Dalam sistem sangat terbatas (misalnya mikrokontroler), overhead OOP bisa jadi masalah.

Jadi, gunakan OOP secara bijak: pakai ketika kompleksitas mulai meningkat, bukan untuk setiap program kecil.

Latihan & Praktikum

1. Sebutkan 3 perbedaan utama antara pemrograman prosedural dan OOP dengan contoh sederhana.
2. Pilih 5 objek nyata di sekitar Anda (misalnya sepeda, telepon, meja). Tentukan atribut dan metode yang bisa dimodelkan dengan PBO
3. Buat kelas Mobil sederhana dengan atribut merk, warna, dan kecepatan. Tambahkan metode jalan() dan rem(). Lalu buat dua objek mobil berbeda dan uji.

Resume

- Pemrograman berkembang dari bahasa mesin → assembly → bahasa tingkat tinggi → prosedural → PBO
- Paradigma pemrograman adalah cara pandang; OOP melihat dunia sebagai kumpulan objek.
- OOP lahir untuk mengatasi kompleksitas software modern.
- Kelebihan OOP: modularitas, enkapsulasi, reusability, maintainability, dan kemudahan memodelkan dunia nyata.
- Namun, OOP bukan selalu solusi terbaik untuk semua masalah.

Referensi

- Booch, Grady. *Object-Oriented Analysis and Design with Applications*. Addison-Wesley.
- Lafore, Robert. *Object-Oriented Programming in C++*. Sams Publishing.

- Rumbaugh, James et al. *Object-Oriented Modeling and Design*. Prentice Hall.

BAB 2
Konsep Dasar Pemrograman Berorientasi Objek

Tujuan Pembelajaran

Setelah mempelajari bab ini, Anda diharapkan mampu:

1. Menjelaskan konsep fundamental dalam OOP (kelas, objek, atribut, metode).
2. Memahami perbedaan antara kelas dan objek.
3. Membuat representasi sederhana dari dunia nyata ke dalam kode PBO
4. Menggunakan konstruktor untuk membangun objek.
5. Menerapkan contoh OOP dalam bahasa pemrograman (misalnya Python/Java/C++).

A. Mengapa Perlu Konsep Dasar?

Kalau kita analogikan OOP dengan kehidupan nyata, **kelas** adalah cetak biru (blueprint), sedangkan **objek** adalah wujud nyata dari cetak biru itu. Sama seperti arsitek yang menggambar rancangan rumah, lalu kontraktor membangun rumah asli berdasarkan gambar tersebut (Rangisetti, 2024).

Kalau kita tidak memahami apa itu kelas dan objek, sama saja seperti tukang bangunan yang tidak tahu cara membaca denah rumah—hasilnya pasti kacau.

B. Kelas (Class): Cetak Biru

Kelas adalah rancangan atau definisi dari sebuah objek (Sharan & Davis, 2022). Ia belum nyata, tapi berisi "aturan main" tentang apa saja yang dimiliki dan bisa dilakukan oleh objek.

Contoh sederhana:

- **Kelas Mobil** bisa punya atribut seperti warna, merk, kecepatan.
- Ia juga bisa punya metode (perilaku) seperti jalan(), rem(), klakson().

Tapi ingat, **kelas hanyalah rancangan**. Kalau kita hanya menulis class Mobil, belum ada mobil yang benar-benar bisa dipakai di jalan.

C. Objek (Object): Wujud Nyata

Objek adalah hasil konkret dari kelas (Prettyman, 2016). Jika kelas adalah cetak biru, maka objek adalah rumah yang sudah jadi. Kalau kelas adalah resep masakan, maka objek adalah masakan yang sudah siap disantap.

Contoh:

- Dari kelas Mobil, kita bisa buat objek mobil1 (warna merah, merk Toyota) dan mobil2 (warna hitam, merk Honda).
- Keduanya mengikuti cetak biru yang sama, tapi punya nilai atribut yang berbeda.

Dengan demikian, kelas itu general, sementara objek itu spesifik.

D. Atribut dan Metode

- **Atribut** adalah data yang dimiliki objek. Kalau bicara tentang manusia, atributnya bisa berupa nama, umur, tinggi badan.
- **Metode** adalah perilaku atau aksi yang bisa dilakukan. Misalnya, manusia bisa berjalan(), makan(), atau tidur().

Contoh dalam kode (misalnya Python):

```
class Mobil:
    def __init__(self, merk, warna):
        self.merk = merk      # atribut
        self.warna = warna    # atribut

    def jalan(self):          # metode
        print(f"{self.merk} berwarna {self.warna} sedang
berjalan...")

# membuat objek
mobil1 = Mobil("Toyota", "Merah")
mobil2 = Mobil("Honda", "Hitam")

mobil1.jalan()
mobil2.jalan()
```

Output:
Toyota berwarna Merah sedang berjalan...
Honda berwarna Hitam sedang berjalan...
Di sini, merk dan warna adalah atribut, sedangkan jalan() adalah metode.

E. Konstruktor: Pintu Masuk Objek

Ketika membuat objek, kita sering ingin langsung memberi nilai awal (misalnya merk dan warna mobil). Di sinilah peran **konstruktor**.

Dalam Python, konstruktor adalah metode khusus __init__(). Dalam Java/C++, konstruktor biasanya punya nama yang sama dengan kelas.

Contoh (Python):

```
class Mahasiswa:
     def __init__(self, nama, nim):
        self.nama = nama
        self.nim = nim

    def perkenalan(self):
        print(f"Halo, nama saya {self.nama} dengan NIM
{self.nim}")

mhs1 = Mahasiswa("Andi", "12345")
mhs2 = Mahasiswa("Budi", "67890")

mhs1.perkenalan()
mhs2.perkenalan()
```

Output:
Halo, nama saya Andi dengan NIM 12345
Halo, nama saya Budi dengan NIM 67890
Konstruktor memastikan setiap objek lahir dalam keadaan "lengkap" sesuai kebutuhan.

F. Analogi Kehidupan Sehari-hari

Agar lebih mudah dipahami, mari gunakan beberapa analogi:

- **Kelas = resep kue, Objek = kue yang sudah jadi.**
  Resep bisa dipakai berkali-kali untuk membuat kue dengan variasi berbeda.
- **Kelas = desain baju, Objek = baju yang dijahit.**
  Desainnya sama, tapi hasilnya bisa berbeda ukuran atau warna.
- **Kelas = blueprint mobil, Objek = mobil nyata di jalan.**
  Satu blueprint bisa melahirkan banyak mobil.

Latihan & Praktikum

Latihan Teori:

1. Jelaskan perbedaan kelas dan objek dengan analogi sehari-hari.
2. Sebutkan 3 contoh atribut dan metode untuk objek "Telepon Genggam".

Praktikum:

1. Buat kelas Hewan dengan atribut nama dan jenis. Tambahkan metode suara() yang menampilkan suara khas hewan.
2. Dari kelas Hewan, buat tiga objek berbeda (misalnya kucing, anjing, burung) dengan suara masing-masing.

Resume

- **Kelas** adalah cetak biru; **objek** adalah realisasi dari kelas.

- **Atribut** menyimpan data; **metode** menentukan perilaku.
- **Konstruktor** digunakan untuk memberi nilai awal pada objek.
- OOP memodelkan dunia nyata dengan cara yang intuitif, sehingga kode lebih mudah dipahami.

Referensi

- Lafore, Robert. *Object-Oriented Programming in C++*. Sams Publishing.
- Eckel, Bruce. *Thinking in Java*. Prentice Hall.
- Lutz, Mark. *Learning Python*. O'Reilly Media.

G. Atribut dan Metode

Contoh dalam Python

```
    class Mobil:
        def __init__(self, merk, warna):
            self.merk = merk      # atribut
            self.warna = warna    # atribut

        def jalan(self):          # metode
            print(f"{self.merk} berwarna {self.warna} sedang
    berjalan...")

    # membuat objek
    mobil1 = Mobil("Toyota", "Merah")
    mobil2 = Mobil("Honda", "Hitam")

    mobil1.jalan()
    mobil2.jalan()
```

Output:
Toyota berwarna Merah sedang berjalan...
Honda berwarna Hitam sedang berjalan...

Contoh dalam Java

```
class Mobil {
    String merk;
    String warna;

    // konstruktor
    Mobil(String merk, String warna) {
        this.merk = merk;
        this.warna = warna;
    }

    // metode
    void jalan() {
        System.out.println(merk + " berwarna " + warna + "
sedang berjalan...");
```

```
    }
}

public class Main {
    public static void main(String[] args) {
        Mobil mobil1 = new Mobil("Toyota", "Merah");
        Mobil mobil2 = new Mobil("Honda", "Hitam");

        mobil1.jalan();
        mobil2.jalan();
    }
}
```

Output:
Toyota berwarna Merah sedang berjalan...
Honda berwarna Hitam sedang berjalan...

Contoh dalam PHP

```
<?php
class Mobil {
    public $merk;
    public $warna;

    // konstruktor
    function __construct($merk, $warna) {
        $this->merk = $merk;
        $this->warna = $warna;
    }

    // metode
    function jalan() {
        echo $this->merk . " berwarna " . $this->warna .
" sedang berjalan...\n";
    }
}

$mobil1 = new Mobil("Toyota", "Merah");
$mobil2 = new Mobil("Honda", "Hitam");

$mobil1->jalan();
$mobil2->jalan();
?>
```

Output:
Toyota berwarna Merah sedang berjalan...
Honda berwarna Hitam sedang berjalan...

H. Konstruktor: Pintu Masuk Objek

Contoh dalam Python

```
class Mahasiswa:
    def __init__(self, nama, nim):
        self.nama = nama
        self.nim = nim

    def perkenalan(self):
        print(f"Halo, nama saya {self.nama} dengan NIM
{self.nim}")

mhs1 = Mahasiswa("Andi", "12345")
mhs2 = Mahasiswa("Budi", "67890")

mhs1.perkenalan()
mhs2.perkenalan()
```

Contoh dalam Java

```
class Mahasiswa {
    String nama;
    String nim;

    // konstruktor
    Mahasiswa(String nama, String nim) {
        this.nama = nama;
        this.nim = nim;
    }

    void perkenalan() {
        System.out.println("Halo, nama saya " + nama +
" dengan NIM " + nim);
    }
}
    •
public class Main {
    public static void main(String[] args) {
        Mahasiswa mhs1 = new Mahasiswa("Andi",
"12345");
        Mahasiswa mhs2 = new Mahasiswa("Budi",
"67890");

        mhs1.perkenalan();
        mhs2.perkenalan();
    }
}
```

Contoh dalam PHP

```
<?php
class Mahasiswa {
    public $nama;
    public $nim;

    // konstruktor
    function __construct($nama, $nim) {
        $this->nama = $nama;
        $this->nim = $nim;
    }

    function perkenalan() {
        echo "Halo, nama saya " . $this->nama . " dengan NIM "
. $this->nim . "\n";
    }
}

$mhs1 = new Mahasiswa("Andi", "12345");
$mhs2 = new Mahasiswa("Budi", "67890");

$mhs1->perkenalan();
$mhs2->perkenalan();
?>
```

Latihan Teori

1. Jelaskan perbedaan antara **kelas** dan **objek** menggunakan analogi sehari-hari.
2. Sebutkan minimal 3 atribut dan 3 metode untuk objek **Telepon Genggam**.
3. Apa fungsi konstruktor dalam OOP?

Latihan Praktikum

Soal 1

Buat kelas Hewan dengan atribut nama dan jenis. Tambahkan metode suara() yang menampilkan suara khas hewan. Buat 3 objek: **Kucing**, **Anjing**, dan **Burung** dengan suara masing-masing.

Solusi dalam Python

```
class Hewan:
    def __init__(self, nama, jenis, suara):
        self.nama = nama
        self.jenis = jenis
        self.suara = suara

    def suara_hewan(self):
        print(f"{self.nama} adalah seekor {self.jenis} dan
bersuara: {self.suara}")
```

```
# membuat objek
kucing = Hewan("Kitty", "Kucing", "Meong")
anjing = Hewan("Buddy", "Anjing", "Guk guk")
burung = Hewan("Tweety", "Burung", "Cuit cuit")

kucing.suara_hewan()
anjing.suara_hewan()
burung.suara_hewan()
```

Output:

```
Kitty adalah seekor Kucing dan bersuara: Meong
Buddy adalah seekor Anjing dan bersuara: Guk guk
Tweety adalah seekor Burung dan bersuara: Cuit cuit
```

Solusi dalam Java

```
class Hewan {
    String nama;
    String jenis;
    String suara;

    // konstruktor
    Hewan(String nama, String jenis, String suara) {
        this.nama = nama;
        this.jenis = jenis;
        this.suara = suara;
    }

    void suaraHewan() {
        System.out.println(nama + " adalah seekor " + jenis +
" dan bersuara: " + suara);
    }
}

public class Main {
    public static void main(String[] args) {
        Hewan kucing = new Hewan("Kitty", "Kucing", "Meong");
        Hewan anjing = new Hewan("Buddy", "Anjing", "Guk
guk");
        Hewan burung = new Hewan("Tweety", "Burung", "Cuit
cuit");

        kucing.suaraHewan();
        anjing.suaraHewan();
        burung.suaraHewan();
    }
}
```

Solusi dalam PHP

```
<?php
class Hewan {
    public $nama;
    public $jenis;
    public $suara;

    // konstruktor
    function __construct($nama, $jenis, $suara) {
        $this->nama = $nama;
        $this->jenis = $jenis;
        $this->suara = $suara;
    }

    function suaraHewan() {
        echo $this->nama . " adalah seekor " . $this->jenis .
" dan bersuara: " . $this->suara . "\n";
    }
}

$kucing = new Hewan("Kitty", "Kucing", "Meong");
$anjing = new Hewan("Buddy", "Anjing", "Guk guk");
$burung = new Hewan("Tweety", "Burung", "Cuit cuit");

$kucing->suaraHewan();
$anjing->suaraHewan();
$burung->suaraHewan();
?>
```

Latihan Tambahan

Soal 2

Buat kelas Mahasiswa dengan atribut nama dan nim. Tambahkan metode perkenalan() untuk menampilkan identitas mahasiswa. Buat 2 objek mahasiswa berbeda.
(Solusi bisa diadaptasi dari contoh pada subbab 2.5)

Resume Bab 2

- **Kelas** adalah blueprint; **objek** adalah hasil nyata dari blueprint.
- **Atribut** menyimpan data; **metode** adalah aksi/perilaku.
- **Konstruktor** digunakan sebagai pintu masuk saat objek dibuat.
- Contoh di Python, Java, dan PHP menunjukkan bahwa meskipun sintaks berbeda, konsep OOP tetap sama.
- Dengan latihan ini, mahasiswa diharapkan mampu menghubungkan teori dengan praktik di berbagai bahasa.

BAB 3
Empat Pilar Pemrograman Berorientasi Objek

Tujuan Pembelajaran
Setelah mempelajari bab ini, mahasiswa diharapkan mampu:
1. Menjelaskan empat pilar utama PBO
2. Memahami fungsi masing-masing pilar dalam membangun perangkat lunak.
3. Menerapkan konsep pilar OOP ke dalam kode Python, Java, dan PHP.
4. Menghubungkan konsep OOP dengan analogi kehidupan nyata.

A. Enkapsulasi — Menjaga Data, Seperti Brankas di Bank

Pernahkah Anda menabung di bank? Uang yang Anda titipkan tidak bisa sembarang orang ambil. Bahkan Anda sendiri pun tidak bisa langsung membuka brankas bank. Satu-satunya cara adalah lewat **mekanisme resmi**: tarik tunai lewat teller, ATM, atau aplikasi mobile banking. Bayangkan saldo rekening bank Anda. Apakah semua orang boleh tahu dan mengubah nilainya langsung? Tentu tidak. Saldo hanya bisa diakses lewat "pintu resmi" seperti ATM atau aplikasi mobile banking. Inilah konsep enkapsulasi: data penting disembunyikan dan hanya bisa diakses melalui metode yang sudah ditentukan.

Nah, konsep inilah yang disebut **enkapsulasi**. Dalam OOP, kita menyimpan data penting di dalam sebuah kelas dan **membatasi akses** terhadapnya. Tidak semua atribut boleh diakses langsung; kadang perlu metode khusus untuk membaca atau mengubahnya.

Enkapsulasi bermanfaat untuk:
1. **Keamanan data** — data tidak bisa sembarangan dimodifikasi.
2. **Kontrol penuh** — kita bisa menetapkan aturan bagaimana data diubah.
3. **Mudah perawatan** — jika suatu hari aturan berubah, cukup modifikasi metode aksesnya.

Contoh sederhana:
- Di kelas BankAccount, atribut saldo dibuat private (__saldo di Python, private saldo di Java/PHP).

- Untuk menambah saldo, kita tidak bisa langsung rekening.saldo += 500, melainkan harus lewat metode deposit(500).

Dengan begitu, kita bisa menambahkan aturan: misalnya deposit tidak boleh bernilai negatif.

Analogi lain: tubuh manusia. Anda tidak bisa langsung "mengganti" detak jantung. Yang bisa Anda lakukan adalah makan, olahraga, atau beristirahat. Tubuh sudah punya **mekanisme internal** untuk mengatur jantung.

Python

```
class BankAccount:
    def __init__(self, pemilik, saldo):
        self.pemilik = pemilik
        self.__saldo = saldo  # atribut private

    def deposit(self, jumlah):
        self.__saldo += jumlah

    def get_saldo(self):
        return self.__saldo

rekening = BankAccount("Andi", 1000)
rekening.deposit(500)
print(rekening.get_saldo())  # 1500
```

Java

```
class BankAccount {
    private String pemilik;
    private double saldo;

    public BankAccount(String pemilik, double saldo) {
        this.pemilik = pemilik;
        this.saldo = saldo;
    }

    public void deposit(double jumlah) {
        saldo += jumlah;
    }

    public double getSaldo() {
        return saldo;
    }
}

public class Main {
    public static void main(String[] args) {
        BankAccount rekening = new BankAccount("Andi", 1000);
        rekening.deposit(500);
        System.out.println(rekening.getSaldo()); // 1500
    }
}
```

PHP

```
<?php
class BankAccount {
    private $pemilik;
    private $saldo;

    public function __construct($pemilik, $saldo) {
        $this->pemilik = $pemilik;
        $this->saldo = $saldo;
    }

    public function deposit($jumlah) {
        $this->saldo += $jumlah;
    }

    public function getSaldo() {
        return $this->saldo;
    }
}

$rekening = new BankAccount("Andi", 1000);
$rekening->deposit(500);
echo $rekening->getSaldo(); // 1500
?>
```

B. Inheritance: Pewarisan Sifat

Dalam kehidupan nyata, anak bisa mewarisi sifat dari orang tua. Dalam OOP, kelas juga bisa mewarisi atribut dan metode dari kelas lain.

Coba pikirkan: mengapa anak sering mirip orang tua? Bisa dari warna mata, bentuk wajah, atau sifat. Dalam dunia OOP, fenomena ini disebut **inheritance** atau **pewarisan** (Sekulsoki, 2022).

Sebuah **kelas induk (parent class)** dapat mewariskan atribut dan metode ke **kelas anak (child class)**. Dengan begitu, kita tidak perlu menulis ulang kode yang sama berulang-ulang.

Contoh nyata:

- Kelas Hewan punya metode makan().
- Kelas Kucing dan Anjing adalah turunan dari Hewan. Mereka otomatis bisa "makan" tanpa kita tulis ulang metodenya.
- Tapi, masing-masing bisa menambahkan perilaku unik: suara() berbeda antara kucing (meong) dan anjing (guk-guk).

Analogi sehari-hari:

- Universitas punya "aturan umum" untuk semua mahasiswa (harus registrasi, punya NIM, bayar UKT).
- Tapi mahasiswa Fakultas Kedokteran mungkin punya tambahan aturan (praktek laboratorium wajib), dan mahasiswa Teknik punya aturan lain (praktikum bengkel).
- Semua tetap mahasiswa, tapi dengan variasi.

Inheritance membuat program lebih **terstruktur, hemat kode, dan mudah dikembangkan**.

C. Polimorfisme — Satu Nama, Banyak Wajah

Kata "poli" berarti banyak, dan "morph" berarti bentuk. Jadi, **polimorfisme** adalah kemampuan satu hal untuk memiliki banyak bentuk.
Misalnya, kata "lari":

- Atlet berlari di stadion.
- Aplikasi komputer "lari" di laptop.
- Waktu pun "lari" ketika Anda asyik menonton film.

Semua menggunakan kata yang sama: "lari", tapi maknanya berbeda tergantung konteks.

Dalam OOP, polimorfisme memungkinkan satu metode dengan nama sama tapi implementasi berbeda.

- Kelas Kucing dan Anjing sama-sama punya metode suara().
- Ketika dipanggil, hasilnya berbeda: kucing berkata "Meong", anjing berkata "Guk guk".
- Jika kita punya daftar hewan, cukup panggil suara() tanpa peduli hewan apa, program tahu cara menanganinya.

Analogi lain: tombol "play" di remote control.

- Tekan "play" di DVD → putar film.
- Tekan "play" di radio → putar musik.
- Tekan "play" di game console → jalankan game.

Satu tombol, banyak aksi.
Polimorfisme membuat kode lebih **fleksibel dan adaptif**.

D. Abstraksi — Fokus pada Inti, Sembunyikan Detail

Ketika Anda mengendarai mobil, apakah Anda tahu bagaimana mesin mengatur campuran bensin dan udara? Tidak perlu. Yang Anda tahu hanya: injak gas untuk jalan, injak rem untuk berhenti, putar setir untuk belok.

Inilah **abstraksi**: kita **menyembunyikan detail teknis** yang rumit, hanya memperlihatkan bagian penting yang relevan (Rangisetti, 2024).

Dalam OOP, abstraksi biasanya diwujudkan dengan **kelas abstrak** atau **interface**. Kelas abstrak hanya memberi **kerangka (kontrak)**:

- Misalnya kelas Kendaraan punya metode bergerak().
- Tapi cara bergeraknya tidak didefinisikan di kelas Kendaraan. Itu tanggung jawab kelas turunan: Mobil bergerak di jalan, Pesawat bergerak di udara.

Analogi nyata:

- Remote TV punya tombol "power", "volume", "channel".
- Anda tidak perlu tahu bagaimana gelombang inframerah bekerja di dalamnya. Yang penting, setiap remote wajib punya tombol dasar itu (abstraksi).

Abstraksi membantu programmer fokus pada **"apa yang dilakukan"**, bukan **"bagaimana dilakukan"**.

Python

```
class Hewan:
    def __init__(self, nama):
        self.nama = nama

    def makan(self):
        print(self.nama + " sedang makan.")

class Kucing(Hewan):  # Kucing mewarisi Hewan
    def suara(self):
        print("Meong!")

kucing = Kucing("Kitty")
kucing.makan()
kucing.suara()
 Java
class Hewan {
    String nama;

    Hewan(String nama) {
        this.nama = nama;
    }

    void makan() {
        System.out.println(nama + " sedang makan.");
    }
}

class Kucing extends Hewan {
    Kucing(String nama) {
        super(nama);
    }

    void suara() {
        System.out.println("Meong!");
    }
}

public class Main {
    public static void main(String[] args) {
        Kucing kitty = new Kucing("Kitty");
        kitty.makan();
        kitty.suara();
    }
}
```

PHP

```php
<?php
class Hewan {
    public $nama;

    public function __construct($nama) {
        $this->nama = $nama;
    }

    public function makan() {
        echo $this->nama . " sedang makan.\n";
    }
}

class Kucing extends Hewan {
    public function suara() {
        echo "Meong!\n";
    }
}

$kitty = new Kucing("Kitty");
$kitty->makan();
$kitty->suara();
?>
```

E. Polimorfisme: Satu Nama, Banyak Bentuk

Polimorfisme memungkinkan satu metode punya bentuk berbeda tergantung konteks. Misalnya, kata “lari” bisa berarti atlet berlari di stadion, atau aplikasi sedang “lari” di komputer.

Python

```python
class Hewan:
    def suara(self):
        print("Hewan bersuara...")

class Kucing(Hewan):
    def suara(self):
        print("Meong!")

class Anjing(Hewan):
    def suara(self):
        print("Guk guk!")

hewan_list = [Kucing(), Anjing()]
for h in hewan_list:
    h.suara()
```

Java

```
class Hewan {
    void suara() {
        System.out.println("Hewan bersuara...");
    }
}

class Kucing extends Hewan {
    void suara() {
        System.out.println("Meong!");
    }
}

class Anjing extends Hewan {
    void suara() {
        System.out.println("Guk guk!");
    }
}

public class Main {
    public static void main(String[] args) {
        Hewan[] hewan = { new Kucing(), new Anjing() };
        for (Hewan h : hewan) {
            h.suara();
        }
    }
}
```

PHP

```
<?php
class Hewan {
    public function suara() {
        echo "Hewan bersuara...\n";
    }
}

class Kucing extends Hewan {
    public function suara() {
        echo "Meong!\n";
    }
}

class Anjing extends Hewan {
    public function suara() {
        echo "Guk guk!\n";
    }
}

$hewan = [new Kucing(), new Anjing()];
foreach ($hewan as $h) {
```

```
    $h->suara();
}
?>
```

F. Abstraksi: Fokus pada Inti

Abstraksi berarti kita menyembunyikan detail teknis dan hanya menampilkan hal-hal penting. Sama seperti Anda mengendarai mobil: cukup tahu gas, rem, dan setir—tidak perlu pusing bagaimana mesin bekerja.Abstraksi juga dapat diartikan sebagai simplikasi atau penyerhanaan dari sesuatu yang komplek dibuat bentuk sederhana namun tetap bias mewakili entitas tersebut.

Java (contoh dengan abstract class)

```
abstract class Bentuk {
    abstract void gambar();
}

class Lingkaran extends Bentuk {
    void gambar() {
        System.out.println("Menggambar Lingkaran");
    }
}

class Persegi extends Bentuk {
    void gambar() {
        System.out.println("Menggambar Persegi");
    }
}

public class Main {
    public static void main(String[] args) {
        Bentuk b1 = new Lingkaran();
        Bentuk b2 = new Persegi();

        b1.gambar();
        b2.gambar();
    }
}
```

PHP (abstract class)

```
<?php
abstract class Bentuk {
    abstract public function gambar();
}

class Lingkaran extends Bentuk {
    public function gambar() {
        echo "Menggambar Lingkaran\n";
```

```
    }
}

class Persegi extends Bentuk {
    public function gambar() {
        echo "Menggambar Persegi\n";
    }
}

$b1 = new Lingkaran();
$b2 = new Persegi();

$b1->gambar();
$b2->gambar();
?>
(Di Python, abstraksi biasanya dicapai dengan modul abc, tapi
konsepnya sama: membuat kerangka tanpa detail implementasi.)
```

Latihan & Praktikum

Buat kelas Pegawai dengan atribut nama dan gaji. Buat kelas Manajer yang mewarisi Pegawai dan menambahkan bonus. Gunakan polimorfisme untuk menampilkan total gaji masing-masing.

Buat kelas abstrak Kendaraan dengan metode bergerak(). Turunkan ke Mobil dan Pesawat, lalu implementasikan cara bergeraknya masing-masing.

Resume

- Enkapsulasi: menjaga data tetap aman dengan akses terbatas.
- Inheritance: memungkinkan pewarisan atribut dan metode.
- Polimorfisme: satu metode bisa punya banyak bentuk.
- Abstraksi: fokus pada inti, sembunyikan detail teknis.
- Empat pilar ini adalah fondasi PBO Jika diibaratkan rumah:
- Enkapsulasi = pintu yang hanya bisa dibuka kunci tertentu.
- Inheritance = desain rumah diwariskan ke rumah berikutnya.
- Polimorfisme = satu rumah bisa dipakai untuk tujuan berbeda.
- Abstraksi = Anda cukup tahu cara pakai, tanpa paham detail konstruksi.

Referensi

Booch, Grady. *Object-Oriented Analysis and Design with Applications*. Addison-Wesley.
Lafore, Robert. *Object-Oriented Programming in C++*. Sams Publishing.
Eckel, Bruce. *Thinking in Java*. Prentice Hall.

📌 Penutup

Keempat pilar OOP ini ibarat **empat kaki meja**. Jika satu kaki rapuh, meja tidak akan kokoh. Demikian juga OOP: tanpa memahami **enkapsulasi, inheritance, polimorfisme, dan abstraksi**, pemrograman berorientasi objek akan terasa setengah matang.

- **Enkapsulasi** → menjaga rahasia (keamanan data).
- **Inheritance** → mewarisi sifat (efisiensi & struktur).
- **Polimorfisme** → satu nama, banyak bentuk (fleksibilitas).
- **Abstraksi** → fokus pada inti, sembunyikan detail (kesederhanaan).

Jika mahasiswa benar-benar memahami keempatnya, mereka bisa mendesain sistem perangkat lunak yang **modular, rapi, mudah diperluas, dan tahan lama**.

Worksheet Bab 3 — Empat Pilar OOP

Bagian A: Pemahaman Konsep

1. **Enkapsulasi**
   - Jelaskan dengan kata-kata Anda sendiri apa itu enkapsulasi.
   - Berikan contoh nyata dalam kehidupan sehari-hari (selain rekening bank).
2. **Inheritance**
   - Apa manfaat pewarisan dalam OOP?
   - Buat contoh analogi sederhana selain "anak mewarisi sifat orang tua".
3. **Polimorfisme**
   - Jelaskan dengan contoh bagaimana satu metode bisa memiliki banyak bentuk.
   - Apa perbedaan polimorfisme dengan inheritance?
4. **Abstraksi**
   - Mengapa abstraksi membuat sistem lebih mudah digunakan?
   - Berikan contoh nyata abstraksi di sekitar Anda.

Bagian B: Analisis Kode

Soal 1

Perhatikan kode berikut:

Python

```
class Kendaraan:
    def __init__(self, nama):
        self.nama = nama

    def bergerak(self):
        print("Kendaraan bergerak.")

class Mobil(Kendaraan):
```

```
    def bergerak(self):
        print(f"{self.nama} melaju di jalan raya.")

class Pesawat(Kendaraan):
    def bergerak(self):
        print(f"{self.nama} terbang di udara.")

kendaraan_list = [Mobil("Avanza"), Pesawat("Garuda")]

for k in kendaraan_list:
    k.bergerak()
```

Pertanyaan:

1. Pilar OOP apa yang terlihat pada kode di atas? Sebutkan minimal dua.
2. Apa output dari kode program tersebut?

Soal 2 (Java)

```
abstract class Hewan {
    String nama;
    Hewan(String nama) { this.nama = nama; }
    abstract void suara();
}

class Kucing extends Hewan {
    Kucing(String nama) { super(nama); }
    void suara() { System.out.println(nama + " berkata:
Meong!"); }
}

class Anjing extends Hewan {
    Anjing(String nama) { super(nama); }
    void suara() { System.out.println(nama + " berkata: Guk
guk!"); }
}

public class Main {
    public static void main(String[] args) {
        Hewan[] daftar = { new Kucing("Kitty"), new
Anjing("Buddy") };
        for (Hewan h : daftar) {
            h.suara();
        }
    }
}
```

Pertanyaan:

1. Konsep abstraksi terlihat di bagian mana?
2. Bagaimana polimorfisme muncul di program ini?

Soal 3 (PHP)

```php
<?php
class Pegawai {
    private $nama;
    private $gaji;

    public function __construct($nama, $gaji) {
        $this->nama = $nama;
        $this->gaji = $gaji;
    }

    public function getNama() {
        return $this->nama;
    }

    public function getGaji() {
        return $this->gaji;
    }
}

class Manajer extends Pegawai {
    private $bonus;

    public function __construct($nama, $gaji, $bonus) {
        parent::__construct($nama, $gaji);
        $this->bonus = $bonus;
    }

    public function getTotalGaji() {
        return $this->getGaji() + $this->bonus;
    }
}

$m = new Manajer("Budi", 5000, 2000);
echo $m->getNama() . " total gaji: " . $m->getTotalGaji();
?>
```

Pertanyaan:

1. Bagian mana yang menunjukkan enkapsulasi?
2. Bagian mana yang menunjukkan inheritance?

Bagian C: Praktikum Mandiri

1. **Enkapsulasi**
   - Buat kelas Mahasiswa dengan atribut nama dan IPK yang bersifat private.
   - Tambahkan metode getIPK() dan setIPK() dengan aturan: nilai IPK hanya boleh 0.0 – 4.0.

- Tulis programnya dalam **Python, Java, atau PHP** sesuai pilihan Anda.

2. **Inheritance & Polimorfisme**
   - Buat kelas induk AlatMusik dengan metode bunyi().
   - Turunkan menjadi Gitar, Drum, dan Piano, masing-masing dengan bunyi berbeda.
   - Buat daftar alat musik, lalu panggil metode bunyi() untuk semuanya.
3. **Abstraksi**
   - Buat kelas abstrak BangunDatar dengan metode abstrak luas().
   - Implementasikan untuk Persegi dan Lingkaran.
   - Uji dengan menghitung luas persegi sisi 4 dan lingkaran radius 7.

📝 Bagian D: Refleksi

Tuliskan jawaban singkat:

1. Dari keempat pilar OOP, mana yang menurut Anda paling mudah dipahami? Mengapa?
2. Mana yang paling sulit? Apa kendalanya?
3. Menurut Anda, bagaimana empat pilar ini bisa membantu membangun aplikasi besar (misalnya sistem e-commerce)?

BAB 4
Atribut (Properties) dan Metode (Methods)

Tujuan Pembelajaran

Setelah mempelajari bab ini, Anda diharapkan mampu:

1. Menjelaskan perbedaan dan peran **atribut** (properties) dan **metode** (methods) dalam sebuah kelas.
2. Mengenali jenis atribut (instance vs class/static, mutable vs immutable) dan jenis metode (instance, class/static, konstruktor, destruktor, abstrak).
3. Mengimplementasikan kontrol akses (public/protected/private) dan getter/setter/validation di Python, Java, dan PHP.
4. Mendesain property yang aman (encapsulated) dan metode yang jelas kontraknya (precondition/postcondition).
5. Menerapkan pola praktis: computed properties, lazy loading, fluent API, dan immutability untuk value objects.

Materi

- Definisi atribut (state) dan metode (behavior).
- Instance attribute vs class (static) attribute.
- Visibility / access modifiers: public, protected, private (konvensi Python vs bahasa strictly-typed).
- Getter / Setter / property decorators / magic methods.
- Method types: instance, static, class, constructor, destructor/finalizer, abstract.
- Overriding vs overloading (perbedaan antar bahasa).
- Computed properties & lazy evaluation.
- Immutability (value objects) dan konsekuensinya.
- Best practices desain atribut & metode (SRP kecil, naming, side-effects, dokumentasi).

Bayangkan sebuah **mobil**. Mobil punya ciri-ciri—warna, merk, kapasitas tangki—dan punya perilaku—jalan(), rem(), klakson(). Di OOP, ciri-ciri disebut **atribut** (properties) sedangkan perilaku disebut **metode** (methods). Bab ini

mengupas keduanya sampai ke detail desain yang penting bagi software yang tahan uji dan mudah dirawat.

A. Atribut: Apa, Jenis, dan Aturan Desain

Atribut menyimpan *state* objek (Dennis, Wixom, & Tegarden, 2021). Ada beberapa hal penting yang harus dimengerti:

- **Instance attribute** — data yang unik untuk setiap objek. Contoh: mobil1.warna = "Merah", mobil2.warna = "Hitam".
- **Class/static attribute** — data yang dibagi oleh semua instance, cocok untuk konfigurasi bersama. Contoh: Vehicle.taxRate = 0.1.
- **Mutable vs immutable** — beberapa atribut boleh berubah (kecepatan), beberapa sebaiknya tidak berubah setelah inisialisasi (UUID, tanggal lahir). Value object yang immutable mempermudah reasoning dan thread-safety.
- **Visibility / akses** — prinsip enkapsulasi menuntun kita menyembunyikan atribut internal. Di Java/PHP ada modifier (private/protected/public). Di Python konvensinya memakai _private atau __mangled dan property untuk kontrol akses.

Desain yang baik untuk atribut:

- Buat sedikit public surface area: minimalkan atribut publik langsung; gunakan getter/setter jika perlu validasi.
- Gunakan final/readonly/frozen bila atribut harus immutable.
- Pertimbangkan memisahkan state yang sering berubah vs jarang berubah untuk optimisasi.

Contoh atribut & property — Python, Java, PHP

Python (property, validation, class attr):

```
class Product:
    tax_rate = 0.10  # class attribute (shared)

    def __init__(self, name: str, price: float):
        self.name = name            # public by convention
        self._price = float(price)  # "protected" by
convention

    @property
    def price(self):
        return self._price

    @price.setter
    def price(self, value):
        if value < 0:
            raise ValueError("price harus non-negatif")
        self._price = value

    @property
```

```
    def price_with_tax(self):
        return self._price * (1 + self.tax_rate)
```

Java (private field + getter/setter + static field):

```
public class Product {
    private static double taxRate = 0.10; // class / static
    private final String name;            // immutable after
construction
    private double price;                 // instance state

    public Product(String name, double price) {
        this.name = name;
        setPrice(price);
    }

    public String getName() { return name; }
    public double getPrice() { return price; }
    public void setPrice(double price) {
        if (price < 0) throw new
IllegalArgumentException("price must be >= 0");
        this.price = price;
    }

    public double getPriceWithTax() { return price * (1 +
taxRate); }
    public static void setTaxRate(double rate) { taxRate =
rate; }
}
```

PHP 8+ (typed properties, accessors):

```
class Product {
    public static float $taxRate = 0.10;
    private string $name;
    private float $price;

    public function __construct(string $name, float $price) {
        $this->name = $name;
        $this->setPrice($price);
    }

    public function getPrice(): float { return $this->price; }
    public function setPrice(float $p): void {
        if ($p < 0) throw new InvalidArgumentException("price
must be >= 0");
        $this->price = $p;
    }

    public function getPriceWithTax(): float {
        return $this->price * (1 + self::$taxRate);
```

```
    }
}
```

B. Metode: Jenis, Kontrak, dan *Best Practice*

Metode adalah fungsi yang terkait dengan objek/kelas. Jenis-jenis metode penting yang sering muncul:

- **Instance method** — menerima self/this dan beroperasi pada state instance. (umum)
- **Class method / static method** — beroperasi pada kelas, bukan instance. (konfigurasi bersama, helper)
  - Python: @classmethod (menerima cls), @staticmethod (tidak menerima self/cls).
  - Java: static methods.
  - PHP: public static function ....
- **Constructor** — inisialisasi state awal (__init__ / nama kelas / __construct).
- **Destructor / finalizer** — pembersihan resource (jarang dipakai; gunakan context manager atau try-with-resources).
- **Abstract method** — kontrak yang harus diimplementasi oleh turunan.
- **Magic / special methods** — bahasa tertentu punya metode khusus (__repr__, __eq__ di Python; toString() di Java; __toString() di PHP).

Kontrak metode (pre/postconditions):
Setiap metode idealnya punya kontrak yang jelas: input diperiksa (precondition), efek samping dijelaskan, dan output dijamin (postcondition). Dokumentasikan lewat docstring/Javadoc/phpdoc.

Best practice metode:

- **Single Responsibility**: setiap metode punya satu tugas kecil.
- Hindari efek samping tersembunyi (mutasi global).
- Buat metode mudah di-test (dependensi disuntikkan).
- Beri nama metode yang jelas—apply_discount() lebih deskriptif daripada doIt().

Contoh metode khusus: static, class, fluent, magic
Python (instance, classmethod, staticmethod, fluent):

```
class Product:
    tax_rate = 0.10

    def apply_discount(self, percent: float):
        if not 0 <= percent <= 100:
            raise ValueError("percent between 0 and 100")
        self._price *= (100 - percent) / 100
        return self  # fluent API (chainable)
```

```
    @classmethod
    def set_tax_rate(cls, rate: float):
        cls.tax_rate = rate

    @staticmethod
    def currency_symbol():
        return "Rp"
```

Java (static, instance, method chaining via returning this):

```
public Product applyDiscount(double percent) {
    if (percent < 0 || percent > 100) throw new
IllegalArgumentException();
    this.price *= (100 - percent) / 100.0;
    return this; // fluent
}

public static void setTaxRate(double rate) { taxRate = rate; }
```

PHP (static, chaining):

```
public function applyDiscount(float $percent): self {
    if ($percent < 0 || $percent > 100) throw new
InvalidArgumentException();
    $this->price *= (100 - $percent) / 100;
    return $this;
}
```

C. Overloading vs Overriding — Perbedaan Antar Bahasa

- **Overriding**: turunan mengganti implementasi metode yang sama dari superclass — tersedia di semua bahasa OOP utama.
- **Overloading**: definisi beberapa metode dengan nama sama tapi parameter berbeda — didukung native di **Java** dan **PHP** (PHP tidak strict method overloading, tapi bisa via default params / magic), **Python** tidak mendukung overloading berdasarkan signature pada runtime (bisa dicapai dengan default args, *args/**kwargs, atau functools.singledispatch).

Java (overloading & overriding):

```
void add(int a, int b) { ... }       // overloading (signature
berbeda)
void add(double a, double b) { ... }

class Parent { void speak() { ... } }
class Child extends Parent { @Override void speak() { ... } }
// overriding
```

Python (emulasi overloading):

```
def greet(name=None):
    if name is None:
        print("Hi")
    else:
        print(f"Hi {name}")
```

D. Computed Properties & Lazy Loading

Kadang atribut bukan disimpan, melainkan dihitung ketika diminta—itu **computed property**. Lazily computed property hanya dihitung pada akses pertama dan disimpan.

Python (property + lazy):

```
class User:
    def __init__(self, data_source):
        self._data_source = data_source
        self._profile = None  # lazy cache

    @property
    def profile(self):
        if self._profile is None:
            self._profile = self._data_source.load_profile()
        return self._profile
```

Lazy berguna untuk resource berat (I/O, DB queries).

E. Immutability & Value Objects

Untuk objek yang mewakili nilai (money, date, coordinate), **immutability** mengurangi bug. Di Python: @dataclass(frozen=True), Java: final fields + no setters, PHP: no public setters.

Python:

```
from dataclasses import dataclass

@dataclass(frozen=True)
class Money:
    amount: float
    currency: str
Immutability mempermudah reasoning, concurrent use, dan
implementasi sebagai key di map (hashability).
```

F. Performance & Trade-Offs

Akses properti langsung sedikit lebih cepat daripada method call di beberapa bahasa, tetapi perbedaan biasanya kecil dibanding keuntungan enkapsulasi. Jangan korbankan desain hanya demi micro-optimisasi; profil sebenar aplikasi sebelum optimasi.

G. Dokumentasi & Testing

- Tuliskan docstring / Javadoc / phpdoc pada atribut dan metode penting.
- Tulis unit test untuk: validasi setter, behavior metode, invariants kelas.
- Gunakan type hints (Python typing), strict types (PHP declare(strict_types=1)), dan type checking (Java compiler) untuk mendeteksi kelas bug lebih awal.

Latihan & Praktikum (praktek langsung, dengan contoh solusi singkat)
Di bawah ini ada beberapa tugas praktikum. Saya sertakan **solusi contoh singkat** untuk tugas Product agar Anda bisa langsung mencoba.

Soal 1 — Desain kelas Product (validasi + computed property + fluent)

- Kebutuhan:
  - atribut name, price (price ≥ 0), class attribute tax_rate.
  - property price_with_tax.
  - metode apply_discount(percent) yang mengembalikan self untuk chaining.
  - method/class untuk mengubah tax_rate.

Solusi ringkas — Python

```
class Product:
    tax_rate = 0.10
    def __init__(self, name: str, price: float):
        self.name = name
        self._price = float(price)
    @property
    def price(self): return self._price
    @price.setter
    def price(self, v):
        if v < 0: raise ValueError("price >= 0")
        self._price = v
    @property
    def price_with_tax(self): return self._price * (1 +
self.tax_rate)
    def apply_discount(self, percent):
        if not 0 <= percent <= 100: raise ValueError
        self._price *= (100 - percent)/100
        return self
    @classmethod
    def set_tax_rate(cls, r): cls.tax_rate = r
```

Solusi ringkas — Java
(terlihat pada bagian penjelasan sebelumnya; gunakan private field, getter/setter, static taxRate, method chaining returning this)
Solusi ringkas — PHP
(terlihat pada bagian penjelasan sebelumnya; typed properties, private fields, static, method chaining return $this)

Soal 2 — Alternate constructors / Factory methods
Buat cara alternatif membuat objek (mis: Product.from_dict() di Python; overloaded constructor di Java).
Panduan:

- Di Python: gunakan @classmethod def from_dict(cls, data):
- Di Java: overload constructor atau static factory public static Product from(Map<String,Object> m)
- Di PHP: static public static function fromArray(array $a): self.

Soal 3 — Immutability (Money value object)
Buat kelas Money immutable: operasi add(Money) menghasilkan Money baru.
Panduan implementasi: di Java gunakan final fields; di Python gunakan @dataclass(frozen=True); di PHP hindari setter, gunakan private fields dan return instance baru dari operasi.

Resume (ringkasan praktis)

- **Atribut** menyimpan state; bedakan instance vs class/static; gunakan encapsulation (private + getter/setter) untuk validasi.
- **Metode** mengimplementasikan perilaku; desain kontrak yang jelas; hindari efek samping yang mengejutkan.
- Gunakan **property** (Python) atau getter/setter (Java/PHP) untuk computed attributes dan validasi.
- Prefer **immutability** untuk value objects; gunakan final / frozen/no-setter pola.
- **Overriding** mengganti perilaku superclass; **overloading** (Java) definisikan metode sama dengan parameter berbeda.
- Dokumentasikan dan tulis unit tests untuk invariants dan perilaku metode.

Referensi singkat (untuk pendalaman)

- *Effective Java* — Joshua Bloch (prinsip desain kelas & methods di Java).
- *Fluent Python* — Luciano Ramalho (pattern idiomatik Python).
- *PHP Objects, Patterns, and Practice* — Matt Zandstra (OOP & praktik PHP).

- Dokumentasi resmi: Python Data Model (special methods), Java Language Specification (fields & methods), PHP Manual (OOP features).

Worksheet Bab 4 — Atribut (Properties) dan Metode (Methods)

Bagian A: Pemahaman Konsep

1. Jelaskan dengan kata-kata Anda sendiri apa yang dimaksud dengan **atribut** dan **metode** dalam PBO
2. Apa perbedaan **instance attribute** dengan **class/static attribute**? Berikan contoh nyata.
3. Mengapa dalam OOP kita sering menjadikan atribut **private** lalu menyediakan getter/setter?
4. Apa perbedaan antara **instance method**, **class/static method**, dan **constructor**?
5. Jelaskan dengan contoh sederhana perbedaan **overloading** dan **overriding**.

Bagian B: Analisis Kode

Soal 1 (Python)

```
class Mahasiswa:
    total_mahasiswa = 0   # class attribute

    def __init__(self, nama):
        self.nama = nama
        Mahasiswa.total_mahasiswa += 1

    def sapa(self):
        print(f"Halo, saya {self.nama}")

m1 = Mahasiswa("Ani")
m2 = Mahasiswa("Budi")
m1.sapa()
print("Jumlah mahasiswa:", Mahasiswa.total_mahasiswa)
```

Pertanyaan:

1. Mana yang merupakan atribut instance dan mana yang class attribute?
2. Apa output dari program di atas?

Soal 2 (Java)

```
public class Lingkaran {
    private double radius;

   public Lingkaran(double radius) {
        setRadius(radius);
    }

    public double getRadius() { return radius; }

    public void setRadius(double r) {
```

```
        if (r < 0) throw new IllegalArgumentException("Radius tidak
boleh negatif");
        this.radius = r;
    }

    public double luas() {
        return Math.PI * radius * radius;
    }

    public static void main(String[] args) {
        Lingkaran l = new Lingkaran(7);
        System.out.println("Luas lingkaran: " + l.luas());
    }
}
```

Pertanyaan:

1. Bagian mana yang menunjukkan enkapsulasi?
2. Mengapa setter digunakan untuk mengatur radius, bukan langsung public field?

Soal 3 (PHP)

```
<?php
class Kalkulator {
    public static function tambah($a, $b) {
        return $a + $b;
    }

    public function kuadrat($x) {
        return $x * $x;
    }
}

echo Kalkulator::tambah(5, 3) . PHP_EOL;

$k = new Kalkulator();
echo $k->kuadrat(4);
?>
```

Pertanyaan:

1. Mana yang merupakan static method dan mana instance method?
2. Apa perbedaan cara pemanggilannya?

Bagian C: Praktikum Mandiri

1. **Validasi dengan Getter/Setter**
    - Buat kelas Pegawai dengan atribut nama (string) dan gaji (float, tidak boleh negatif).
    - Terapkan enkapsulasi: gunakan private field + getter/setter.
    - Tambahkan metode info() untuk menampilkan data pegawai.

2. **Static Attribute & Method**
   - Buat kelas Counter dengan atribut static jumlah yang menghitung berapa kali objek dibuat.
   - Tambahkan static method getJumlah() untuk menampilkan jumlah instance.
   - Uji dengan membuat 3 objek dan cetak hasilnya.
3. **Overriding Method**
   - Buat kelas Hewan dengan metode suara().
   - Buat kelas turunan Kucing dan Anjing yang masing-masing meng-override suara().
   - Buat array/list of Hewan, lalu panggil suara() pada setiap elemen.
4. **Fluent API**
   - Buat kelas Produk dengan metode setHarga(), setDiskon(), hitungHargaFinal().
   - Gunakan method chaining sehingga bisa dipanggil seperti:
   - p = Produk("Laptop", 10000).setDiskon(10).hitungHargaFinal()

Bagian D: Refleksi

Tuliskan jawaban singkat (1–2 paragraf):

1. Menurut Anda, apa tantangan terbesar dalam memahami atribut dan metode dalam OOP?
2. Bagaimana perbedaan konsep ini antara Python, Java, dan PHP?
3. Bagaimana Anda akan mendesain kelas **"AkunBank"** agar aman dari penyalahgunaan atribut oleh pengguna program?

Latihan Uji Kompetensi – Bab 4

Topik: Atribut (Properties) & Metode (Methods)

Bagian A: Pilihan Ganda

1. Di bawah ini yang **bukan** contoh atribut dalam OOP adalah ...
   a. nama
   b. usia
   c. hitungLuas()
   d. alamat
2. Jika sebuah atribut dibuat private, maka ...
   a. Tidak dapat diakses langsung dari luar kelas
   b. Tidak bisa digunakan sama sekali
   c. Hanya bisa digunakan di kelas turunan
   d. Bisa diakses bebas oleh semua kelas
3. **Static attribute** digunakan ketika ...
   a. Nilai atribut berbeda-beda untuk setiap objek
   b. Nilai atribut harus sama dan dibagikan ke semua objek
   c. Nilai atribut hanya ada di constructor
   d. Atribut hanya bisa dipanggil lewat objek