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).

Buku Ajar
Pemrograman Berorientasi Objek

Diterbitkan pertama kali dalam bahasa Indonesia
oleh Penerbit Global Aksara Pers

ISBN: **978-623-462-991-0**
ix + 188 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

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

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.

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.

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. Ia belum nyata, tapi berisi "aturan main" tentang apa saja yang dimiliki dan bisa dilakukan oleh objek. (Kadir, A., 2018).

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.

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. Ada beberapa hal penting yang harus dimengerti: (Lutz, M., 2013)

- **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
```


- Disarankan untuk tidak langsung memberi solusi agar mahasiswa berlatih analisis.

(Oracle, 2024) (Python Software Foundation., 2024) (Saputra, Agus. (2017)

Cheat Sheet – Atribut & Metode (Python vs Java vs PHP)

Konsep	Python	Java	
Membuat Kelas	python class Mahasiswa: pass	java class Mahasiswa { }	php <?php class Mahasiswa { }

Constructor	python
	def __init__(self, nama):
	 self.nama = nama
		java
		public Mahasiswa(String nama) {
		 this.nama = nama;
		}
			php
			public function __construct($nama) {
			 $this->nama = $nama;
			}

Atribut Instance	self.nama	this.nama	$this->nama

Atribut Static	class Mahasiswa:
	 total = 0akses: Mahasiswa.total
		class Mahasiswa {
		 static int total = 0;
		}akses: Mahasiswa.total
			class Mahasiswa {
			 public static $total = 0;
			}akses: Mahasiswa::$total

Method Instance	def sapa(self):
	 print("Halo")
		public void sapa() {
		 System.out.println ("Halo");
		}
			public function sapa() {
			 echo "Halo\n";
			}

Static Method	@staticmethod def hitung(): ...panggil: Kelas.hitung()	public static void hitung() { ... }panggil: Kelas.hitung();	public static function hitung() { ... }panggil: Kelas::hitung();
Getter/Setter (manual)	def get_nama(self): return self.nama	public String getNama() { return nama; }	public function getNama() { return $this->nama; }
Enkapsulasi (private)	Konvensi _atribut (tidak ketat)	Modifier private	Modifier private
Overloading	Tidak ada bawaan, bisa pakai default argumen	Metode dengan nama sama, parameter beda	Tidak didukung penuh, bisa pakai argumen opsional
Overriding	Definisikan ulang metode di subclass	Gunakan @Override	Definisikan ulang metode di subclass
Pemanggilan Method	obj.sapa()	obj.sapa();	$obj->sapa();

```
        System.out.println(nama + " (" + nim + ") " + jurusan + "
IPK:" + ipk);
    }

    public static void main(String[] args) {
        Mahasiswa m = new Mahasiswa.Builder()
            .nama("Ani")
            .nim("123")
            .jurusan("Informatika")
            .ipk(3.9)
            .build();
        m.info();
    }
}
```

📌 **Bagian E: Refleksi**

1. Apa trade-off antara **constructor sederhana** vs **builder pattern**?
2. Mengapa factory methods dianggap lebih fleksibel daripada constructor public?
3. Apa konsekuensi jika kita lupa membersihkan resource di aplikasi jangka panjang (misalnya server)?
4. Bagaimana Anda merancang kelas agar tetap mudah diuji meskipun punya banyak dependensi?

🔎 **Apa itu MRO (Method Resolution Order)?**

Ketika sebuah **objek** diakses untuk menjalankan **metode atau atribut**, bahasa pemrograman harus tahu: *dari mana seharusnya metode itu dipanggil?*

Ini sederhana kalau hanya ada satu kelas induk. Tapi akan membingungkan kalau ada **multiple inheritance** (kelas punya lebih dari satu parent).

Nah, **MRO (Method Resolution Order)** adalah aturan atau algoritma yang dipakai Python (dan beberapa bahasa lain dengan multiple inheritance) untuk **menentukan urutan pencarian metode atau atribut** dalam hierarki kelas (Zend T. , 2024)

🔷 **Ilustrasi sederhana**

Bayangkan Anda mahasiswa yang belajar dari banyak dosen:

- Dosen A mengajarkan *Metode Penelitian*.
- Dosen B mengajarkan *Metode Penelitian* juga, tapi dengan gaya berbeda.
- Anda punya "super-dosen" (kelas anak) yang berguru ke **dua dosen sekaligus**.

Kalau Anda ditanya, *"Bagaimana cara penelitian menurutmu?"* → dari dosen mana jawaban Anda diambil?

BAB 14
Design Patterns dalam OOP

🎯 **Tujuan Pembelajaran**

Setelah mempelajari bab ini, mahasiswa akan mampu:

1. Memahami konsep **Design Patterns** dan manfaatnya dalam PBO
2. Mengenal kategori utama design pattern: **Creational, Structural, dan Behavioral**.
3. Mengimplementasikan beberapa design pattern sederhana dalam kode nyata.
4. Menggunakan design pattern untuk membuat **sistem modular, reusable, dan maintainable**.

A. Pendahuluan

Dalam pengembangan software, **reusability, scalability, dan maintainability** adalah kunci. Design pattern adalah **solusi standar yang sudah terbukti** untuk masalah umum dalam PBO (Gamma, E., Helm, R., Johnson, R., & Vlissides, J., 1994)

Analogi ilmiah:

- Bayangkan kamu sedang membangun laboratorium robotik.
- Alih-alih mendesain setiap mesin dari nol, kamu menggunakan **template blueprint yang sudah terbukti aman dan efisien**.
- Blueprint ini adalah **design pattern**.
- Dengan pattern, kamu menghemat waktu, mengurangi bug, dan mempermudah kolaborasi tim.

B. Kategori Design Pattern

1️⃣ **Creational Pattern**

Membantu membuat objek dengan cara fleksibel dan terkontrol.

- Contoh: **Singleton**, **Factory Method**
- Manfaat: menghindari duplikasi, mengontrol jumlah instance, membuat kode lebih modular.

2️⃣ **Structural Pattern**

Membantu menyusun class dan objek sehingga membentuk struktur yang mudah dipahami.