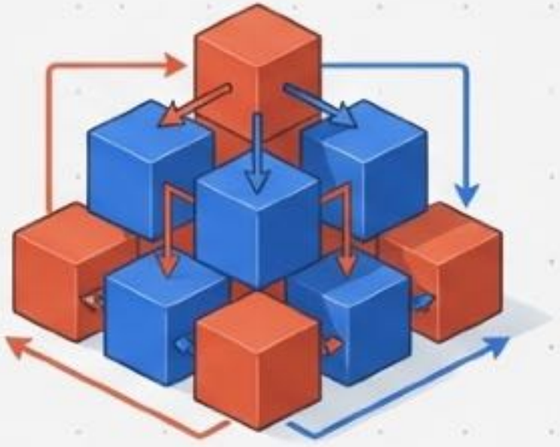
# Object-Oriented Programming (OOP) dalam PHP

Arsitektur Masa Depan Aplikasi Web


Rizki Muliono, M.Kom | Teknik Informatika UMA
Aplikasi Berbasis Web II

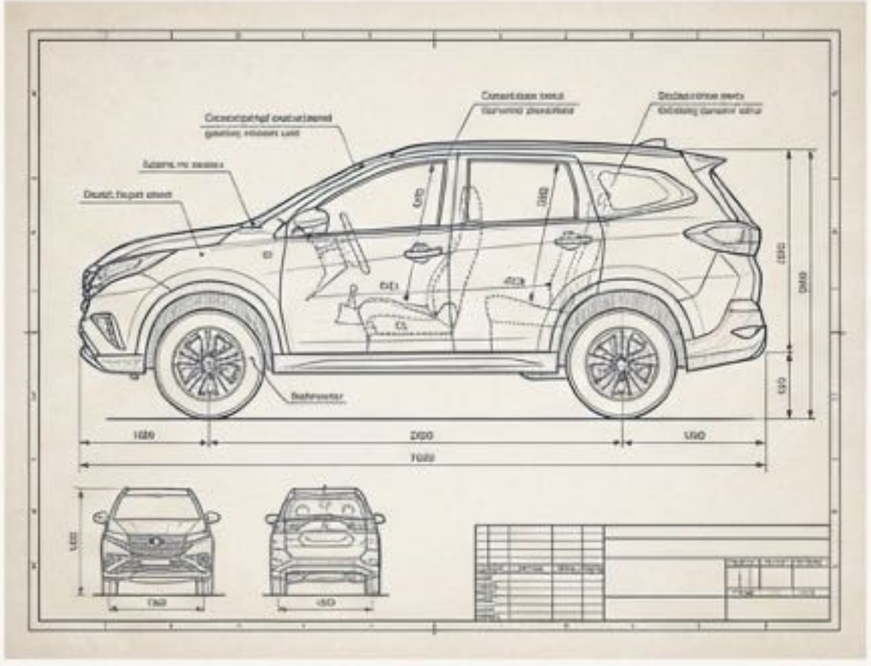
# Evolusi Paradigma: Prosedural vs. Berorientasi Objek

| | Prosedural | Berorientasi Objek |
|---|---|---|
| |  |  |
| 1 | **Fokus Utama:** Urutan perintah dan eksekusi Fungsi. | **Fokus Utama:** Entitas (Objek) yang menggabungkan Data & Fungsi. |
| 2 | **Maintenance:** Sulit dilacak saat skala aplikasi membesar (Spaghetti Code). | **Maintenance:** Terstruktur, mudah dimodifikasi, dan di-debug. |
| 3 | **Skalabilitas:** Terbatas. | **Skalabilitas:** Reusability Tinggi / DRY (Don't Repeat Yourself). |

⚠ **Realitas OOP:** Menawarkan struktur dan kecepatan pengembangan jangka panjang, namun memiliki kurva belajar yang lebih curam dan *overhead* memori sedikit lebih besar dibanding prosedural murni.

# Konsep Inti: Pemisahan Class dan Object

## CLASS (Blueprint/Cetakan)



Kode aktual yang mendefinisikan struktur, atribut, dan fungsi dasar. Contoh: Class Mobil.

## OBJECTS (Instansiasi)

Hasil nyata dari class yang hidup di dalam memori dan mengandung data spesifik.

Objek Avanza

Objek Fortuner

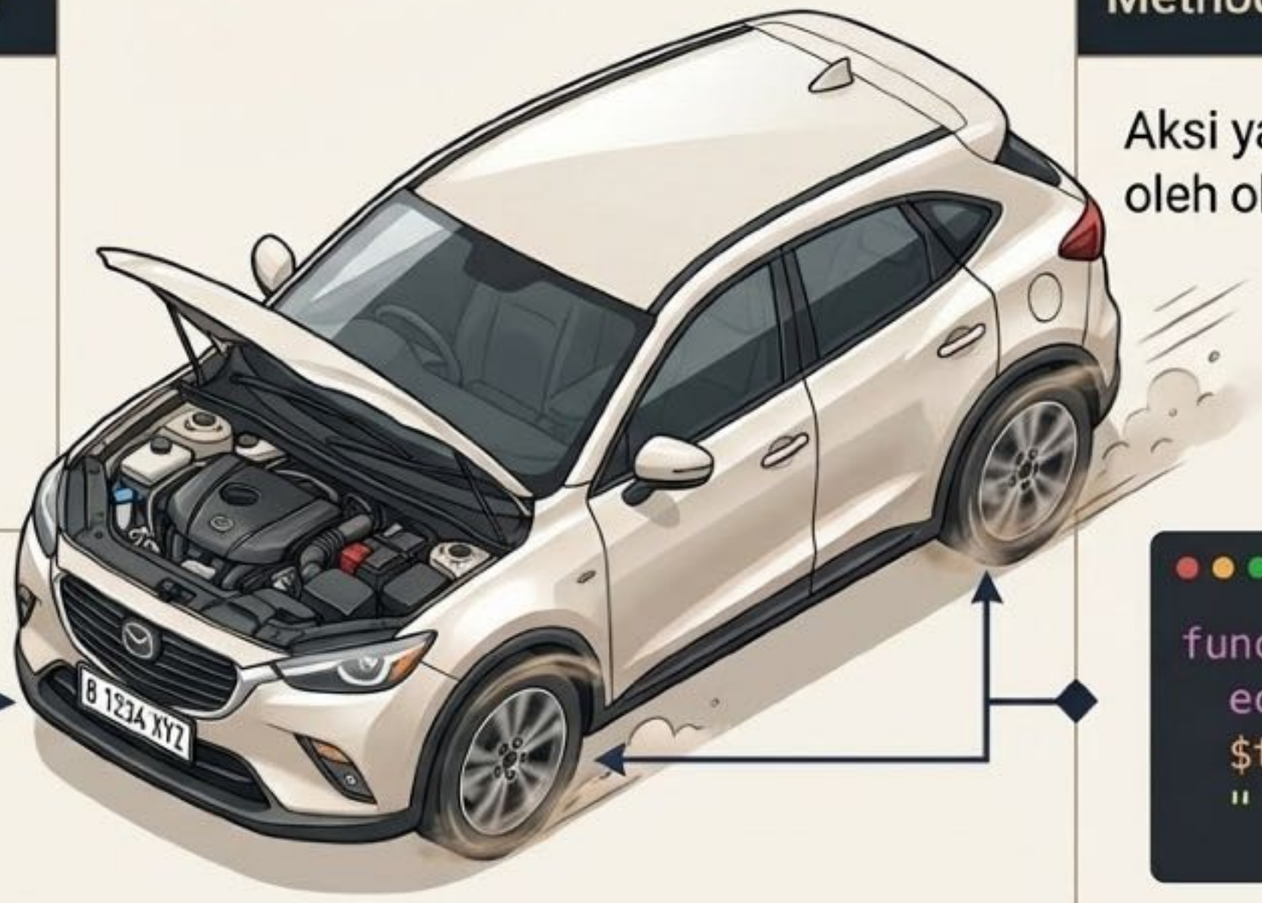
```
$avanza = new Mobil();
```

# Anatomi Class PHP: Properties & Methods

## Properties (State/Atribut)

Variabel yang menyimpan informasi objek.

```php
public $nama_mobil;
public $plat_nomor;
```




## Methods (Behavior/Fungsi)

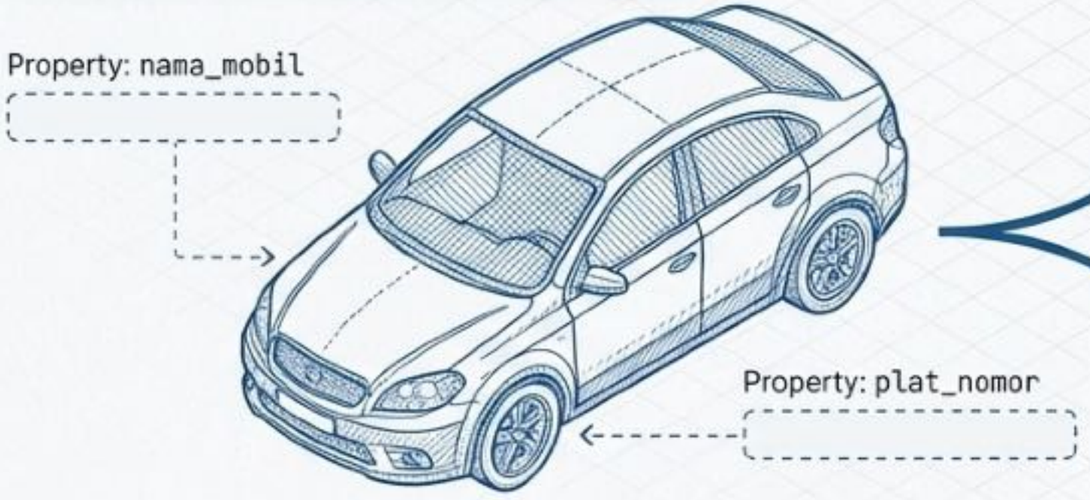
Aksi yang dapat dilakukan oleh objek.

```php
function maju() {
  echo "Mobil " .
  $this->nama_mobil .
  " maju"; }
```

**Insight:** Gunakan keyword **$this->** untuk mengakses properties dari dalam class itu sendiri.

# The Core Concept: Classes & Objects

**Class (The Blueprint)**
The universal template. It defines what properties exist, but holds no actual data.




```php
class Mobil {
    public $nama_mobil;
    public $plat_nomor;
}
```

**Object (The Instance)**
Inter

```php
$nama_mobil = 'Avanza';
$plat_nomor = 'AD 1234 BC';
```

**Object (The Instance)**
Inter

```php
$nama_mobil = 'Fortuner';
$plat_nomor = 'AB 5678 DE';
```

# The Anatomy of a Class: State and Behavior

```php
class Mobil {
    public $nama_mobil;
    public $plat_nomor;

    public function maju() {
        echo $this->nama_mobil . ' telah maju.';
    }
}
```

### Properties (The State)

Variables belonging directly to the class. Defined with access modifiers.

### Methods (The Behavior)

Functions belonging to the class that define exactly what the object can do.

### The $this Keyword

An internal mirror. It allows the class to point to its own specific instantiated properties from within a method.

# Proteksi Akses Data (Visibility Modifiers)

Public

## Public

Tersedia dari mana saja (Default level). Seluruh kode dapat melihat dan mengubahnya.

Protected

## Protected

Akses terbatas. Hanya tersedia pada class itu sendiri dan class turunannya (Inheritance).

Private

## Private

Terkunci rapat. Hanya dapat diakses secara eksklusif dari dalam class-nya sendiri.

**Best Practice:** Selalu gunakan visibilitas Private/Protected untuk data sensitif guna mencegah modifikasi tak terduga dari luar sistem.

# 4 Pilar Utama Arsitektur OOP

Arsitektur Software Tangguh

# 4 Pilar Utama Object-Oriented Programming

**Encapsulation (Enkapsulasi)**
- Melindungi data.

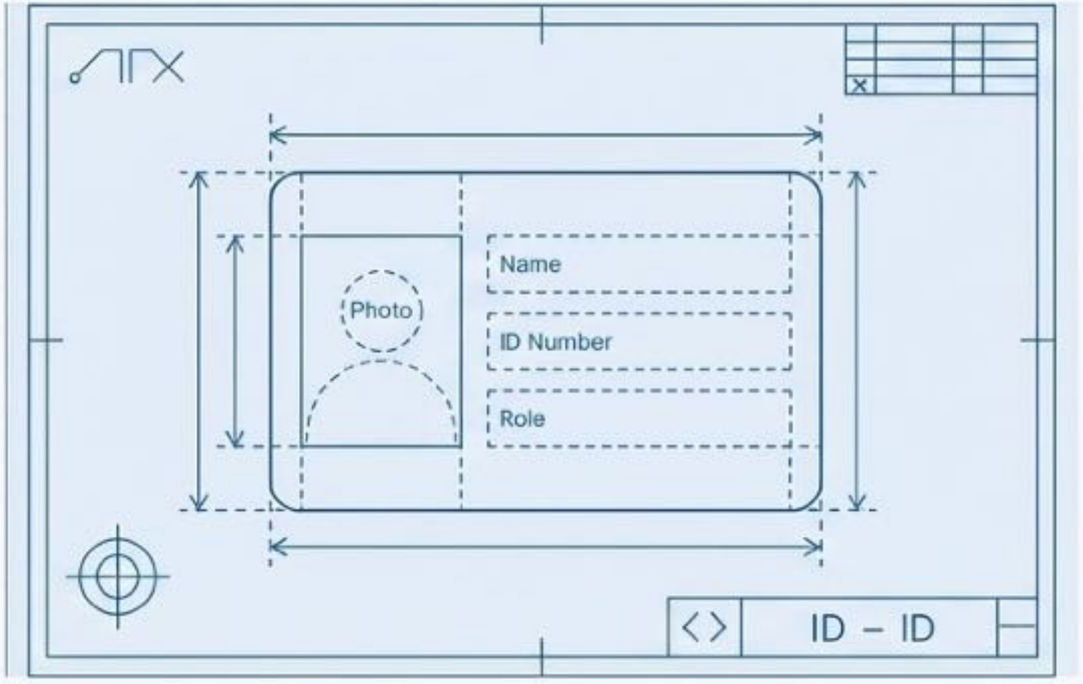
**Abstraction (Abstraksi)**
- Menyembunyikan kerumitan.

**Inheritance (Pewarisan)**
- Berbagi kapabilitas.

**Polymorphism (Banyak Bentuk)**
- Fleksibilitas aksi.

Keempat pilar ini bekerja berkesinambungan untuk mengubah kode acak menjadi sistem enterprise yang dapat diskalakan.

# Blueprint vs. Realita (Class & Object)





Class: Cetakan utama yang mendefinisikan Properties dan Methods. Tidak mengandung data spesifik.

```php
<?php
class Demo {
    // Blueprint
}

$objDemo = new Demo(); // Instansi Object
?>
```

Object: Instansi yang berjalan dari Class. Mengandung data internal aktual yang dibutuhkan aplikasi.

# Pillar 1: Encapsulation & The Access Matrix

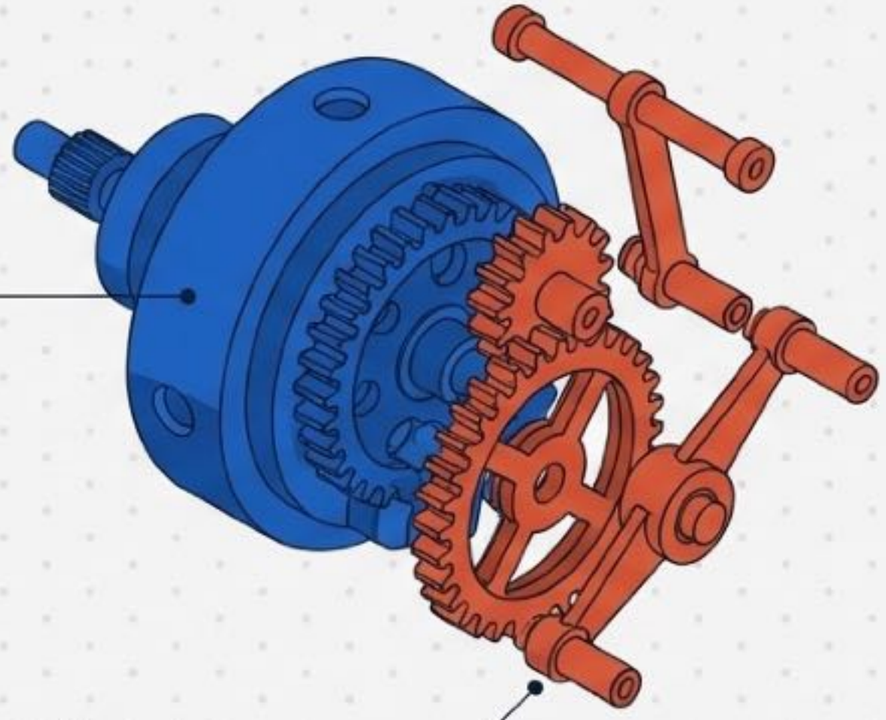
**Concept:** Bundling data and methods while strictly controlling outside interference to prevent invalid states.

| | Inside Own Class | Inside Child Classes | Outside the Class |
|---|---|---|---|
| `public` | ✅ | ✅ | ✅ |
| `protected` | ✅ | ✅ | 🔒 |
| `private` | ✅ | 🔒 | 🔒 |

**Best Practice:** Keep properties private or protected. Use explicit getter and setter methods to control and validate all data modifications.

# Anatomi Class: Properties & Methods

**Properties**: Variabel yang hidup di dalam class. Menyimpan state atau data (contoh: $name).



**Methods**: Fungsi yang hidup di dalam class. Menentukan behavior atau aksi objek (contoh: SayHello()).

```php
<?php
class Demo {
    public $name; // Property

    function SayHello() { // Method
        echo "Hello $this->name!";
    }
}
?>
```

# Menjaga Keamanan Data (Encapsulation)

**Public**

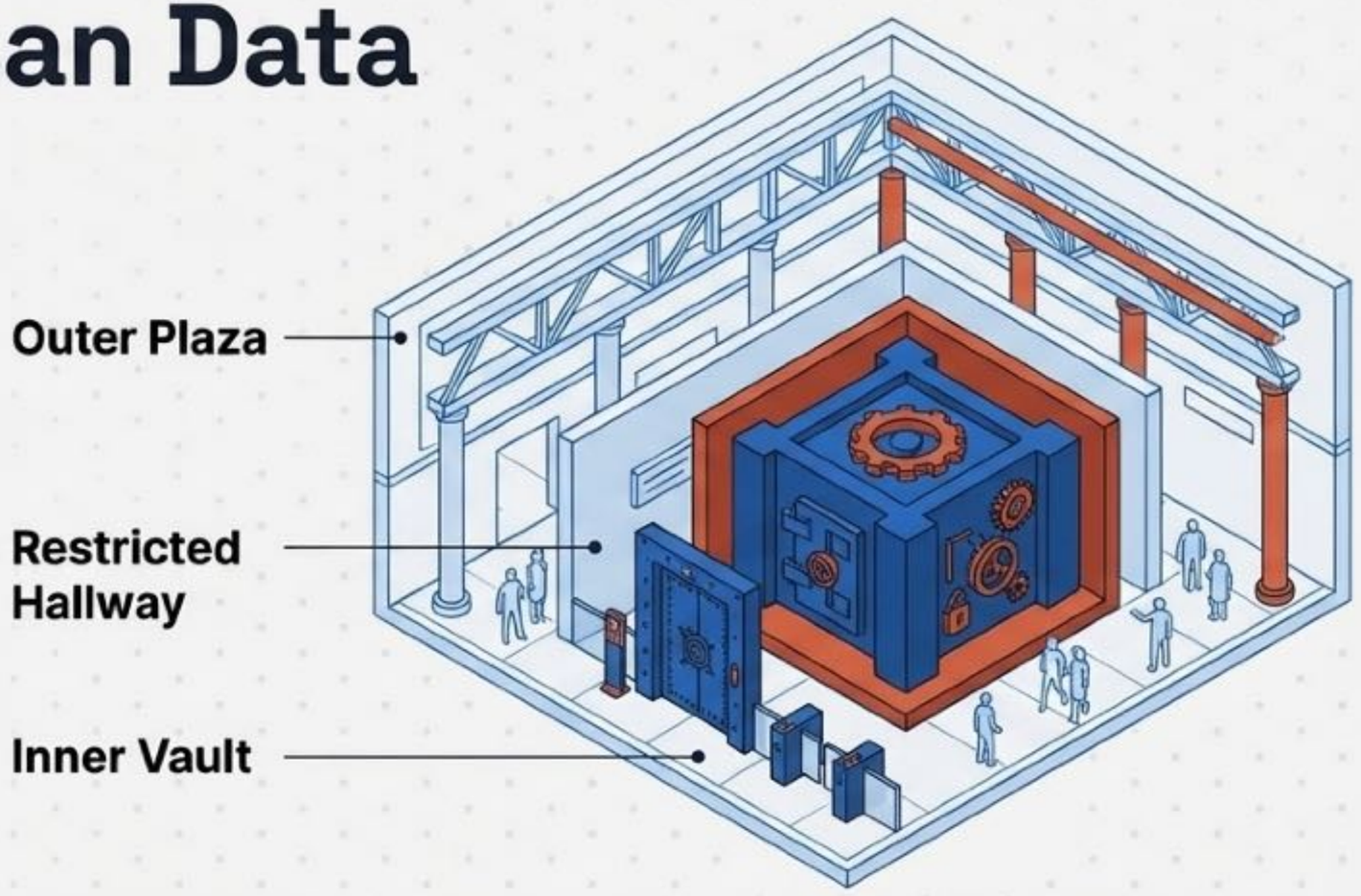
Jangkauan default. Siapa saja dapat mengakses properti atau method dari seluruh bagian kode.

**Protected**

Hanya dapat diakses oleh class itu sendiri dan class turunannya (melalui proses Inheritance).

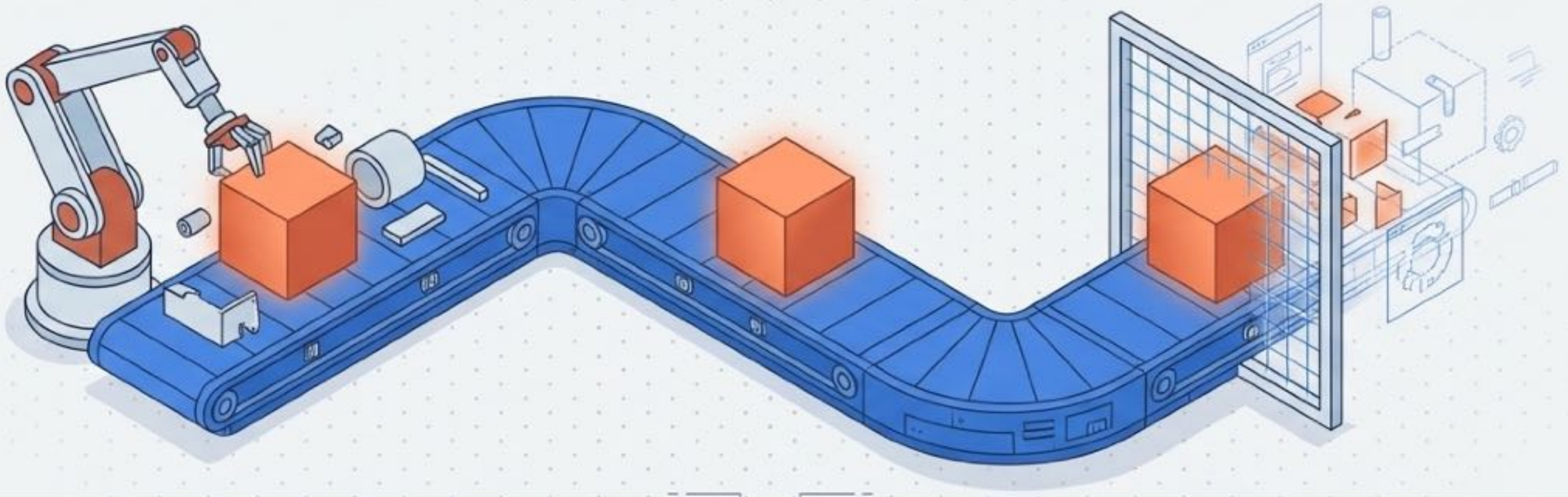
**Private**

Akses paling ketat. Hanya bisa dibaca/diubah secara eksklusif oleh class itu sendiri.




**Golden Rule:** Selalu gunakan fungsi Getter dan Setter untuk properti private. Ini memastikan bisnis logic dan validasi data tetap terkontrol.

# Siklus Hidup Objek (Constructor & Destructor)



## Titik Awal: __construct()

Method 'Magic' yang dijalankan otomatis saat objek pertama kali diinisialisasi menggunakan perintah new Class(). Paling sering digunakan untuk setup nilai awal dan injeksi dependensi.

## Titik Akhir: __destruct()

Dijalankan otomatis sesaat sebelum objek dihapus dari memori atau ketika eksekusi script berakhir. Sangat krusial untuk menutup koneksi database atau membersihkan resource memori.

# Mewariskan Karakteristik (Inheritance)

## Superclass & Subclass

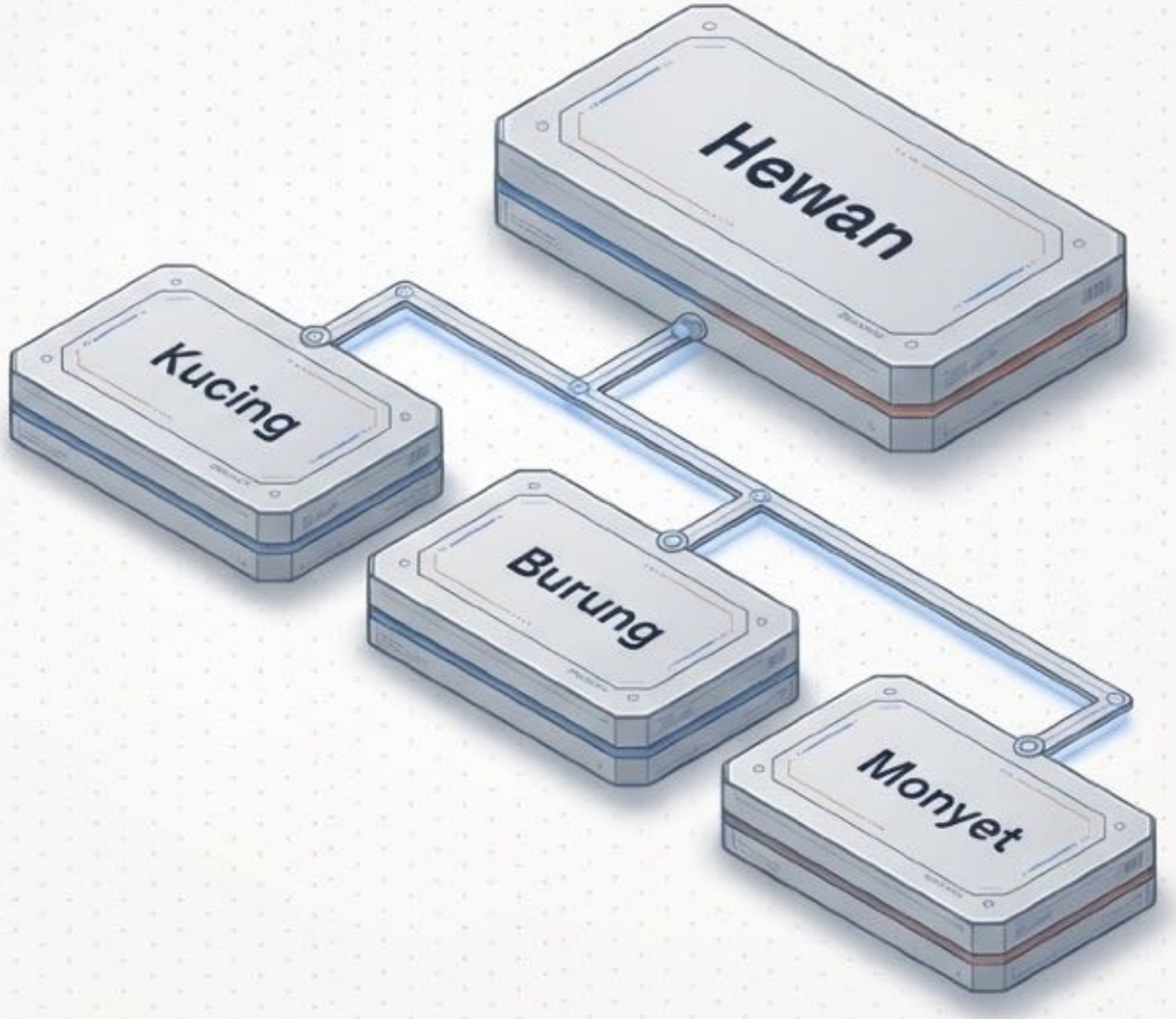
Mendefinisikan class baru yang merupakan sub-tipe spesifik dari class induk yang sudah ada secara hierarkis.

## Menghindari Redundansi

Class anak secara otomatis mewarisi semua properties dan methods (berstatus Public/Protected) dari induknya.

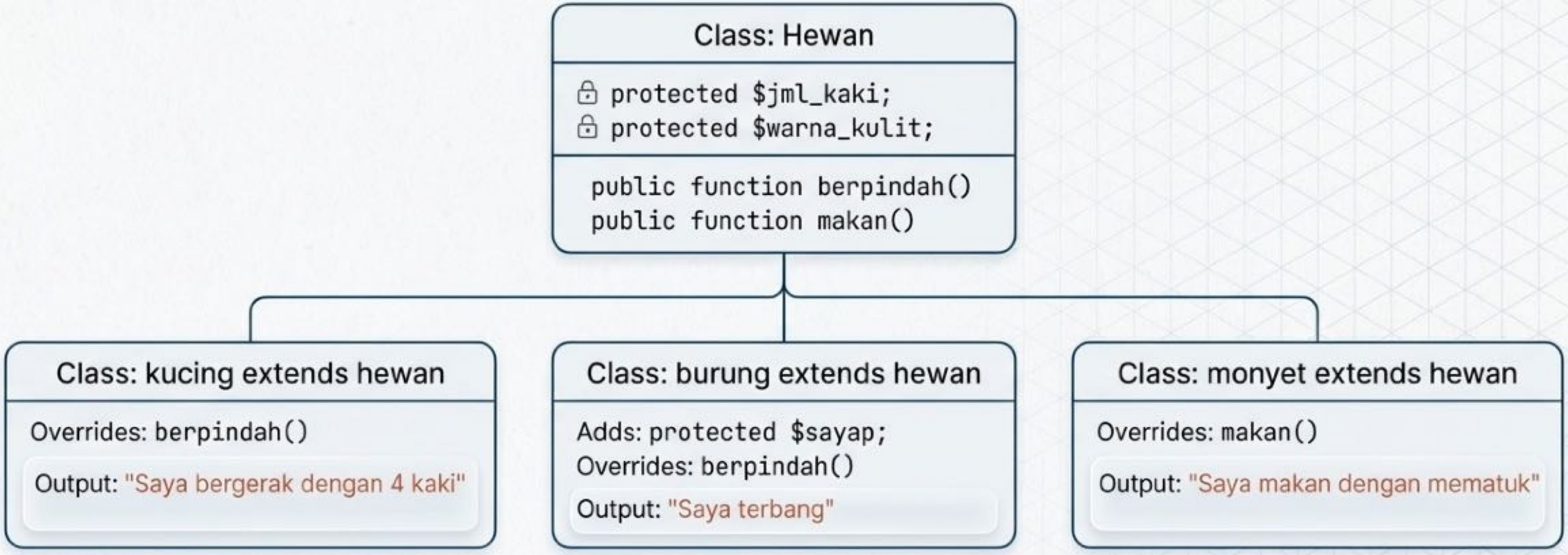
## Sintaksis

Diimplementasikan secara drastis menyederhanakan kode menggunakan keyword extends.



# Pillar 3: Inheritance (The extends Keyword)

**Concept:** A child class inherits the state and behavior of a parent class, eliminating redundant code.



# Membangun Hierarki dalam Kode

Parent Class (Hewan)

```php
<?php
class hewan {
    protected $jml_kaki;
    protected $warna_kulit;

    function __construct() {
    }

    function berpindah() {
        echo 'Saya berpindah';
    }
}
?>
```

Child Classes (Kucing & Burung)

```php
<?php
class kucing extends hewan {
    function berpindah() {
        echo 'Saya bergerak dengan 4 kaki';
    }
}

class burung extends hewan {
    protected $sayap;

    function berpindah() {
        echo 'Saya terbang';
    }
}
?>
```

Perhatikan bagaimana class anak meng-override method berpindah() untuk menghasilkan perilaku unik, sambil tetap berbagi properti dasar dari class hewan.

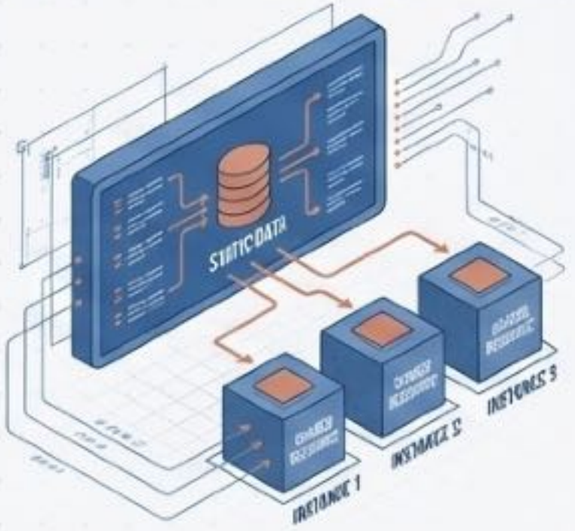
# Aturan Tetap & Sumber Daya Bersama

## Konstanta Class (const)


- Nilai tetap absolut yang tidak dapat diubah setelah dideklarasikan.
- Dideklarasikan tanpa menggunakan tanda $.
- Akses internal menggunakan sintaks self::

```
const constant = 'constant value';
echo self::constant;
```

## Keyword Static (static)




- Property atau method yang menjadi milik Class itu sendiri, bukan milik instansi objek.
- Dapat dipanggil dan diakses langsung tanpa harus melakukan inisiasi objek (tanpa keyword new).

```
public static $my_static = 'foo';
return self::$my_static;
```

# Absolute Immutability: Class Constants

Concept: Data that forms the fixed basis of a class and must never be altered during the application's lifecycle.

## Standard Variables (Mutable)

```
public $tax_rate = 0.1;
```

Can be overwritten by any object.

## Class Constants (Immutable)

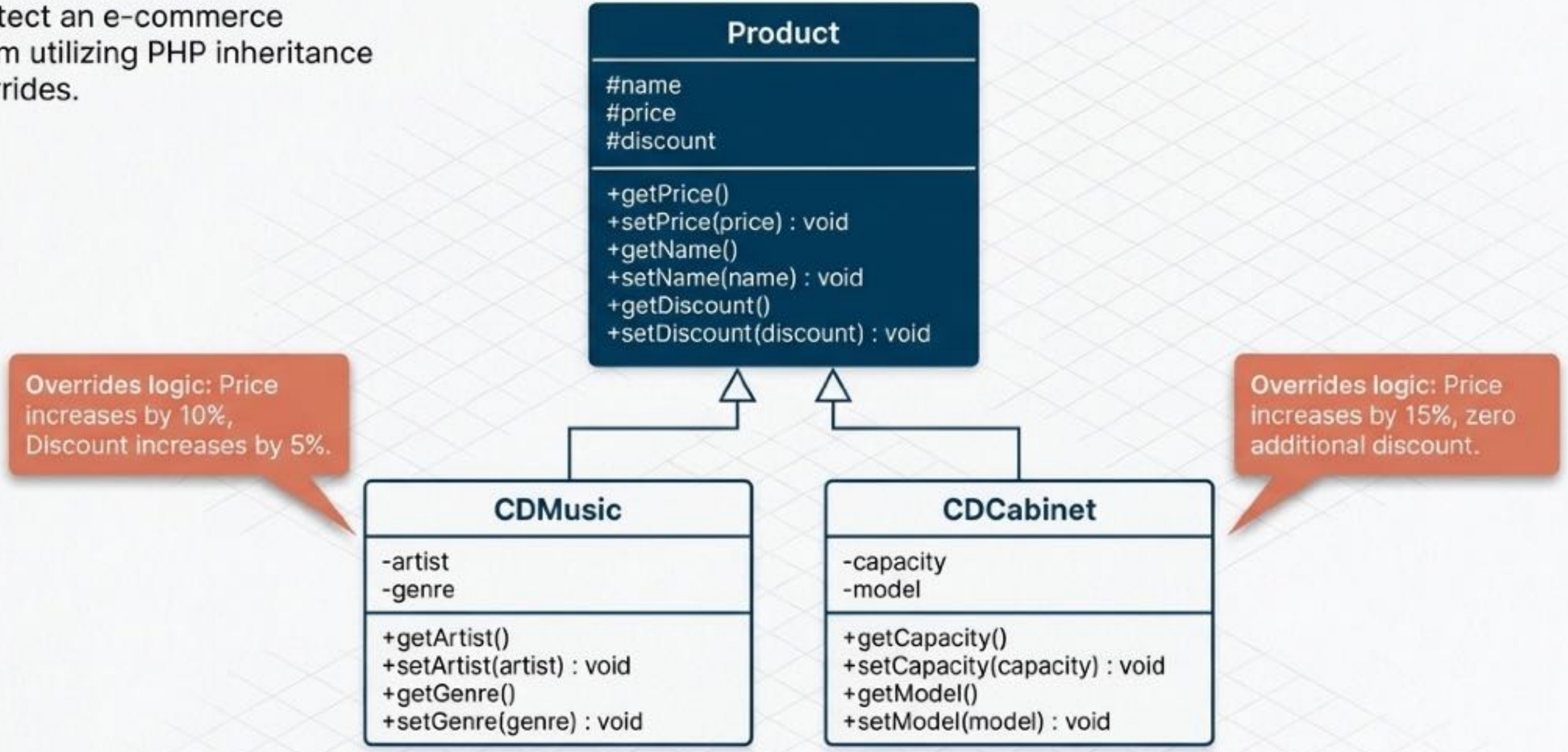
```
const TAX_RATE = 0.1;
```

Defined without a '$' symbol. Locked forever upon declaration.

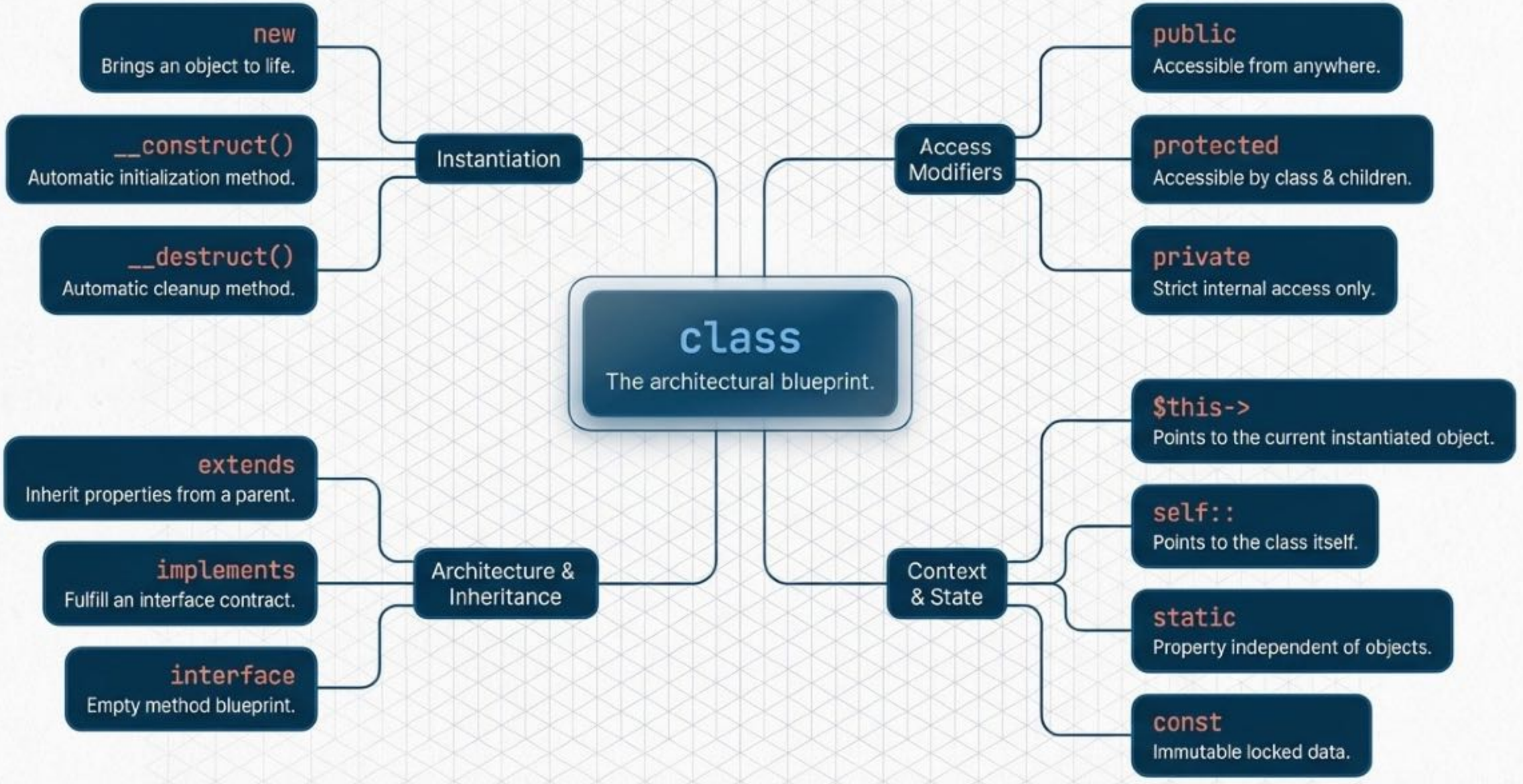
```
class MyClass {
    const CONSTANT = 'constant value';

    function showConstant() {
        echo self::CONSTANT;
    }
}

// Accessing externally:
echo MyClass::CONSTANT;
```

# Putting It Into Practice: System Architecture Challenge

The Task: Architect an e-commerce inventory system utilizing PHP inheritance and logical overrides.



# The Complete PHP OOP Blueprint (Cheat Sheet)



# Fitur Arsitektur Lanjutan: Static & Constants

## Constants (Konstanta)

- Nilai absolut yang tidak bisa diubah.
- Deklarasi tanpa menggunakan tanda **$**.
- Akses dari dalam class: **self::nama_konstanta**

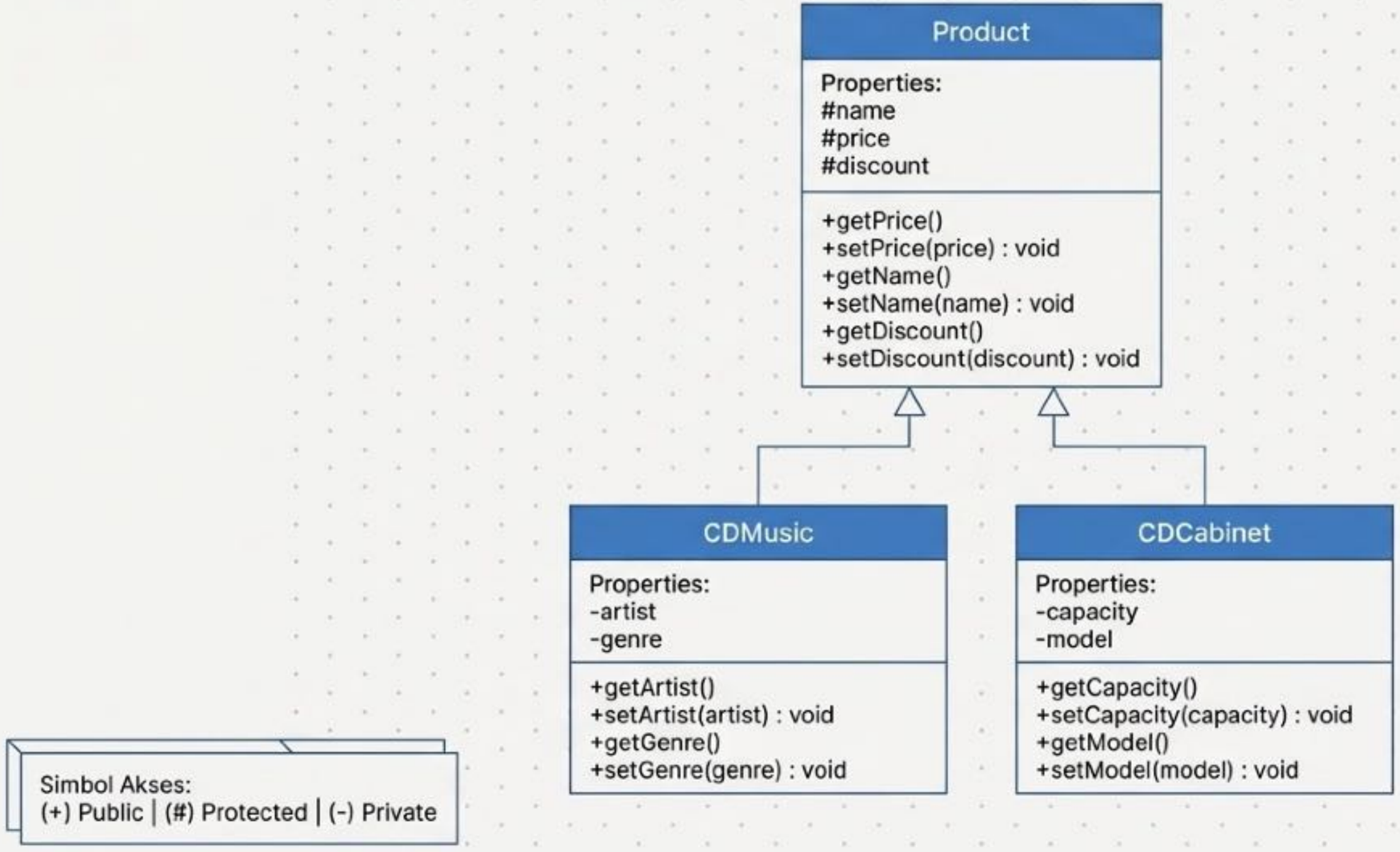
```php
<?php
class MyClass {
    const constant = 'constant value';
    function showConstant() {
        echo self::constant . "\n";
    }
}
echo MyClass::constant . "\n";
?>
```

## Keyword Static

- Menjadikan properti/method milik class itu sendiri, bukan milik objek.
- **TIDAK** perlu instansiasi (new) untuk memanggilnya.
- Akses dari luar class: **NamaClass::$properti**

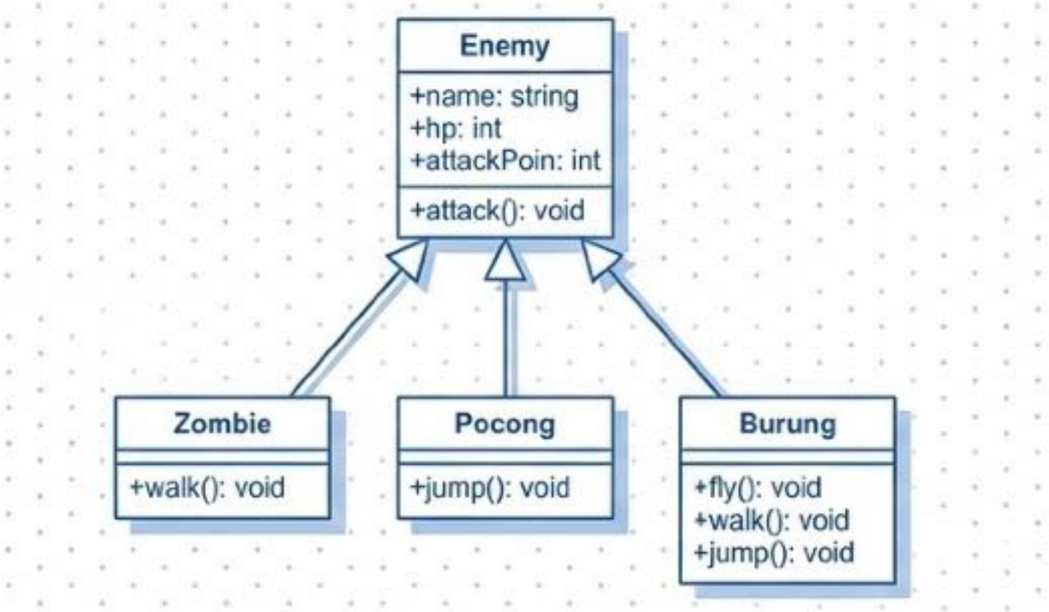
```php
<?php
class Foo{
    public static $my_static = 'foo';
    public function staticValue() {
        return self::$my_static;
    }
}
class Bar extends Foo{
    public function fooStatic() {
        return parent::$my_static;
    }
}
```

# Sintesis: Mengubah Konsep Menjadi Arsitektur E-Commerce



# Tantangan Praktek: Buktikan Konsep Anda

## Tugas #2a: Arsitektur Ekosistem




**Instruksi:** Implementasikan Class Diagram Enemy di atas menggunakan sintaks OOP PHP yang valid. Perhatikan property access dan inheritance.

## Tugas #2b: Business Logic Inheritance

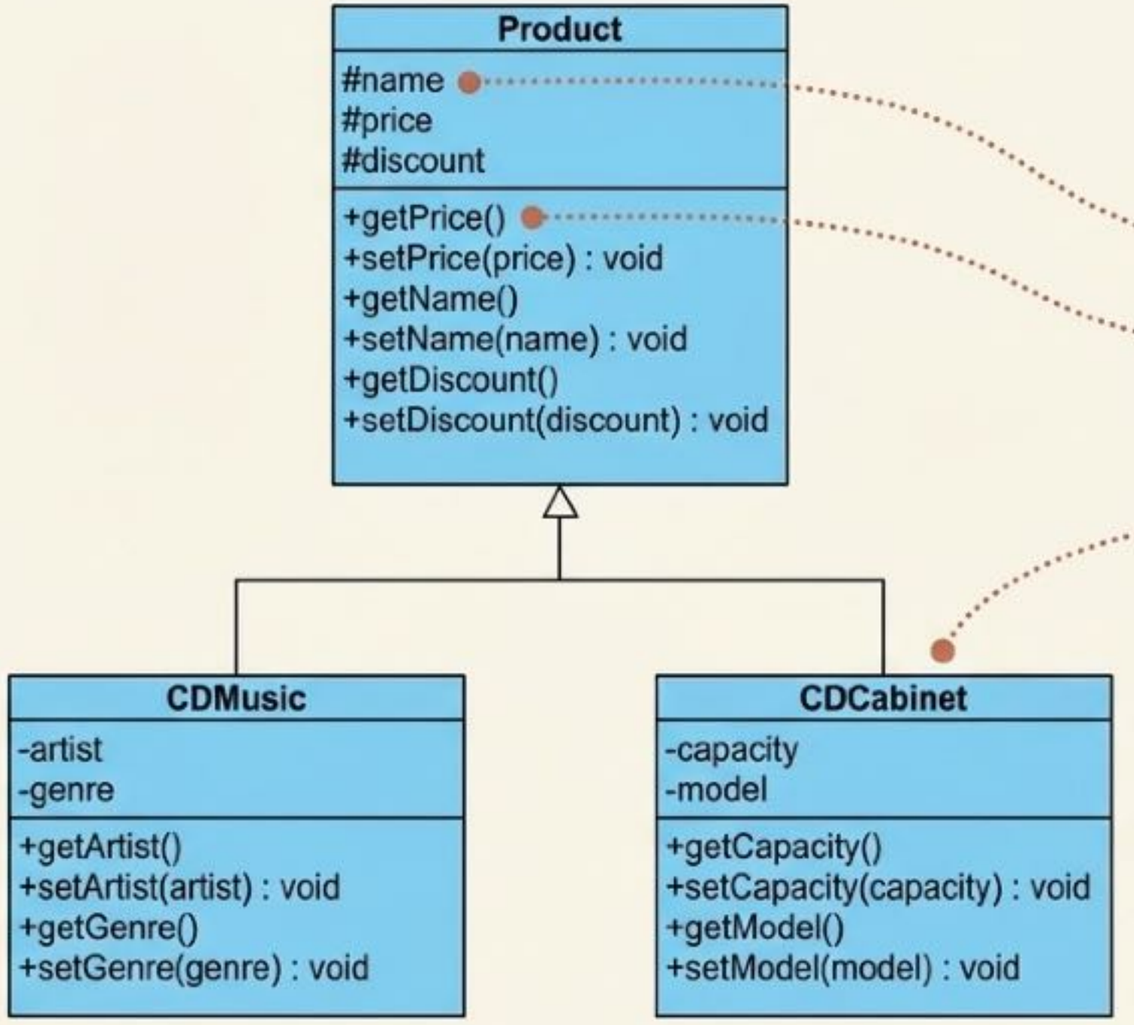
Kembangkan class Product menjadi sistem utuh!

- **CDMusic**: Menuruni atribut dari Product. Implementasikan modifikasi Price = price + 10% dan penambahan discount sebesar 5%.

- **CDRack**: Menuruni atribut dari Product. Implementasikan modifikasi Price = price + 15%. Tidak ada penambahan discount.

**Goal Akhir**: Buatlah eksekusi kode PHP dan jalankan simulasi dari kasus arsitektur di atas.

# Studi Kasus: Translasi UML ke PHP

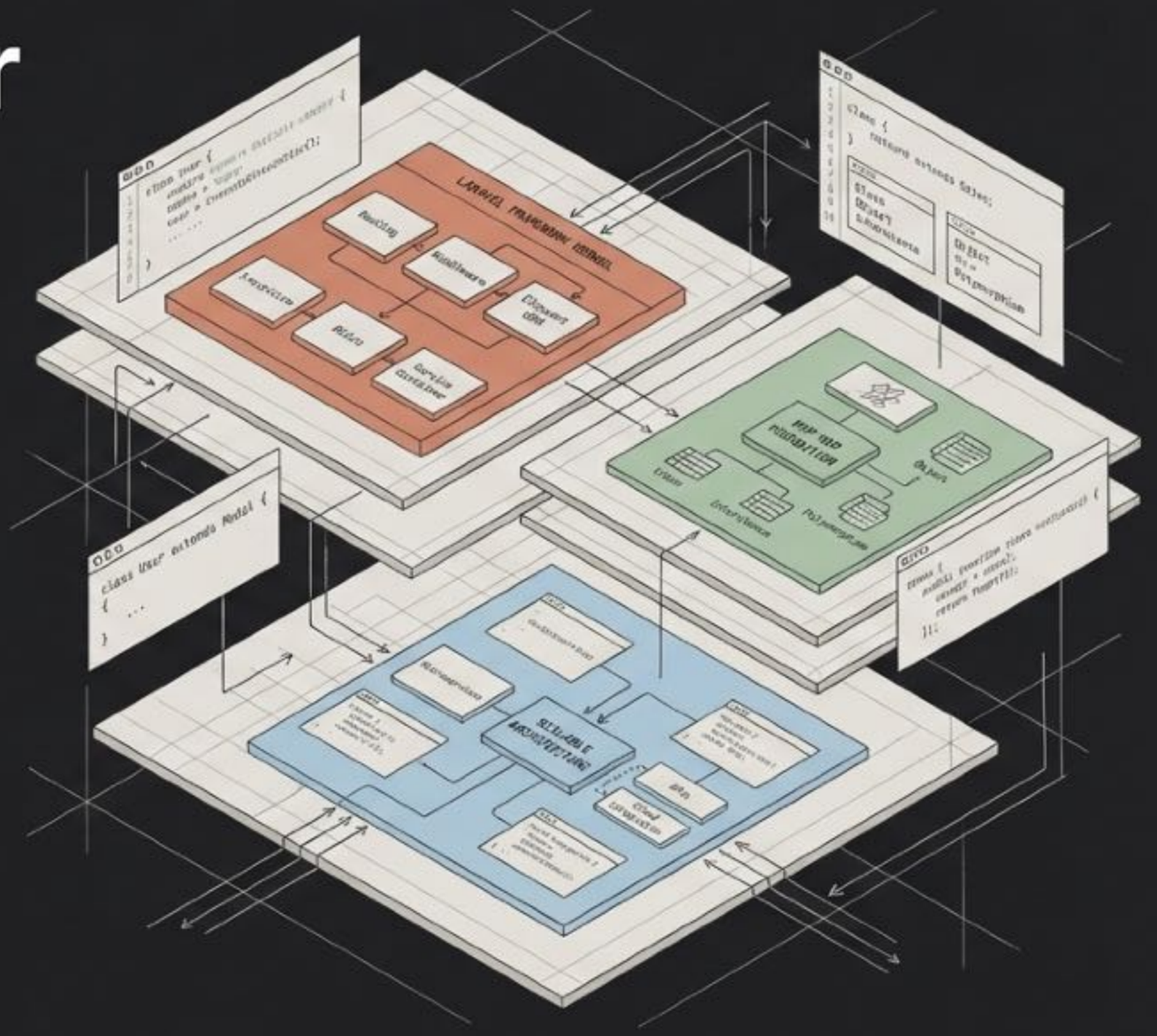



PHP Code Blueprint

```php
class Product {
    protected $name;

    public function getPrice() { ... }
}

class CDMusic extends Product {
    private $artist;
}
```

# Fondasi Arsitektur OOP: Menguasai PHP & Laravel

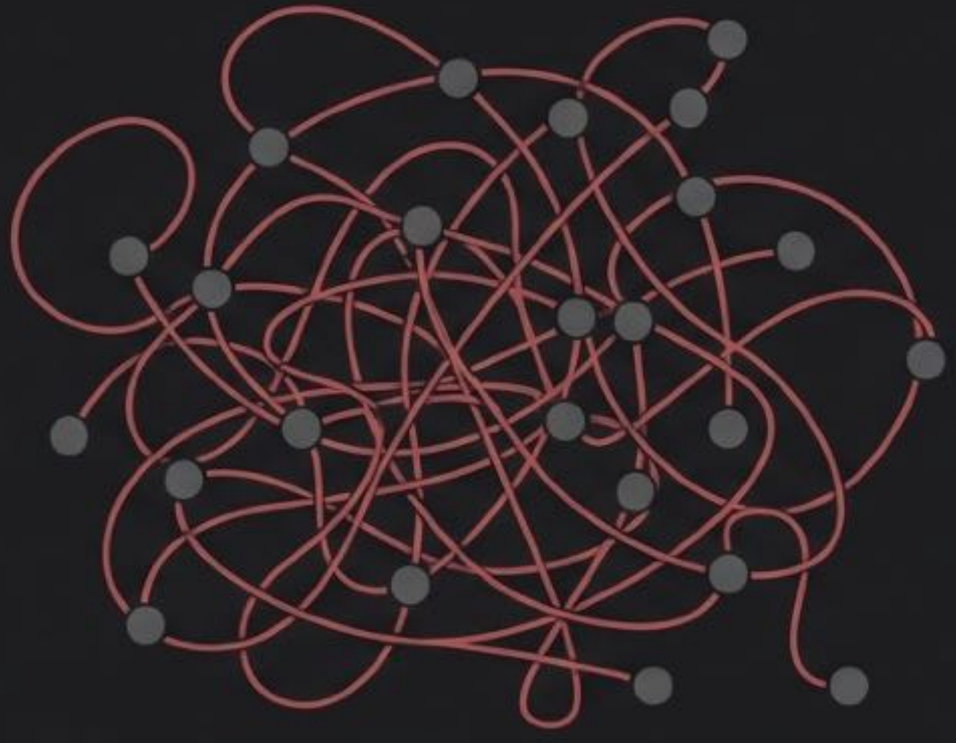
4 Pilar Pemrograman Berorientasi Objek untuk Skalabilitas dan Clean Code



# Mengapa Kita Membutuhkan Paradigma OOP?

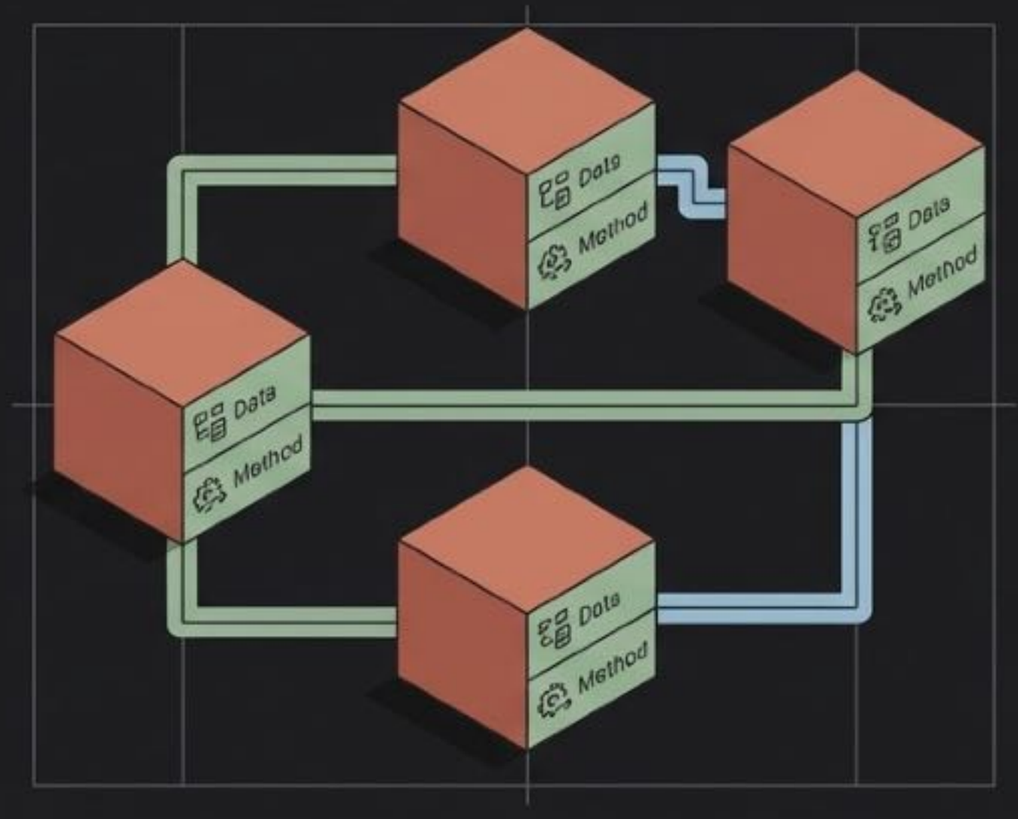
## Pemrograman Prosedural

Fokus pada fungsi berurutan. Rentan terhadap pengulangan kode tak berujung (Spaghetti Code).



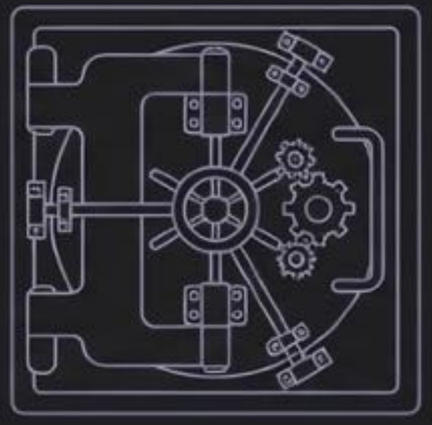
## Object-Oriented Programming

Fokus pada OBJEK (Data + Method). Menjaga kode tetap DRY (Don't Repeat Yourself), terstruktur, dan mudah dipelihara.



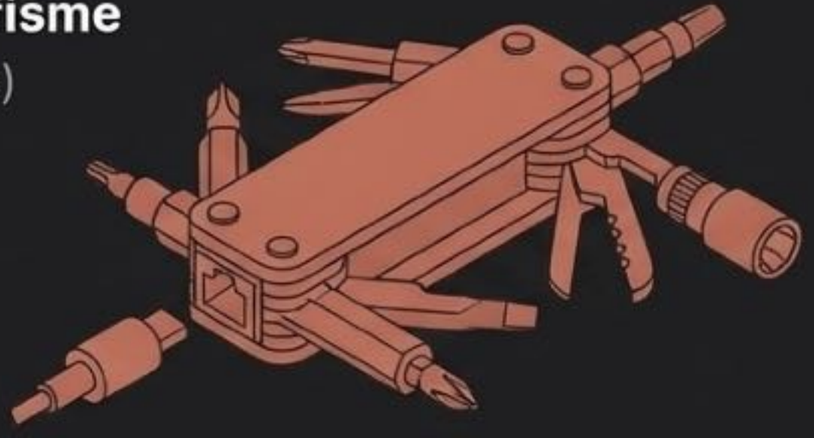
# 4 Pilar Utama Arsitektur Objek

**1. Enkapsulasi**
(Keamanan Data)

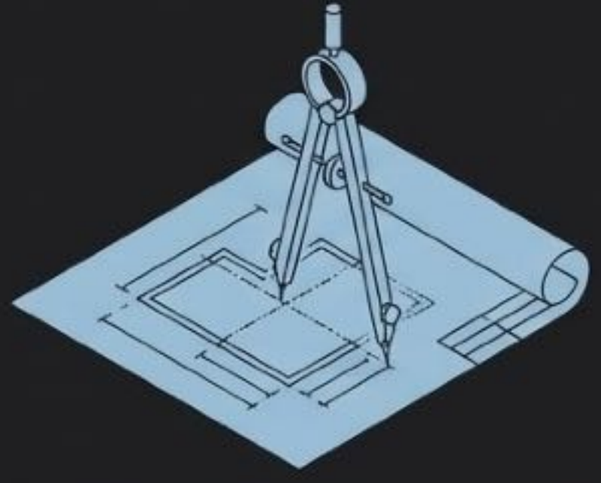


**2. Pewarisan**
(Reusabilitas)

**3. Polimorfisme**
(Fleksibilitas)

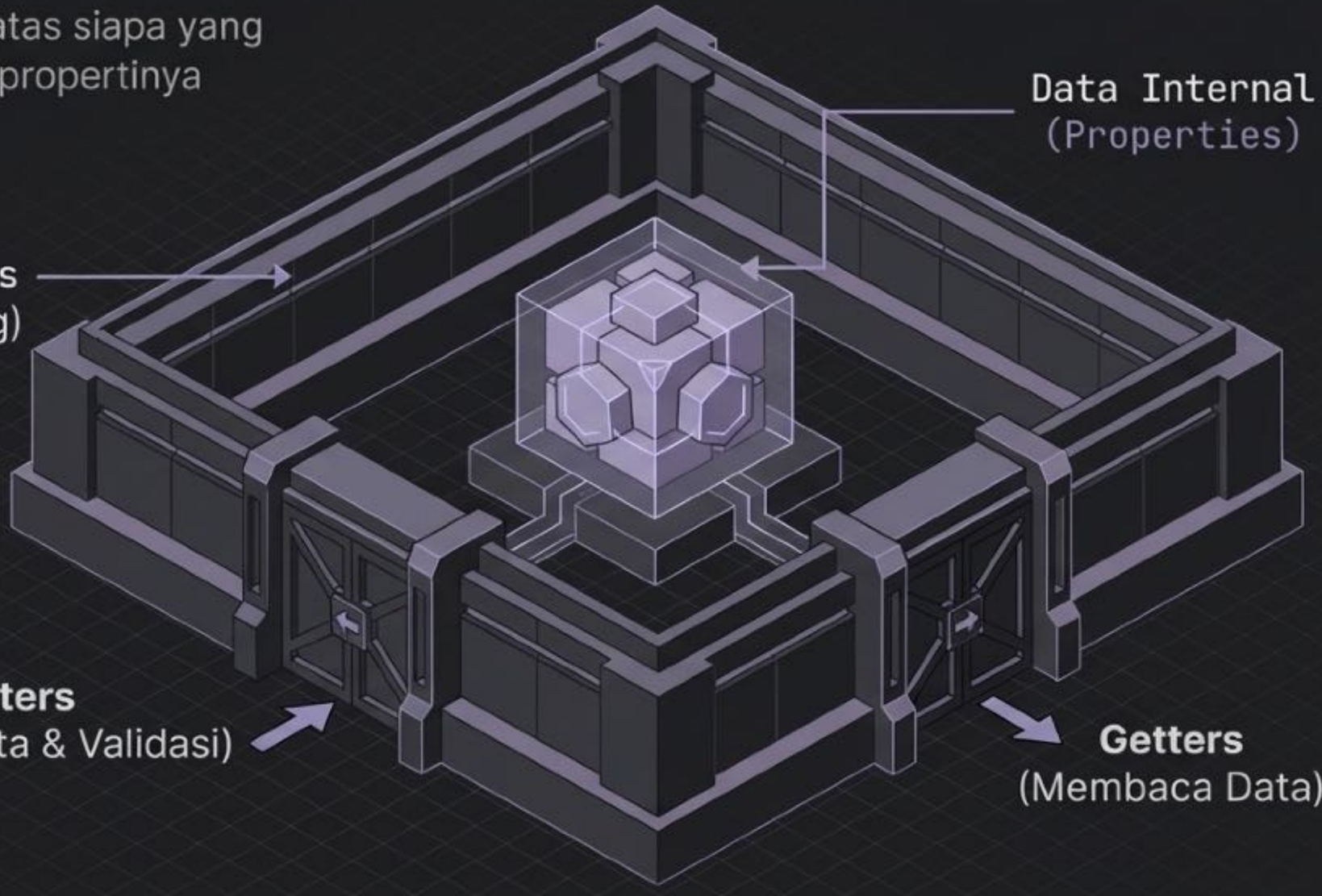


**4. Abstraksi**
(Penyederhanaan)



# Pilar 1: **Enkapsulasi** Melindungi Data Internal

Objek memiliki kendali penuh atas siapa yang boleh melihat atau mengubah propertinya melalui fungsi pelindung.



# Hak Akses: Siapa yang Boleh Membaca dan Mengubah Data?

| Modifier | Dari Class Sendiri | Dari Class Turunan | Dari Luar Class |
|---|---|---|---|
| **Public** (Terbuka Bebas) | ✓ | ✓ | ✓ |
| **Protected** (Akses Hierarki) | ✓ | ✓ | ✗ |
| **Private** (Sangat Rahasia) | ✓ | ✗ | ✗ |

# Enkapsulasi di Dunia Nyata: Keamanan Model Laravel

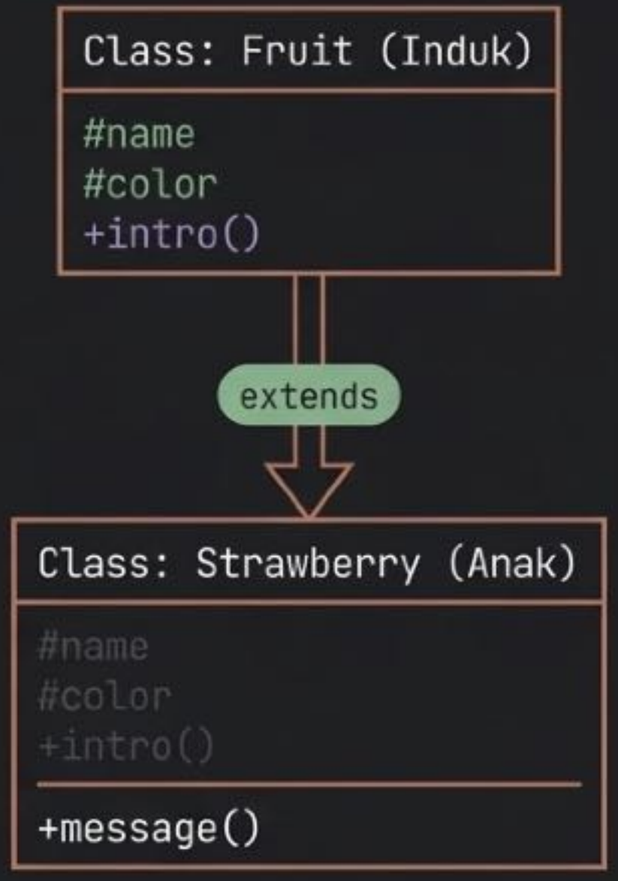
Mencegah Mass-Assignment Vulnerability menggunakan daftar putih.

```php
class User extends Model {

    // Daftar putih atribut yang boleh diisi secara massal
    protected $fillable = ['name', 'email', 'password'];
}
```

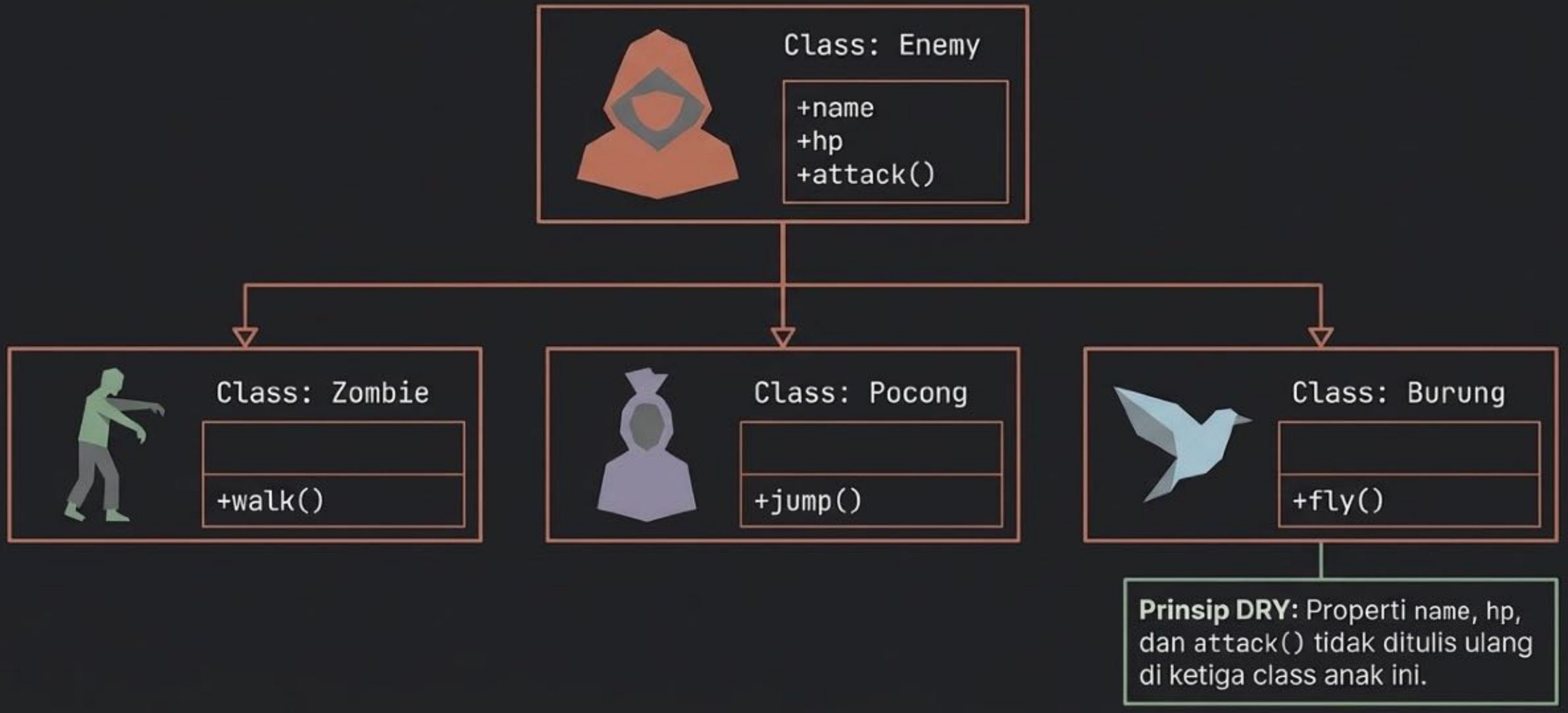
Hanya properti dalam array ini yang dapat disisipkan secara eksternal. Semua data lain diabaikan oleh sistem.

# Pilar 2: Pewarisan Menghilangkan Redundansi Kode

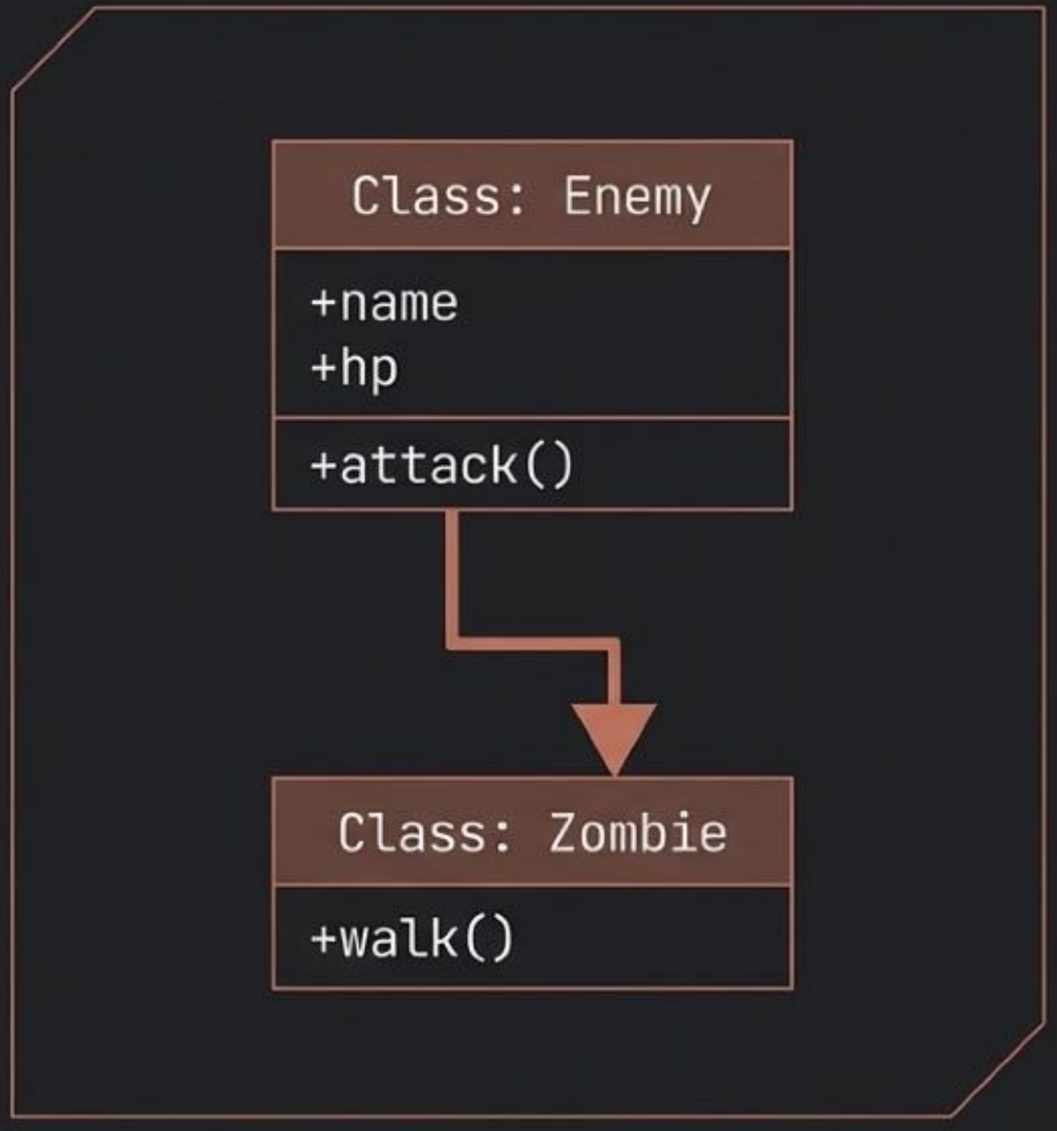
Mewarisi properti dan method secara otomatis tanpa menulis ulang kode yang sama.



# Studi Kasus: Membangun Hierarki Class Musuh Game



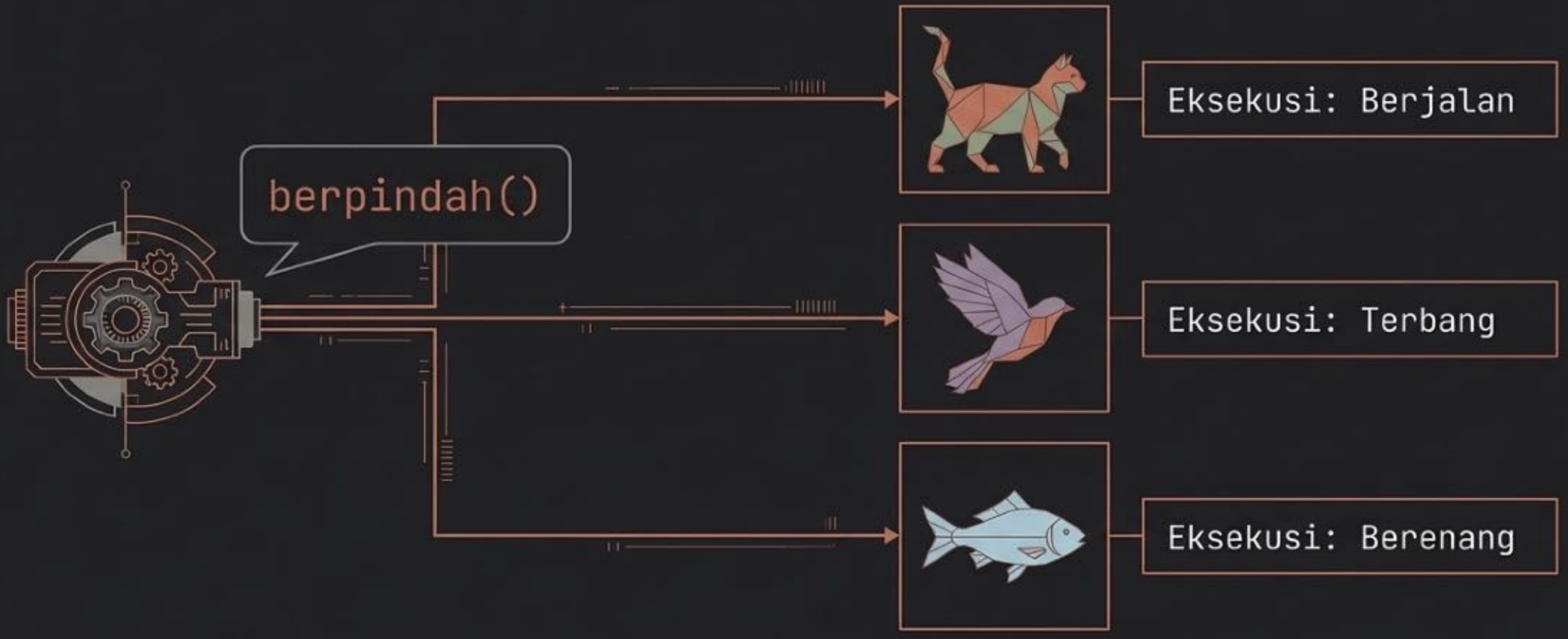
# Menerjemahkan Blueprint Pewarisan ke dalam PHP




```php
<?php

class Enemy {
    public $name;
    public $hp;
    public function attack() {
        // Logika serang...
    }
}

class Zombie extends Enemy {
    public function walk() {
        // Logika berjalan...
    }
}
```

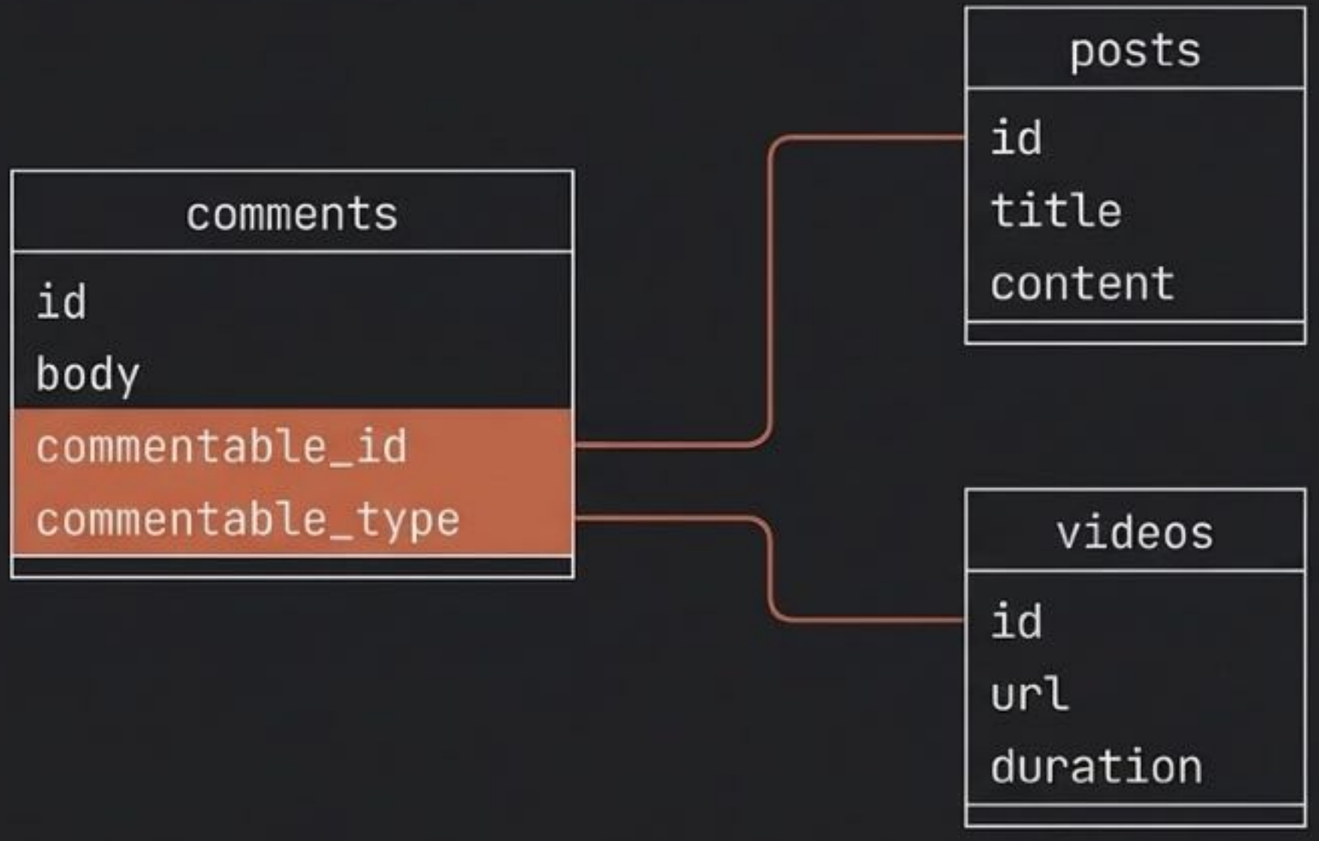
# Pilar 3: Polimorfisme Memberikan Respons Fleksibel

Satu perintah. Banyak bentuk. Implementasi unik untuk setiap objek.



# Polimorfisme pada Skema Database Laravel

Satu tabel mengabdi pada banyak tabel lain secara dinamis tanpa mengubah struktur dasar.



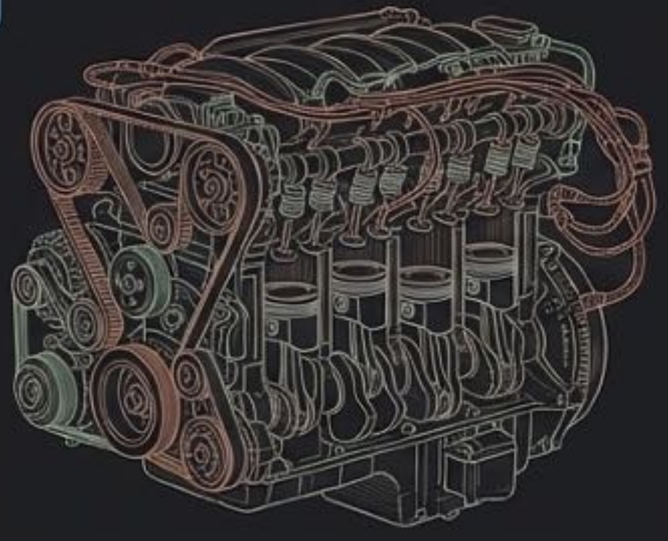

Relasi Polymorphic: Menghindari pembuatan tabel 'post_comments' dan 'video_comments' secara terpisah.

# Pilar 4: Abstraksi Menyembunyikan Kompleksitas Sistem

Bertindak sebagai cetakan utama. Menyembunyikan implementasi backend yang rumit, menampilkan antarmuka yang bersih.

**Interface** (Yang Dilihat User / Front-end)

Implementasi Detail (Disembunyikan / Backend)

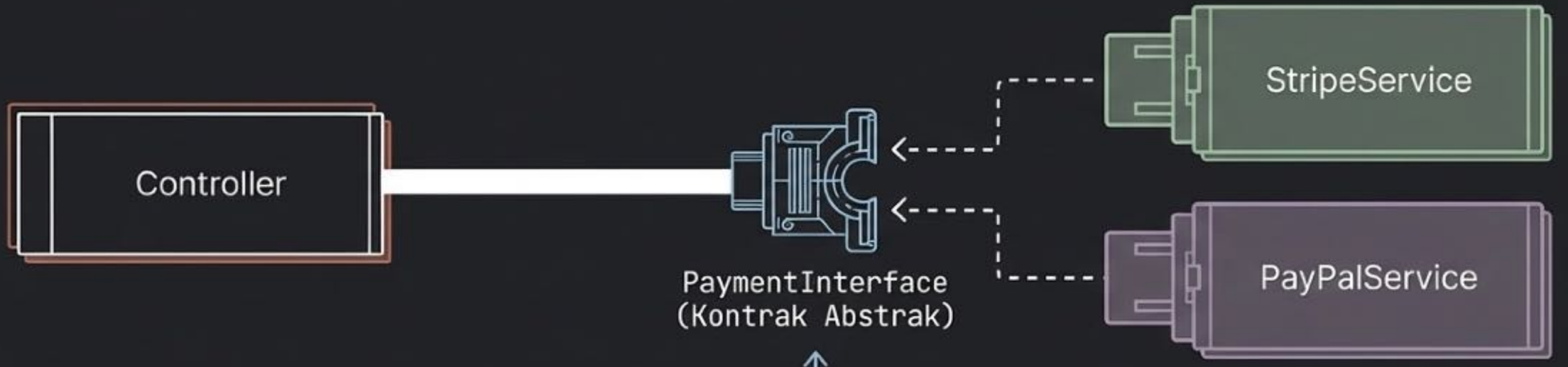


Keyword `'abstract'` memaksa standarisasi.

Class induk hanya menyediakan blueprint, memaksa class anak mengurus detail rumit mesinnya.

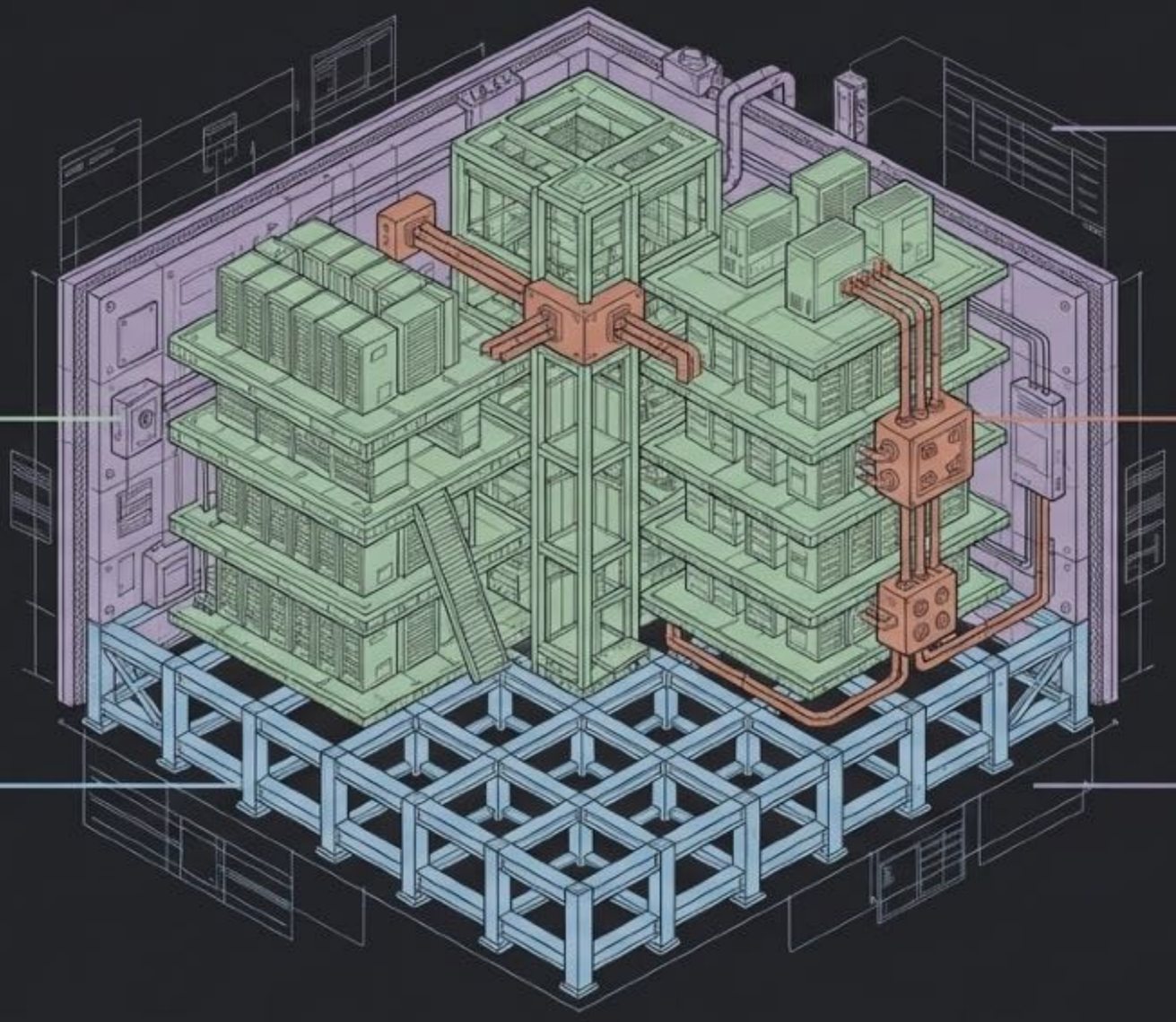
# Abstraksi sebagai Mesin Utama Arsitektur Laravel

Service Container menggunakan Interfaces untuk menukar komponen tanpa merusak ekosistem aplikasi.



# Gambaran Besar: Ekosistem Arsitektur OOP

Sebuah sintesis menyeluruh dari empat pilar utama yang membentuk struktur dan evolusi sistem perangkat lunak modern.



**Enkapsulasi:** Dinding pelindung yang menjaga integritas dan keamanan data internal.

**Pewarisan:** Struktur yang berevolusi dan tumbuh tanpa harus membangun dari nol.

**Polimorfisme:** Soket universal yang berinteraksi secara fleksibel dengan berbagai modul.

**Abstraksi:** Cetakan utama yang menstandarisasi seluruh bentuk bangunan.

**Enkapsulasi:** Dinding pelindung yang menjaga integritas dan keamanan data internal.

# Kode yang Baik Adalah Arsitektur yang Baik

Menguasai OOP bukan sekadar menghafal aturan syntax. Ini adalah pergeseran mindset untuk merancang sistem yang terstruktur, aman, dan siap berekspansi di standar industri modern.

# Status Quo: OOP Sebagai Standar Industri Modern


## Bukan Sekadar Pilihan

OOP di PHP bukan lagi sekadar gaya penulisan alternatif, melainkan standar arsitektur wajib dalam industri perangkat lunak global.

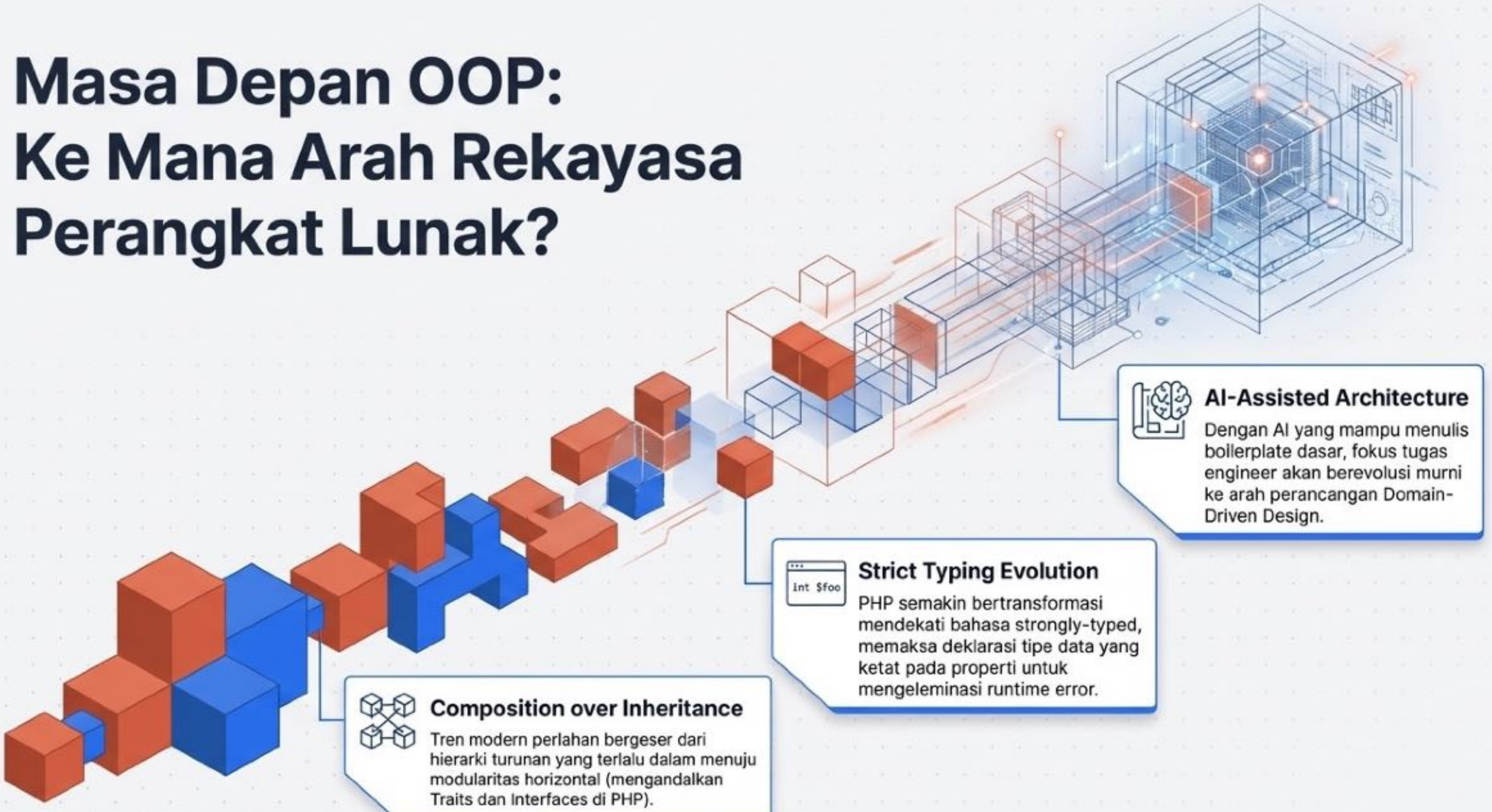
## Dominasi Framework

Seluruh framework modern kelas enterprise (seperti Laravel dan Symfony) beroperasi murni menggunakan arsitektur berbasis Class dan Objek.

## Ekosistem Modular

Ekosistem pengembangan saat ini (khususnya manajer paket Composer) mutlak bergantung pada tingkat modularitas dan reusability yang hanya ditawarkan oleh pola desain OOP.

# Masa Depan OOP: Ke Mana Arah Rekayasa Perangkat Lunak?



### AI-Assisted Architecture

Dengan AI yang mampu menulis bollerplate dasar, fokus tugas engineer akan berevolusi murni ke arah perancangan Domain-Driven Design.

### Strict Typing Evolution

PHP semakin bertransformasi mendekati bahasa strongly-typed, memaksa deklarasi tipe data yang ketat pada properti untuk mengeleminasi runtime error.

### Composition over Inheritance

Tren modern perlahan bergeser dari hierarki turunan yang terlalu dalam menuju modularitas horizontal (mengandalkan Traits dan Interfaces di PHP).

# Matriks Evaluasi: Kelebihan & Kekurangan OOP

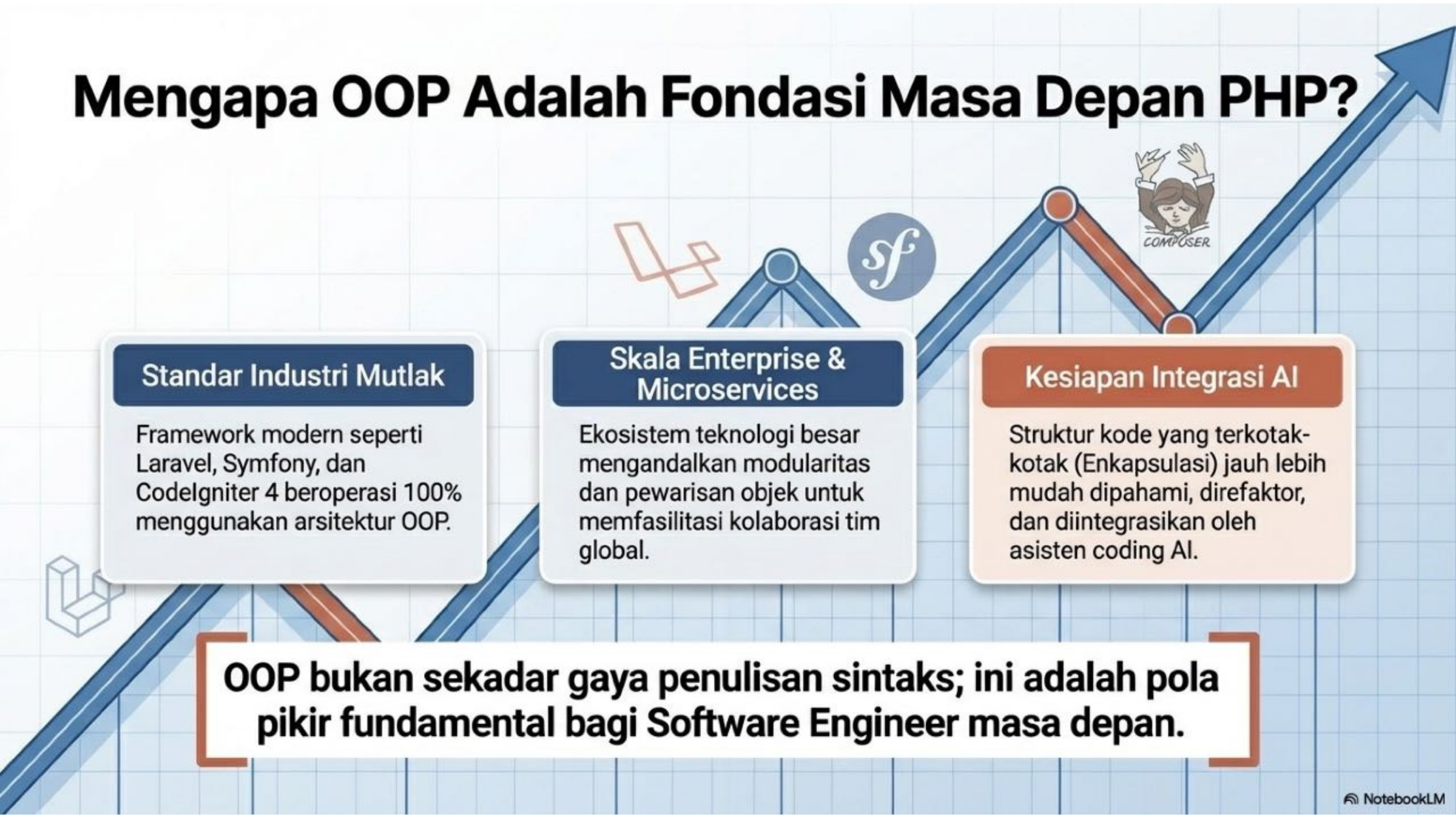
## Kelebihan (Pros)

- **Skalabilitas**: Sangat ideal untuk aplikasi berukuran masif (Enterprise).
- **Modularitas**: Kode dapat diisolasi, di-reuse, dan di-maintain dalam tim besar tanpa tumpang tindih.
- **Keamanan Data**: Enkapsulasi mencegah perubahan state secara brutal dari luar.

## Kekurangan (Cons)

- **Learning Curve**: Membutuhkan pemahaman arsitektur tinggi dibandingkan menulis skrip prosedural sebaris.
- **Overhead**: Memperbesar ukuran file program dan bisa sedikit lebih lambat untuk tugas-tugas matematis yang sangat sederhana.

# Mengapa OOP Adalah Fondasi Masa Depan PHP?




### Standar Industri Mutlak

Framework modern seperti Laravel, Symfony, dan CodeIgniter 4 beroperasi 100% menggunakan arsitektur OOP.

### Skala Enterprise & Microservices

Ekosistem teknologi besar mengandalkan modularitas dan pewarisan objek untuk memfasilitasi kolaborasi tim global.

### Kesiapan Integrasi AI

Struktur kode yang terkotak-kotak (Enkapsulasi) jauh lebih mudah dipahami, direfaktor, dan diintegrasikan oleh asisten coding AI.

**OOP bukan sekadar gaya penulisan sintaks; ini adalah pola pikir fundamental bagi Software Engineer masa depan.**

# Dari Penulis Kode Menjadi Arsitek Sistem

Pemrograman Berorientasi Objek adalah pergeseran mental. Ia memaksa kita berhenti berpikir layaknya mesin yang sekadar mengeksekusi urutan perintah, dan mulai berpikir layaknya seorang arsitek yang merancang ekosistem digital yang tangguh, scalable, dan siap menghadapi masa depan.

**Pahami konsepnya. Rancang blueprint-nya. Bangun masa depan. Terima Kasih!**