

PRAKTIKUM MATA KULIAH PEMROGRAMAN BERORIENTASI OBJEK C

Dosen Pengampu	Syahru Rahmayuda, S.Kom., M.Kom.
Hari/Tanggal	Jumat, 3 Oktober 2025
Nama Mahasiswa	Frans Maylandgo Saragih
NIM	H1101241059

UJIAN TENGAH SEMESTER

Instruksi:

- Download Lembar Soal/Jawaban ini ke dalam format **.docx**.
- Tidak boleh pakai Generative AI, tapi anda boleh mendownload cheat sheet atau lihat kode proyek PHP yang pernah dibuat di komputer masing-masing. Akses internet hanya diberikan untuk download dokumen ini dan upload jawaban ke classroom/edlink.
- Lembar jawaban ini digunakan untuk menjawab **soal penjelasan** dan **hasil tangkapan layar output di terminal**.
- Kompres format **.zip** untuk; **Lembar Jawaban** dan **kode** output ke pengumpulan di Classroom dan Edlink.

Tips biar cepat: Simpan file dalam format nomor (contoh soal 1 -> **1.php**), pelajari juga shortcut keyboard **alt+tab** untuk pindah window dan shortcut keyboard VSCode untuk *insertion* dan *selection*.

Soal Penjelasan, Modifikasi, & Melengkapi Kode

1. Perhatikan kode berikut dan coba jalankan pada perangkat Anda! (10%)

```
<?php
class AkunBank {
    public $saldo = 0;
}

$akun = new AkunBank();
$akun->saldo = -5000;
echo "Saldo: " . $akun->saldo;
```

- a. Menurut Anda, apakah kode tersebut sudah tepat menerapkan teori enkapsulasi pada OOP? Tidak, karena program adalah tentang saldo dari suatu akun bank, sedangkan saldo adalah public, otomatis salah.

- b. Ubah kode tersebut agar saldo **tidak bisa diakses langsung** dan hanya bisa dimodifikasi melalui method **setor()** untuk memasukkan saldo, **tarik()** untuk menarik saldo dengan mengurangi saldo yang ada dengan saldo yang ingin ditarik, dan **getSaldo()** untuk menampilkan jumlah saldo yang ada.

2. Perhatikan kode berikut! (10%)

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

class Anjing {
    public function suara() {
        return "Guk Guk!";
    }
}
```

- a. Apa kesalahan konsep OOP pada kode di atas terkait **inheritance**? Tidak ada extends.
- b. Perbaiki kode agar class **Anjing** merupakan turunan dari **Hewan** dan buatlah sebuah objek dari class tersebut dengan penentuan nama ketika class di-instansiasi serta method **suara()** harus mengembalikan nama hewan dan suara.

3. Perhatikan kode berikut dan coba jalankan pada perangkat Anda! (10%)

```
<?php
class Kendaraan {
    public function jalan() {
        echo "Kendaraan sedang berjalan";
    }
}

class Mobil {
    public function jalan() {
        return "Mobil sedang berjalan di jalan raya";
    }
}

class Motor {
    public function jalan() {
        return "Motor berjalan di jalan raya";
    }
}

$kendaraan1 = new Mobil();
$kendaraan2 = new Motor();

echo "=== OUTPUT HASIL ===\n";
echo "1. " . $kendaraan1->jalan() . "\n";
```

```
echo "2. " . $kendaraan2->jalan() . "\n";
?>
```

- a. Apakah kode di atas sudah mencerminkan **Polimorfisme dan Abstraksi** pada konsep OOP? Jelaskan alasannya! Tidak, karena Polimorfisme sendiri 1 metode mempunyai perilaku yang berbeda, sedangkan pada kode method jalan berkali-kali muncul.
- b. Modifikasi kode agar **Kendaraan** menerapkan konsep abstraksi dengan method **jalan()** yang diterapkan oleh kelas turunannya berupa **Mobil** dan **Motor**!

Ikuti instruksi dan selesaikan kode berikut:

4. 4 Pilar OOP (20%)

```
<?php
```

```
// 1. ENCAPSULATION
// Buat class Produk dengan atribut: nama, harga (gunakan access modifier yang sesuai)
// Lengkapi constructor, setter, dan getter
// buat method umum: tampilkanInfo() dengan build in function number_format() untuk format harga
```

```
class Produk {
    // TODO: isi atribut private $nama, $harga

    // TODO: buat constructor dengan parameter ($nama, $harga)

    // TODO: buat getter dan setter untuk masing-masing atribut

    // TODO: buat method umum: tampilkanInfo(), format harga menggunakan number_format() (15200 jadi 15.200)
}
```

```
// 2. INHERITANCE
// Buat class Buku yang mewarisi dari Produk
// Tambahkan atribut baru: penulis
// Override method tampilkanInfo()
```

```
class Buku extend Produk {
    // TODO: buat atribut $penulis

    // TODO: buat constructor (nama, harga, penulis)
```

```

        // TODO: override method tampilkanInfo()
    }

// 3. POLYMORPHISM
// Buat class Elektronik yang juga mewarisi dari Produk
// Override method tampilkanInfo() dengan cara berbeda dari INHERITANCE (pakai
__construct)

class Elektronik extend Produk {
    // TODO: constructor (nama, harga)
    // TODO: override method tampilkanInfo()
}

// 4. ABSTRACTION
// Buat abstract class User dengan method abstract getRole()
// Buat class Admin dan Customer yang meng-extend User dan mengimplementasikan
getRole()

abstract class User {
    protected $username;

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

    // TODO: buat abstract method getRole()
}

class Admin extend User {
    // TODO: implementasikan getRole()
}

class Customer extend User {
    // TODO: implementasikan getRole()
}

// MAIN PROGRAM
$buku1 = new Buku("Pemrograman PHP", 80000, "Budi");
$elektronik1 = new Elektronik("Laptop", 7000000);

echo $buku1->tampilkanInfo() . "\n";
echo $elektronik1->tampilkanInfo() . "\n"

$admin = new Admin("Leo");

```

```
$customer = new Customer("Andi");
```

```
echo $admin->getRole() . "<br>";
```

```
echo $customer->getRole() . "<br>";
```

output yang diharapkan:

Buku: Pemrograman PHP, Penulis: Budi, Harga: Rp 80.000

Elektronik: Laptop, Harga Spesial: Rp 7.000.000

Admin: Leo

Customer: Andi

5. Static & Const (10%)

```
<?php
```

```
class Matematika {  
    // TODO: buat konstanta PI dengan nilai 3.14  
    // TODO: buat atribut static $counter untuk menghitung berapa kali method  
    hitungLuasLingkaran dipanggil
```

```
    // Method untuk menghitung luas lingkaran
```

```
    public static function hitungLuasLingkaran($r) {
```

```
        // TODO: tambahkan counter setiap kali method dipanggil
```

```
        // TODO: gunakan konstanta PI untuk menghitung luas lingkaran
```

```
    }
```

```
    // Method untuk menampilkan berapa kali method dipanggil
```

```
    public static function getCounter() {
```

```
        // TODO: kembalikan nilai $counter
```

```
    }
```

```
}
```

```
// MAIN PROGRAM
```

```
// Panggil method hitungLuasLingkaran beberapa kali tanpa membuat objek
```

```
echo "Luas lingkaran (r=7): " . Matematika::hitungLuasLingkaran(7) . "\n";
```

```
echo "Luas lingkaran (r=10): " . Matematika::hitungLuasLingkaran(10) . "\n";
```

```
// Tampilkan berapa kali method sudah dipanggil
```

```
echo "Method hitungLuasLingkaran dipanggil sebanyak: " . Matematika::getCounter() .  
" kali\n";
```

Hasil output yang diharapkan:

Luas lingkaran (r=7): 153.86

Luas lingkaran (r=10): 314

Method hitungLuasLingkaran dipanggil sebanyak: 2 kali

6. Trait (10%)

```
<?php
```

```

// 1. Buat trait Pesan dengan method tampilPesan()
//    Method ini menampilkan tulisan "Halo, ini pesan dari trait!"
trait Pesan {
    // TODO: buat method tampilPesan()
}

// 2. Buat trait Logger dengan method log()
//    Method log menerima parameter $pesan dan menampilkan "[LOG]: <pesan>"
trait Logger {
    // TODO: buat method log($pesan)
}

// 3. Buat class User yang menggunakan trait Pesan dan Logger
class User {
    private $nama;

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

    // TODO: gunakan trait Pesan dan Logger

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

// 4. Buat class Admin yang mewarisi User
//    Admin juga menggunakan trait Logger saja
class Admin extends User {
    // TODO: gunakan trait Logger
}

// MAIN PROGRAM
$user1 = new User("Budi");
$admin1 = new Admin("Andi");

// Panggil method dari trait
$user1->tampilPesan();
$user1->log("User " . $user1->getNama() . " berhasil login.");

$admin1->log("Admin " . $admin1->getNama() . " menghapus data.");

```

Hasil output yang diharapkan:

Halo, ini pesan dari trait!

[LOG]: User Budi berhasil login.
[LOG]: Admin Andi menghapus data.

Soal Studi Kasus Mini

7. Kasus 1 (15%)

- Buat kelas **Kamar** dengan atribut **nomorKamar**, **tipeKamar**, dan **harga**. Gunakan **encapsulation** untuk atribut **harga**.
- Buat kelas **Tamu** dengan atribut **nama** dan **idTamu**.
- Buat kelas **Reservasi** yang memiliki hubungan **composition** dengan **Tamu** dan **Kamar**.
- Tambahkan method **hitungTotalBiaya(\$malam)** di **Reservasi**.
- Terapkan **polymorphism** untuk tipe kamar:
 - i. **Deluxe** → Tambahan biaya 20%
 - ii. **Suite** → Tambahan biaya 50%
- Gunakan method **static** untuk menghitung jumlah total reservasi yang sudah dibuat.
- Contoh output program adalah sebagai berikut:

Tamu: Budi (ID: T001)
Kamar: **101** - Deluxe
Jumlah Malam: **3**
Total Biaya: Rp **1.800.000**

Tamu: Sari (ID: T002)
Kamar: **202** - Suite
Jumlah Malam: **2**
Total Biaya: Rp **2.400.000**

8. Kasus 2 (15%)

- Buat kelas **Film** dengan atribut **judul**, **kategori (Reguler atau Premium)**, dan **hargaDasar**.
- Buat kelas **Tiket** dengan atribut **film**, **umurPenonton**, dan method **hitungHarga()**.
- Aturan harga:
 - Jika kategori film = **Premium**, harga dasar + 50%.
 - Jika umur penonton < 12 tahun → diskon 30%.
 - Jika umur penonton ≥ 60 tahun → diskon 20%.
- Buat algoritma kondisi untuk menghitung harga final sesuai aturan di atas.
- Implementasikan beberapa contoh pemesanan tiket dan cetak hasilnya.
- Contoh output program:

Film: Avengers (Reguler)
Umur Penonton: **10**
Harga Tiket: Rp **35.000**

Film: Avengers (Reguler)
Umur Penonton: 30
Harga Tiket: Rp 50.000

Film: Avatar 2 (Premium)
Umur Penonton: 65
Harga Tiket: Rp 84.000

Film: Avatar 2 (Premium)
Umur Penonton: 25
Harga Tiket: Rp 105.000