PRAKTIKUM MATA KULIAH PEMROGRAMAN BERORIENTASI OBJEK C	
Dosen Pengampu	Syahru Rahmayuda, S.Kom., M.Kom.
Hari/Tanggal	Jumat, 3 Oktober 2025
Nama Mahasiswa	Lucky Septhiano
NIM	H1101241021
UJIAN TENGAH SEMESTER	

Soal Penjelasan, Modifikasi, & Melengkapi Kode

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

```php
<?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, kode awal tidak menerapkan enkapsulasi karena atribut saldo bisa diakses langsung dari luar class

   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.

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

    public function setor($jumlah) {
        if ($jumlah > 0) {
            $this->saldo += $jumlah;
        }
    }

    public function tarik($jumlah) {
        if ($jumlah > 0 && $jumlah <= $this->saldo) {
            $this->saldo -= $jumlah;
        }
```

```
    }

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

$akun = new AkunBank();
$akun->setor(10000);
$akun->tarik(5000);
echo "Saldo: " . $akun->getSaldo();
?>
```

2. Perhatikan kode berikut! (10%)

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

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

a. Apa kesalahan konsep OOP pada kode di atas terkait **inheritance**?
Kesalahan konsep OOP diatas adalah class Anjing tidak mewarisi class Hewan dan tidak ada hubungan inheritance.

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.

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

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

class Anjing extends Hewan {
    public function suara() {
        return $this->nama . " bersuara: Guk Guk!";
    }
}
```

```
$anjing = new Anjing("Buddy");
echo $anjing->suara();
?>
```

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

```php
<?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, kode awal tidak mencerminkan polimorfisme dan abstraksi karena tidak ada class abstract dan method abstract.

b. Modifikasi kode agar `Kendaraan` menerapkan konsep abstraksi dengan method `jalan()` yang diterapkan oleh kelas turunannya berupa `Mobil` dan `Motor`!

```
<?php
abstract class Kendaraan {
   abstract public function jalan();
}

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

```
class Motor extends Kendaraan {
   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";
?>
```

Ikuti instruksi dan selesaikan kode berikut:

4. 4 Pilar OOP (20%)

```php
<?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 {
```

```php
    // 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);
```

```php
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
```

Jawab :

```php
<?php
// 1. ENCAPSULATION
class Produk {
    private $nama;
    private $harga;

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

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

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

    public function getHarga() {
        return $this->harga;
    }

    public function setHarga($harga) {
        $this->harga = $harga;
    }

    public function tampilkanInfo() {
        return $this->nama . ", Harga: Rp " . number_format($this->harga, 0, ',', '.');
    }
}
```

```
// 2. INHERITANCE
class Buku extends Produk {
    private $penulis;

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

    public function tampilkanInfo() {
        return "Buku: " . $this->getNama() . ", Penulis: " . $this->penulis . ", Harga: Rp " .
number_format($this->getHarga(), 0, ',', '.');
    }
}

// 3. POLYMORPHISM
class Elektronik extends Produk {
    public function __construct($nama, $harga) {
        parent::__construct($nama, $harga);
    }

    public function tampilkanInfo() {
        return "Elektronik: " . $this->getNama() . ", Harga Spesial: Rp " .
number_format($this->getHarga(), 0, ',', '.');
    }
}

// 4. ABSTRACTION
abstract class User {
    protected $username;

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

    abstract public function getRole();
}

class Admin extends User {
    public function getRole() {
        return "Admin: " . $this->username;
    }
}

class Customer extends User {
    public function getRole() {
        return "Customer: " . $this->username;
```

```
    }
}

// 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>";
?>
```

5. Static & Const (10%)

```php
<?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
```

Jawab :

```php
<?php
class Matematika {
    const PI = 3.14;
    private static $counter = 0;

    public static function hitungLuasLingkaran($r) {
        self::$counter++;
        return self::PI * $r * $r;
    }

    public static function getCounter() {
        return self::$counter;
    }
}

// MAIN PROGRAM
echo "Luas lingkaran (r=7): " . Matematika::hitungLuasLingkaran(7) . "\n";
echo "Luas lingkaran (r=10): " . Matematika::hitungLuasLingkaran(10) . "\n";
echo "Method hitungLuasLingkaran dipanggil sebanyak: " . Matematika::getCounter()
. " kali\n";
?>
```

6. Trait (10%)

```php
<?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.
```

Jawab :

```
<?php
// 1. Trait Pesan
trait Pesan {
    public function tampilPesan() {
        echo "Halo, ini pesan dari trait!\n";
    }
}

// 2. Trait Logger
trait Logger {
    public function log($pesan) {
        echo "[LOG]: " . $pesan . "\n";
    }
}

// 3. Class User
class User {
```

```
    use Pesan, Logger;

    private $nama;

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

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

// 4. Class Admin
class Admin extends User {
    use Logger;

    public function __construct($nama) {
        parent::__construct($nama);
    }
}

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

$user1->tampilPesan();
$user1->log("User " . $user1->getName() . " berhasil login.");
$admin1->log("Admin " . $admin1->getName() . " 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
```

Jawab :

```php
<?php
class Kamar {
    private $nomorKamar;
    private $tipeKamar;
    private $harga;

    public function __construct($nomorKamar, $tipeKamar, $harga) {
        $this->nomorKamar = $nomorKamar;
        $this->tipeKamar = $tipeKamar;
        $this->harga = $harga;
    }

    public function getNomorKamar() { return $this->nomorKamar; }
    public function getTipeKamar() { return $this->tipeKamar; }
    public function getHarga() { return $this->harga; }
}

class Tamu {
    private $nama;
    private $idTamu;

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

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

class Reservasi {
    private $tamu;
    private $kamar;
    private static $totalReservasi = 0;

    public function __construct(Tamu $tamu, Kamar $kamar) {
        $this->tamu = $tamu;
        $this->kamar = $kamar;
        self::$totalReservasi++;
    }

    public function hitungTotalBiaya($malam) {
        $hargaDasar = $this->kamar->getHarga() * $malam;
```

```
        switch($this->kamar->getTipeKamar()) {
            case 'Deluxe':
                return $hargaDasar * 1.2;
            case 'Suite':
                return $hargaDasar * 1.5;
            default:
                return $hargaDasar;
        }
    }

    public static function getTotalReservasi() {
        return self::$totalReservasi;
    }

    public function tampilkanInfo($malam) {
        echo "Tamu: " . $this->tamu->getNama() . " (ID: " .
$this->tamu->getIdTamu() . ")\n";
        echo "Kamar: " . $this->kamar->getNomorKamar() . " - " .
$this->kamar->getTipeKamar() . "\n";
        echo "Jumlah Malam: " . $malam . "\n";
        echo "Total Biaya: Rp " . number_format($this->hitungTotalBiaya($malam),
0, ',', '.') . "\n\n";
    }
}

// Contoh penggunaan
$kamar1 = new Kamar(101, "Deluxe", 500000);
$kamar2 = new Kamar(202, "Suite", 800000);

$tamu1 = new Tamu("Budi", "T001");
$tamu2 = new Tamu("Sari", "T002");

$reservasi1 = new Reservasi($tamu1, $kamar1);
$reservasi2 = new Reservasi($tamu2, $kamar2);

$reservasi1->tampilkanInfo(3);
$reservasi2->tampilkanInfo(2);

echo "Total Reservasi: " . Reservasi::getTotalReservasi() . "\n";
?>
```

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
```

Jawab :

```php
<?php
class Film {
    private $judul;
    private $kategori;
    private $hargaDasar;

    public function __construct($judul, $kategori, $hargaDasar) {
        $this->judul = $judul;
        $this->kategori = $kategori;
        $this->hargaDasar = $hargaDasar;
    }

    public function getJudul() { return $this->judul; }
    public function getKategori() { return $this->kategori; }
    public function getHargaDasar() { return $this->hargaDasar; }
}

class Tiket {
    private $film;
    private $umurPenonton;

    public function __construct(Film $film, $umurPenonton) {
        $this->film = $film;
        $this->umurPenonton = $umurPenonton;
    }

    public function hitungHarga() {
        $harga = $this->film->getHargaDasar();

        // Premium +50%
        if ($this->film->getKategori() == 'Premium') {
            $harga *= 1.5;
        }

        // Diskon umur
        if ($this->umurPenonton < 12) {
```

```
            $harga *= 0.7; // Diskon 30%
        } elseif ($this->umurPenonton >= 60) {
            $harga *= 0.8; // Diskon 20%
        }

        return $harga;
    }

    public function tampilkanInfo() {
        echo "Film: " . $this->film->getJudul() . " (" .
$this->film->getKategori() . ")\n";
        echo "Umur Penonton: " . $this->umurPenonton . "\n";
        echo "Harga Tiket: Rp " . number_format($this->hitungHarga(), 0, ',',
'.') . "\n\n";
    }
}

// Contoh penggunaan
$film1 = new Film("Avengers", "Regular", 50000);
$film2 = new Film("Avatar 2", "Premium", 70000);

$tiket1 = new Tiket($film1, 10);
$tiket2 = new Tiket($film1, 30);
$tiket3 = new Tiket($film2, 65);
$tiket4 = new Tiket($film2, 25);

$tiket1->tampilkanInfo();
$tiket2->tampilkanInfo();
$tiket3->tampilkanInfo();
$tiket4->tampilkanInfo();
?>
```