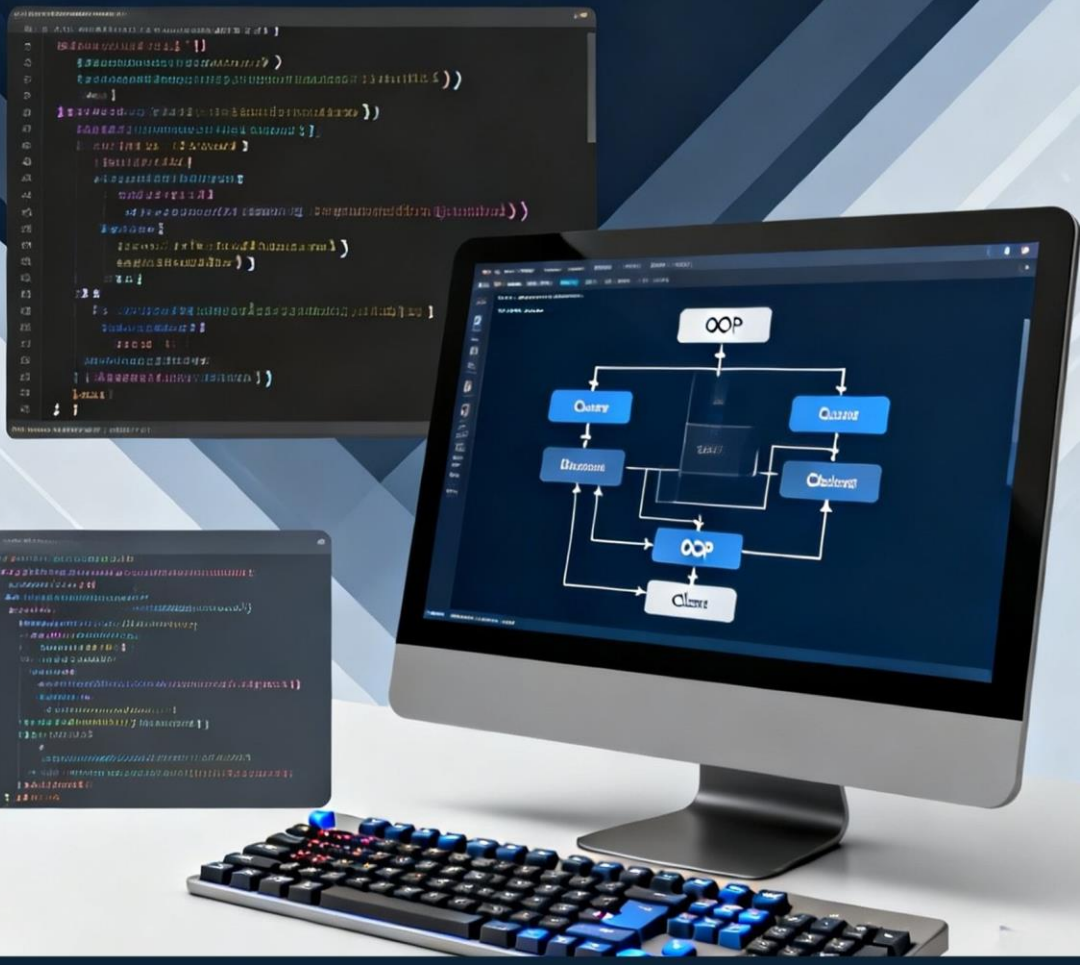
PEMROGRAMAN BERORIENTASI OBJEK DENGAN PHP

Teori dan Praktik



Miftah Farid Adiwisastra - Haerul Fatah - Herlan Sutisna
Bambang Kelana Simpony - Dini Silvi Purnia

PEMROGRAMAN BERORIENTASI OBJEK DENGAN PHP

Teori dan Praktik

UU No 28 tahun 2014 tentang Hak Cipta

Fungsi dan sifat hak cipta Pasal 4
Hak Cipta sebagaimana dimaksud dalam Pasal 3 huruf a merupakan hak eksklusif yang terdiri atas hak moral dan hak ekonomi.

Pembatasan Pelindungan Pasal 26
Ketentuan sebagaimana dimaksud dalam Pasal 23, Pasal 24, dan Pasal 25 tidak berlaku terhadap:

i. penggunaan kutipan singkat Ciptaan dan/atau produk Hak Terkait untuk pelaporan peristiwa aktual yang ditujukan hanya untuk keperluan penyediaan informasi aktual;
ii. Penggandaan Ciptaan dan/atau produk Hak Terkait hanya untuk kepentingan penelitian ilmu pengetahuan;
iii. Penggandaan Ciptaan dan/atau produk Hak Terkait hanya untuk keperluan pengajaran, kecuali pertunjukan dan Fonogram yang telah dilakukan Pengumuman sebagai bahan ajar; dan
iv. penggunaan untuk kepentingan pendidikan dan pengembangan ilmu pengetahuan yang memungkinkan suatu Ciptaan dan/atau produk Hak Terkait dapat digunakan tanpa izin Pelaku Pertunjukan, Produser Fonogram, atau Lembaga Penyiaran.

Sanksi Pelanggaran Pasal 113

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).

PT Insan Cendekia
Mandiri Group

PEMROGRAMAN BERORIENTASI OBJEK DENGAN PHP

Teori dan Praktik

Miftah Farid Adiwisastra - Haerul Fatah - Herlan Sutisna
Bambang Kelana Simpony - Dini Silvi Purnia

PEMROGRAMAN BERORIENTASI OBJEK DENGAN PHP:
Teori dan Praktik

Penulis : **Miftah Farid Adiwisastra, Haerul Fatah, Herlan Sutisna, Bambang Kelana Simpony, Dini Silvi Purnia**
Editor : **Juffi Syahri**
Desain Kover : **www.canva.com**
Tata Letak : **Juffi Syahri**
Ukuran:
x, 200 hlm, 15,5 x 23 cm

ISBN:
978-634-252-250-9

Cetakan Pertama:
Maret 2026

Anggota IKAPI : 020/SBA/20
PENERBIT INSAN CENDEKIA MANDIRI
(PT. INSAN CENDEKIA MANDIRI GROUP)
Jorong Pale, Nagari Pematang Panjang, Kecamatan Sijunjung,
Kabupaten Sijunjung, Provinsi Sumatra Barat – Indonesia 27554
HP/WA: 0813-7272-5118
Website: www.insancendekiamandiri.co.id
E-mail: insancendekiamandirigroup@gmail.com

Perpustakaan Nasional RI : Katalog Dalam Terbitan (KDT)

JUDUL DAN PENANGGUNG JAWAB	Pemrograman berorientasi objek dengan PHP : teori dan praktik / Miftah Farid Adiwisastra, Haerul Fatah, Herlan Sutisna, Bambang Kelana Simpony, Dini Silvi Purnia ; editor, Juffi Syahri
EDISI	Cetakan pertama, Maret 2026
PUBLIKASI	Sijunjung : PT. Insan Cendekia Mandiri Group, 2026
DESKRIPSI FISIK	x, 200 halaman ; 23 cm
IDENTIFIKASI	ISBN 978-634-252-250-9
SUBJEK	PHP (Bahasa pemrograman) Komputer, Pemrograman
KLASIFIKASI	005.133 [23]
PERPUSNAS ID	https://isbn.perpusnas.go.id/bo-penerbit/penerbit/isbn/data/view-kdt/1361079

DAFTAR ISI

PRAKATA

Kami bersyukur kepada Allah Swt., karena atas rahmat dan karunia-Nya buku ajar *"Pemrograman Berorientasi Objek dengan PHP: Teori dan Praktik"* ini dapat disusun dan diselesaikan dengan baik. Buku ajar ini disusun sebagai bahan pendukung pembelajaran bagi mahasiswa, dosen, maupun praktisi yang ingin memahami dan mengimplementasikan konsep Pemrograman Berorientasi Objek (*Object Oriented Programming*/OOP) menggunakan bahasa pemrograman PHP secara sistematis dan aplikatif. Perkembangan teknologi informasi yang sangat pesat menuntut penguasaan paradigma pemrograman yang tidak hanya mampu menghasilkan program yang berjalan dengan baik, tetapi juga mudah dikembangkan, dipelihara, dan diperluas. Pemrograman Berorientasi Objek hadir sebagai solusi untuk membangun perangkat lunak yang modular, terstruktur, dan berorientasi pada pemodelan dunia nyata. PHP sebagai salah satu bahasa pemrograman *server-side* yang populer dan banyak digunakan dalam pengembangan aplikasi web, telah mendukung konsep OOP secara komprehensif, sehingga sangat relevan untuk dipelajari dalam konteks pengembangan aplikasi modern.

Buku ajar ini dirancang dengan pendekatan bertahap, dimulai dari pengenalan konsep dasar OOP, dasar-dasar PHP, hingga implementasi lanjutan seperti enkapsulasi, pewarisan, polimorfisme, pengelolaan *database* berbasis OOP, desain dan arsitektur aplikasi, serta studi kasus dan proyek akhir. Setiap bab dilengkapi dengan tujuan pembelajaran, contoh kode, rangkuman, dan latihan, sehingga diharapkan dapat membantu pembaca memahami materi secara konseptual maupun praktis.

Penulis menyadari bahwa buku ajar ini masih memiliki keterbatasan dan membutuhkan penyempurnaan. Oleh karena itu, kritik dan saran yang bersifat membangun sangat diharapkan demi perbaikan dan pengembangan buku ajar ini di masa mendatang. Akhir kata, penulis berharap buku ajar ini dapat memberikan manfaat dan kontribusi positif dalam proses pembelajaran pemrograman, khususnya dalam penguasaan Pemrograman Berorientasi Objek dengan PHP.

Semoga buku ajar ini dapat digunakan secara optimal dan menjadi referensi yang bermanfaat bagi seluruh pembaca.

Penulis

BAB 1
Pengenalan Pemrograman Berorientasi Objek (OOP)

DESKRIPSI:

Bab ini membahas konsep dasar Pemrograman Berorientasi Objek (OOP), termasuk perbandingannya dengan pemrograman prosedural, manfaatnya, serta prinsip-prinsip utama yang membentuk paradigma OOP. Pemahaman konsep ini sangat penting bagi pengembang perangkat lunak yang ingin mengembangkan aplikasi PHP yang lebih modular, efisien, dan mudah dipelihara.

TUJUAN PEMBELAJARAN:

Setelah mempelajari bab ini, pembaca diharapkan mampu:

1. Menjelaskan konsep dasar Pemrograman Berorientasi Objek (OOP).
2. Membandingkan OOP dengan pemrograman prosedural.
3. Mengidentifikasi manfaat dan keunggulan OOP dalam pengembangan perangkat lunak.
4. Memahami konsep utama dalam OOP seperti *Class*, *Object*, Enkapsulasi, Pewarisan, dan Polimorfisme.
5. Mengimplementasikan konsep dasar OOP dalam PHP melalui contoh sederhana.

A. Apa Itu OOP?

OOP adalah singkatan dari *Object Oriented Programming* atau Pemrograman Berorientasi Objek. OOP adalah paradigma pemrograman yang berfokus pada konsep "objek" sebagai representasi dari dunia nyata. Objek memiliki atribut (data/properti) dan perilaku (metode/fungsi) yang mendukung modularitas, fleksibilitas, dan pemeliharaan kode yang lebih baik. OOP banyak

digunakan dalam pengembangan perangkat lunak modern karena strukturnya yang lebih terorganisir dibandingkan dengan pendekatan prosedural. OOP membantu dalam memecah kode menjadi bagian-bagian yang lebih kecil dan lebih mudah dikelola.

B. Perbandingan OOP dengan Pemrograman Prosedural

Pemrograman prosedural dan OOP adalah dua pendekatan utama dalam menulis kode.

Tabel I.1 Perbandingan Pemrograman Prosedural dan OOP

Aspek	Pemrograman Prosedural	Pemrograman Berorientasi Objek
Pendekatan	Berbasis prosedur atau fungsi	Berbasis objek
Struktur	Kode terorganisir dalam fungsi/prosedur	Kode terorganisir dalam kelas dan objek
Data dan Fungsi	Data dan fungsi terpisah	Data dan fungsi terintegrasi dalam objek
Keamanan Data	Kurang aman, data bisa diakses dari mana saja	Mendukung enkapsulasi untuk keamanan data
Reusability	Kurang mendukung, sering perlu menulis ulang kode	Mendukung *reusability* melalui pewarisan dan polimorfisme

Berikut adalah perbandingan pendekatan prosedural dan OOP dalam PHP menggunakan contoh sederhana pengelolaan data Mahasiswa:

Contoh pendekatan prosedural dengan menggunakan fungsi untuk mengolah data tanpa konsep objek.

```
<?php
 // Data mahasiswa disimpan dalam array asosiatif
    $mahasiswa = [
        "nama" => "Dzimar Rauhillah",
        "nim" => "20231001",
        "jurusan" => "Sistem Informasi"
    ];

    // Fungsi untuk menampilkan data mahasiswa
    function tampilkanMahasiswa($mhs) {
        echo "Nama: " . $mhs["nama"] . "<br>";
```

```
        echo "NIM: " . $mhs["nim"] . "<br>";
         echo  "Jurusan:  "  .  $mhs["jurusan"]  .
"<br>";
    }

 // Panggil fungsi untuk menampilkan mahasiswa
 tampilkanMahasiswa($mahasiswa);
?>
```

Penjelasan kode di atas, yaitu data mahasiswa disimpan dalam *array* asosiatif, fungsi `tampilkanMahasiswa()` digunakan untuk menampilkan data. Tidak ada hubungan langsung antara data dan fungsinya, yang membuat kode kurang modular dan sulit dikembangkan jika ada perubahan struktur data.

Contoh pendekatan OOP dengan membungkus data dan perilaku dalam satu kesatuan yang disebut kelas (*class*) dan digunakan sebagai objek (*object*).

```
<?php
 // Mendefinisikan kelas Mahasiswa
    class Mahasiswa {
        public $nama;
        public $nim;
        public $jurusan;

        // Constructor untuk inisialisasi objek
         public  function  __construct($nama,  $nim,
$jurusan) {
            $this->nama = $nama;
            $this->nim = $nim;
            $this->jurusan = $jurusan;
        }

        // Metode untuk menampilkan data mahasiswa
        public function tampilkanMahasiswa() {
            echo "Nama: " . $this->nama . "<br>";
            echo "NIM: " . $this->nim . "<br>";
             echo  "Jurusan:  "  .  $this->jurusan  .
"<br>";
        }
    }
```

```
    // Membuat objek Mahasiswa
    $mahasiswa1     =     new     Mahasiswa("Dzimar
Rauhillah", "20231001", "Sistem Informasi");

    // Memanggil metode untuk menampilkan data
    $mahasiswa1->tampilkanMahasiswa();
?>
```

Penjelasan kode di atas, yaitu membuat kelas Mahasiswa untuk merepresentasikan objek mahasiswa. Properti (`$nama`, `$nim`, `$jurusan`) menyimpan data mahasiswa. Constructor `__construct()` digunakan untuk menginisialisasi objek. Metode `tampilkanMahasiswa()` untuk menampilkan data mahasiswa. Objek Mahasiswa dibuat dengan data yang diberikan, lalu metode dipanggil untuk menampilkan informasi.

C. Manfaat dan Keunggulan OOP

Pemrograman Berorientasi Objek (*Object-Oriented Programming* / OOP) memiliki sejumlah keunggulan yang menjadikannya paradigma populer dalam pengembangan perangkat lunak modern. Dengan struktur berbasis objek, OOP menawarkan fleksibilitas, modularitas, dan kemudahan pengelolaan kode yang sangat membantu dalam membangun sistem yang besar dan kompleks. Berikut adalah manfaat dan keunggulan utama dari OOP:

1. Modularitas dan Reusability

OOP memungkinkan kita memecah program menjadi bagian-bagian kecil (kelas) yang independen dan dapat digunakan ulang. Artinya, jika kita sudah membuat kelas tertentu, kita bisa menggunakannya kembali dalam proyek lain tanpa menulis ulang kode.

Contoh:

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

```
    $kalkulator = new Kalkulator();
    echo $kalkulator->tambah(10, 5); // Output:
15
?>
```

Penjelasan kode di atas:

- `class Kalkulator`: Mendefinisikan blueprint atau cetakan dari kalkulator.
- `tambah($a, $b)`: Metode publik yang dapat digunakan untuk menjumlahkan dua angka.
- `new Kalkulator()`: Membuat objek baru dari kelas Kalkulator.
- `$kalkulator->tambah(10, 5)`: Memanggil metode tambah melalui objek.

Contoh kode di atas memperlihatkan bagaimana sebuah modul (dalam hal ini kelas `Kalkulator`) bisa digunakan kembali di banyak tempat dalam aplikasi, hanya dengan memanggil objek dan metodenya.

2. Kemudahan Pemeliharaan

Struktur OOP memungkinkan pengembang untuk mengelola perubahan dengan lebih mudah. Jika suatu fitur perlu diperbarui, cukup ubah kelas terkait tanpa perlu mengganggu bagian program lainnya.

Contoh :

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

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

        public function tampilkan() {
             return "Nama Mahasiswa: " . $this-
>nama;
        }
    }

    $mhs = new Mahasiswa("Dzimar");
```

```
    echo $mhs->tampilkan(); // Output: Nama
Mahasiswa: Dzimar
?>
```

Penjelasan:

- `public $nama`: Properti publik yang bisa menyimpan nama mahasiswa.
- `__construct($nama)`: Konstruktor otomatis yang dijalankan saat objek dibuat.
- `$this->nama`: Referensi ke properti objek saat ini.
- `tampilkan()`: Metode yang mencetak nama mahasiswa.
- `$mhs = new Mahasiswa("Dzimar")`: Membuat objek baru dengan nama `"Dzimar"`.
- `$mhs->tampilkan()`: Menampilkan nama tersebut.

Contoh kode di atas dengan pemisahan logika ke dalam kelas, Anda bisa mengubah cara menampilkan data hanya dengan mengubah satu metode saja (tampilkan), tanpa menyentuh bagian lain.

3. Enkapsulasi dan Keamanan Data

Enkapsulasi berarti menyembunyikan data (properti) dalam objek agar tidak bisa diakses langsung dari luar kelas. Akses ke data dilakukan melalui metode khusus (*getter*/*setter*), sehingga keamanan data lebih terjamin.

Contoh:

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

        public function setSaldo($nilai) {
            if ($nilai >= 0) {
                $this->saldo = $nilai;
            }
        }

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

Penjelasan:

- `private $saldo`: Data hanya bisa diakses dari dalam kelas, tidak bisa langsung diakses dari luar objek.
- `setSaldo()`: Fungsi untuk mengatur saldo, hanya menerima nilai ≥ 0 (validasi).
- `getSaldo()`: Fungsi untuk mengambil nilai saldo.
- `$akun->setSaldo(100000)`: Mengatur saldo dengan cara yang aman.
- `$akun->getSaldo()`: Mengambil nilai saldo.

Contoh kode di atas adalah contoh nyata enkapsulasi: data disembunyikan dari luar, dan akses diatur melalui metode yang aman (*getter* dan *setter*).

4. **Fleksibilitas melalui Pewarisan (Inheritance)**

Pewarisan memungkinkan kita membuat kelas baru yang mewarisi atribut dan metode dari kelas lain. Ini mempermudah ekspansi dan pengorganisasian kode.

Contoh:

```
<?php
class Kendaraan {
        public function jalankan() {
            return "Kendaraan berjalan";
        }
    }

    class Mobil extends Kendaraan {
        public function klakson() {
            return "Klakson berbunyi!";
        }
    }
?>
```

Penjelasan:

- `class Mobil extends Kendaraan`: Mobil adalah kelas turunan dari Kendaraan, artinya ia mewarisi semua properti dan metode dari Kendaraan.
- `jalankan()`: Metode milik Kendaraan, bisa digunakan langsung oleh Mobil.

- `klakson()`: Metode khusus yang hanya dimiliki oleh Mobil.
- `$avanza = new Mobil()`: Objek Mobil bisa memanggil metode dari Kendaraan dan Mobil.

Pewarisan memungkinkan kita membuat kelas baru dari kelas yang sudah ada, tanpa menulis ulang semua kodenya.

5. Penggunaan Polimorfisme

Polimorfisme memungkinkan objek yang berbeda merespon metode yang sama dengan cara yang berbeda. Hal ini sangat berguna saat menggunakan pewarisan dan antarmuka.

Contoh:

```
<?php
class Hewan {
        public function bersuara() {
            return "Hewan bersuara...";
        }
    }

    class Anjing extends Hewan {
        public function bersuara() {
            return "Guk Guk!";
        }
    }

    class Kucing extends Hewan {
        public function bersuara() {
            return "Meong!";
        }
    }
?>
```

Penjelasan:

- `Hewan` adalah kelas induk dengan metode `bersuara()`.
- `Anjing` dan `Kucing` adalah turunan yang meng-*override* (menimpa) metode `bersuara()` dengan versi masing-masing.
- Meski dipanggil dengan nama metode yang sama (`bersuara()`), hasilnya berbeda-beda tergantung objeknya.

Polimorfisme memberi fleksibilitas sehingga bisa memproses objek-objek berbeda dengan cara yang seragam tapi tetap menghasilkan *output* spesifik.

D. Konsep Dasar OOP

Dalam pemrograman berorientasi objek (OOP), terdapat lima konsep utama yang menjadi fondasi: *Class*, *Object*, *Encapsulation*, *Inheritance*, dan *Polymorphism*. Konsep-konsep ini mempermudah pengembangan aplikasi dengan struktur kode yang lebih rapi, terorganisir, dan mudah dikembangkan.

1. Class (Kelas)

Kelas adalah cetak biru (*blueprint*) atau template dari objek. Di dalam kelas, kita mendefinisikan atribut (variabel) dan perilaku (metode/fungsi) yang dimiliki oleh objek. Contoh kelas bernama Buah yang memiliki properti seperti nama, warna, dan berat. Kita dapat menentukan variabel seperti $nama, $warna, dan $berat untuk menampung nilai properti ini. Ketika objek individual (apel, pisang, dll.) dibuat, objek tersebut mewarisi semua properti dan perilaku dari kelas, tetapi setiap objek akan memiliki nilai yang berbeda untuk properti tersebut.

Kelas didefinisikan dengan menggunakan kata kunci `class`, diikuti dengan nama kelas dan sepasang kurung kurawal ({}). Semua properti dan metodenya berada di dalam kurung kurawal:

Sintaks

```
<?php
    class Buah {
      // code goes here...
    }
?>
```

Di bawah ini contoh mendeklarasikan *class* bernama `Buah` yang terdiri dari dua properti yaitu `$nama` dan `$warna` dan dua metode `set_name()` dan `get_name()` untuk mengatur dan mendapatkan properti `$nama`:

```
<?php
    class Buah {
      // Properties
```

```
    public $nama;
    public $warna;

    // Methods
    function set_name($nama) {
      $this->nama = $nama;
    }
    function get_name() {
      return $this->nama;
    }
  }
?>
```

2. Object (Objek)

`Object` adalah instansi nyata dari *class* atau perwujudan dari sebuah *class*. Jika *class* dapat diibaratkan sebagai "cetak biru" atau "template", maka objek adalah produk jadi yang dibuat berdasarkan cetak biru tersebut. *Object* dapat menampilkan atau mengelola isi *class*. Seluruh isi *class* akan instansiasikan menjadi *object* setelah *class* didefinisikan, sehingga objek dapat mengakses dan menggunakan data serta fungsi di dalam kelas. Setiap *object* memiliki semua properti dan metode yang didefinisikan di dalam *class*, tetapi *object* akan memiliki nilai properti yang berbeda.

Object dari sebuah kelas dibuat menggunakan kata kunci `new`. Dalam contoh di bawah ini, $apel dan $pisang adalah *instance* dari *class* Buah:

```
<?php
class Buah {
  // Properties
  public $nama;
  public $warna;

  // Methods
  function set_name($nama) {
    $this->nama = $nama;
  }
  function get_name() {
    return $this->nama;
  }
```

```
}

$apel = new Buah();
$pisang = new Buah();
$apel->set_name('Apel');
$pisang->set_name('Pisang');

echo $apel->get_name();
echo "<br>";
echo $pisang->get_name();
?>
```

Penjelasan kode:

a. Mendefinisikan `class Buah`

Mendefinisikan `class Buah` yang memiliki dua elemen utama:

1) *Properties*

`public $nama` dan `public $warna` adalah variabel yang akan menyimpan data untuk setiap buah. Dalam contoh ini, yang digunakan hanya `$nama`.

2) *Methods*

`set_name()` dan `get_name()` adalah fungsi yang mendefinisikan perilaku dari `object`. Untuk `set_name($nama)` yaitu *methods* yang menerima sebuah nilai `$nama` dan menyimpannya ke dalam properti `$nama` milik `object` saat ini (`$this->nama`). Sedangkan `get_name()` merupakan *methods* yang mengembalikan nilai yang disimpan di dalam properti `$nama`.

b. Membuat *Object (Instansiasi)*

Setelah `class Buah` dibuat, selanjutnya membuat dua object nyata dari class yang menggunakan kata kunci new tersebut yaitu `$apel = new Buah()` dan `$pisang = new Buah()`. Keduanya merupakan entitas yang terpisah meskipun keduanya berasal dari `class` yang sama yaitu `class Buah`.

c. Menggunakan *Object*

Selanjutnya menggunakan metode yang ada di dalam setiap

objek untuk memberikan nilai, yaitu `$apel->set_name('Apel')` untuk memanggil metode `set_name` pada objek `$apel`. Untuk mengatur properti `$nama` di dalam objek `$apel` menjadi `'Apel'`. `$pisang->set_name('Pisang')` untuk memanggil metode yang sama, tetapi pada objek `$pisang`. Ini mengatur properti `$nama` di dalam objek `$pisang` menjadi `'Pisang'`.

d. Menampilkan Hasil
Terakhir menampilkan nilai dari setiap objek yaitu `echo $apel->get_name()` Kode ini menghasilkan output `'Apel'`. Untuk `echo $pisang->get_name()` Kode ini menghasilkan output `'Pisang'` di baris yang berbeda karena menggunakan
.

3. Enkapsulasi (Encapsulation)

Enkapsulasi adalah praktik menggabungkan data (properti) dan metode (fungsi) yang beroperasi pada data tersebut ke dalam satu unit yang disebut kelas. Konsep ini sering diibaratkan seperti sebuah kapsul, di mana data dan logika terkait dikemas menjadi satu. Tujuan utama dari enkapsulasi adalah untuk menyembunyikan detail implementasi dari luar. Dengan kata lain, data internal sebuah objek tidak bisa diakses atau diubah secara langsung dari luar kelas. Akses terhadap data hanya bisa dilakukan melalui metode publik yang telah disediakan oleh kelas. Hal ini memastikan bahwa data tetap valid dan konsisten, karena setiap perubahan harus melalui "gerbang" yang telah ditentukan, yaitu metode-metode tersebut. Contoh penggunaan enkapsulasi pada kode di bawah ini:

```
<?php

class Buah {

    public $nama;
    public $warna;

    public function set_name(string $nama): void
{
```

```
        $this->nama = $nama;
    }

    public function get_name(): string {
        return $this->nama;
    }

    public function set_warna(string $warna):
void {
        $this->warna = $warna;
    }

    public function get_warna(): string {
        return $this->warna;
    }
}

// --- Instansiasi dan Penggunaan Objek ---

$apel = new Buah();

$apel->set_name('Apel');
$apel->set_warna('Merah');

$pisang = new Buah();

$pisang->set_name('Pisang');
$pisang->set_warna('Kuning');

echo "Buah pertama adalah: " . $apel->get_name()
. " dengan warna " . $apel->get_warna() . ".";
echo "<br>";
echo "Buah kedua adalah: " . $pisang->get_name()
. " dengan warna " . $pisang->get_warna() . ".";
?>
```

Penjelasan kode di atas, enkapsulasi diwujudkan sebagai berikut:

a. Penyatuan Data dan Logika

Kelas Buah mengemas properti `$nama` dan `$warna` (data) bersama dengan metode `set_name()`, `get_name()`,

`set_warna()`, dan `get_warna()` (logika) ke dalam satu unit yang sama yang merupakan sebagai inti dari enkapsulasi.

b. Pengendalian Akses (Melalui *Setter* dan *Getter*)
Meskipun properti `$nama` dan `$warna` dideklarasikan sebagai `public` (dapat diakses langsung), praktik yang lebih baik adalah menggunakan metode set (*setter*) dan get (*getter*) untuk mengelola data tersebut. Pada kode `$apel->set_name('Apel')` tidak mengubah `$apel->nama` secara langsung. Sebaliknya, menggunakan metode `set_name()` untuk memberikan nilai. Jika di masa depan kita ingin menambahkan validasi (misalnya, memastikan nama tidak kosong), kita bisa menambahkannya di dalam metode `set_name()` tanpa mengubah kode yang sudah menggunakan kelas ini. Pada kode `echo $apel->get_name()` mengambil nilai properti melalui metode `get_name()`. Metode ini bisa saja diubah untuk memformat data sebelum dikembalikan, dan kode di luar kelas tidak perlu tahu perubahan itu.

Dengan demikian, meskipun contoh ini sederhana, ia sudah menunjukkan prinsip dasar enkapsulasi: data dan fungsionalitas yang mengoperasikannya dikemas dalam satu kesatuan, dan interaksi dengan data tersebut diatur melalui metode-metode yang disediakan oleh kelas.

4. Pewarisan (Inheritance)

Pewarisan adalah salah satu konsep fundamental dalam pemrograman berorientasi objek (OOP) yang memungkinkan sebuah kelas mewarisi properti dan metode dari kelas lain. Konsep ini mempromosikan penggunaan kembali kode (*code reusability*) dan membantu menciptakan hierarki kelas yang logis. Kelas Induk (*Parent Class*) yaitu kelas yang memberikan properti dan metodenya untuk diwarisi. Kelas Anak (*Child Class*) yaitu kelas yang mewarisi properti dan metode dari kelas induk. Kelas anak juga bisa memiliki properti dan metode uniknya sendiri. Secara analogi pewarisan seperti hubungan keluarga. Seorang anak mewarisi beberapa ciri fisik (properti) dan bakat

(metode) dari orang tuanya, namun juga memiliki karakteristik unik yang membedakannya. Contoh penggunaan pewarisan pada kode di bawah ini:

```
<?php

// Kelas Induk (Parent Class)
class Buah {
    // Properti yang akan diwarisi
    public $nama;
    public $warna;

    // Konstruktor untuk inisialisasi properti
    public function __construct($nama, $warna)
{
        $this->nama = $nama;
        $this->warna = $warna;
    }

    // Metode yang akan diwarisi
    public function intro() {
        echo "Nama buah: {$this->nama},
warnanya: {$this->warna}.";
    }
}

// --- Kelas Anak (Child Class) ---

class Apel extends Buah {
    public $jenis;

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

    public function pesan() {
        echo "Ini adalah buah apel jenis
{$this->jenis}. ";
    }
}
```

```
// Kelas Pisang mewarisi dari kelas Buah
class Pisang extends Buah {
    // Metode unik untuk kelas Pisang
    public function intro() {
        echo "Pisang ini manis! ";
        parent::intro();
    }
}

// --- Penggunaan Kelas ---

// Membuat objek dari kelas Apel (kelas anak)
$apel_merah = new Apel("Apel", "Merah", "Granny
Smith");
$apel_merah->pesan();
$apel_merah->intro();
echo "<br>";

// Membuat objek dari kelas Pisang (kelas anak)
$pisang_kuning = new Pisang("Pisang",
"Kuning");
$pisang_kuning->intro();
?>
```

Penjelasan kode di atas:

a. Kelas Induk `Buah` yaitu mendefinisikan kelas dasar yang memiliki properti `$nama` dan `$warna`, serta metode `intro()`. Kedua properti dan metode ini akan diwarisi oleh semua kelas anak.
b. `class Apel extends Buah` yaitu kata kunci `extends` menunjukkan bahwa `Apel` adalah kelas anak dari `Buah`.
    1) Pewarisan Properti dan Metode yaitu Objek `$apel_merah` dapat mengakses properti `$nama` dan `$warna` serta metode `intro()` tanpa harus mendefinisikannya kembali di dalam kelas `Apel`.
    2) Properti Unik yaitu Kelas `Apel` menambahkan properti `$jenis` yang hanya dimiliki oleh apel.

3) Konstruktor `__construct()` yaitu kelas anak memanggil `parent::__construct()` untuk memastikan properti dari kelas induk (`$nama` dan `$warna`) juga diinisialisasi dengan benar.

c. `class Pisang extends Buah` menimpa Metode (*Method Overriding*) yaitu Kelas `Pisang` memiliki metode `intro()` sendiri. Ketika kita memanggil `$pisang_kuning->intro()`, PHP akan menjalankan metode yang ada di kelas `Pisang`, bukan yang diwarisi dari `Buah`. Ini menunjukkan bagaimana kelas anak bisa memodifikasi perilaku yang diwarisi.

Output dari kode di atas:

```
Ini adalah buah apel jenis Granny Smith.
Nama buah: Apel, warnanya: Merah.
Pisang ini manis! Nama buah: Pisang,
warnanya: Kuning.
```

Ini menunjukkan bagaimana objek `$apel_merah` dapat menggunakan metode unik (`pesan()`) sekaligus metode yang diwarisi (`intro()`), sementara objek `$pisang_kuning` dapat menimpa perilaku yang diwarisi dengan metode `intro()` -nya sendiri.

5. Polimorfisme (Polymorphism)

Polimorfisme adalah pilar OOP yang memungkinkan objek dari kelas yang berbeda untuk diperlakukan sebagai objek dari kelas induk yang sama. Secara harfiah, "polimorfisme" berarti "banyak bentuk" (*poly* = banyak, *morph* = bentuk). Dalam praktiknya, ini berarti Anda dapat memanggil metode yang sama pada objek yang berbeda, dan setiap objek akan merespons dengan cara yang unik sesuai dengan implementasinya masing-masing. Polimorfisme sering kali diwujudkan melalui pewarisan dan antarmuka (*interface*). Contoh paling umum dari polimorfisme adalah ketika sebuah metode di kelas induk ditimpa (*overridden*) oleh kelas-kelas anaknya. Ketika Anda memanggil metode tersebut, sistem akan secara otomatis menentukan implementasi mana yang harus dijalankan, tergantung pada jenis objek yang sedang digunakan. Dalam contoh di bawah ini, bagaimana bisa

memanggil metode yang sama (`deskripsi()`) pada objek yang berbeda, dan setiap objek akan memberikan *output* yang sesuai dengan implementasi uniknya.

```
<?php

// Kelas Induk (Parent Class)
abstract class Buah {
    // Properti
    public $nama;

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

    // Ini adalah kunci polimorfisme
    abstract public function deskripsi();
}

// --- Kelas Anak (Child Class) ---

// Kelas Apel mewarisi dari Buah
class Apel extends Buah {
    public $jenis;

    // Konstruktor kelas anak
    public function __construct(string $nama,
string $jenis) {
        parent::__construct($nama);
        $this->jenis = $jenis;
    }

    // Implementasi unik dari metode deskripsi
    public function deskripsi() {
          return "Ini adalah buah apel jenis
{$this->jenis}, rasanya manis dan renyah.";
    }
}

// Kelas Pisang mewarisi dari Buah
class Pisang extends Buah {
```

```
    // Implementasi unik dari metode deskripsi
    public function deskripsi() {
        return "Ini adalah buah pisang, memiliki
kulit kuning dan tekstur yang lembut.";
    }
}

// --- Penggunaan Polimorfisme ---

$buah_buahan = [
    new Apel("Apel", "Granny Smith"),
      new   Pisang("Pisang",   ""),   //   Nama
diinisialisasi di konstruktor induk
];

foreach ($buah_buahan as $buah) {
    echo $buah->deskripsi() . "<br>";
}
?>
```

Penjelasan kode di atas:

a. Kelas Induk `Buah` (Kelas Abstrak) sekarang dideklarasikan sebagai `abstract`. Ini berarti kelas ini tidak bisa dibuat objeknya secara langsung, dan tujuannya hanya sebagai "cetak biru" untuk kelas anaknya. Metode `deskripsi()` juga dideklarasikan sebagai `abstract`. Ini memaksa semua kelas anak untuk menyediakan implementasi unik dari metode ini. Inilah yang menjadi dasar dari polimorfisme.
b. Kelas Anak `Apel` dan `Pisang` Kedua kelas ini mengimplementasikan metode `deskripsi()` dengan cara yang berbeda. `Apel::deskripsi()` memberikan deskripsi khusus untuk apel. `Pisang::deskripsi()` memberikan deskripsi khusus untuk pisang.
c. Penggunaan Polimorfisme: Kita membuat sebuah array `$buah_buahan` yang berisi campuran objek `Apel` dan `Pisang`. Kita menggunakan perulangan foreach untuk mengiterasi array tersebut. Di dalam perulangan, kita memanggil `$buah->deskripsi()`. Perhatikan bahwa kita memanggil metode yang sama pada setiap objek `$buah`.

Hasilnya, PHP secara otomatis mengetahui implementasi `deskripsi()` mana yang harus dipanggil berdasarkan jenis objeknya (apakah itu objek `Apel` atau `Pisang`).

Output dari kode di atas:

```
Ini adalah buah apel jenis Granny Smith,
rasanya manis dan renyah.
Ini adalah buah pisang, memiliki kulit
kuning dan tekstur yang lembut.
```

Polimorfisme memungkinkan kita menulis kode yang rapi dan fleksibel. Kita bisa menambahkan kelas buah baru, misalnya `Mangga`, tanpa perlu mengubah kode perulangan yang ada. Selama kelas `Mangga` mengimplementasikan metode `deskripsi()`, kode akan tetap berfungsi dengan baik.

RANGKUMAN

Pada bab ini telah dibahas konsep dasar Pemrograman Berorientasi Objek (OOP) yang menjadi salah satu paradigma utama dalam pengembangan perangkat lunak modern. Beberapa poin penting yang dapat diambil antara lain:

1. OOP merupakan paradigma pemrograman yang berfokus pada objek, di mana setiap objek memiliki atribut (data/properti) dan perilaku (metode/fungsi).
2. Dibandingkan dengan pemrograman prosedural, OOP lebih terstruktur, modular, dan mudah dipelihara karena menyatukan data dan fungsi ke dalam satu kesatuan.
3. Manfaat utama OOP antara lain: modularitas, *reusability*, kemudahan pemeliharaan, keamanan data melalui enkapsulasi, fleksibilitas dengan pewarisan, dan efisiensi melalui polimorfisme.

4. Lima konsep dasar OOP yang harus dipahami yaitu:
   - *Class* → cetak biru (*blueprint*) objek.
   - *Object* → instansi nyata dari sebuah kelas.
   - *Encapsulation* → menyembunyikan data dan membatasi

akses melalui *getter*/*setter*.

- *Inheritance* → pewarisan properti dan metode dari kelas induk ke kelas anak.
- *Polymorphism* → satu metode dapat memiliki perilaku berbeda tergantung objek yang memanggilnya.

LATIHAN

1. Latihan Teori

1. Jelaskan perbedaan utama antara pemrograman prosedural dan OOP.
2. Sebutkan manfaat penggunaan enkapsulasi dalam pengembangan perangkat lunak.
3. Apa yang dimaksud dengan pewarisan (*inheritance*) dalam OOP? Berikan contohnya.
4. Bagaimana polimorfisme dapat membantu membuat kode lebih fleksibel?
5. Mengapa OOP dianggap lebih modular dibandingkan pemrograman prosedural?

2. Latihan Praktik

1. Buat kelas Mahasiswa dengan properti nama, nim, dan jurusan. Tambahkan *constructor* dan metode untuk menampilkan data mahasiswa.
2. Buat kelas Kalkulator dengan metode tambah(), kurang(), kali(), dan bagi(). Demonstrasikan penggunaannya melalui objek.
3. Implementasikan kelas AkunBank yang memiliki properti private saldo. Buat metode untuk setSaldo(), getSaldo(), deposit(), dan tarik() dengan validasi agar saldo tidak negatif.
4. Buat kelas induk Hewan dengan metode bersuara(). Turunkan menjadi kelas Kucing dan Anjing dengan implementasi suara masing-masing. Tampilkan contoh polimorfisme.

BAB 2
Pengenalan Hypertext Preprocessor (PHP)

DESKRIPSI:

Bab ini membahas pengenalan dasar mengenai PHP (Hypertext Preprocessor), yaitu bahasa pemrograman *server-side* yang banyak digunakan dalam pengembangan aplikasi web dinamis. PHP dirancang khusus untuk membuat halaman web interaktif yang dapat berkomunikasi dengan *database* dan mendukung berbagai protokol internet.

TUJUAN PEMBELAJARAN:

Setelah mempelajari bab ini, diharapkan mampu:

1. Menjelaskan secara singkat sejarah dan evolusi bahasa pemrograman PHP.
2. Melakukan instalasi dan konfigurasi lingkungan pengembangan PHP di komputer.
3. Memahami dan menulis struktur dasar program PHP, termasuk sintaks, variabel, dan tipe data.
4. Penggunaan operator, percabangan, perulangan, fungsi, dan *array* untuk menyusun program dasar.
5. Penerapan PHP dalam mengolah data *form* serta integrasi sederhana dengan HTML.

A. Sejarah dan Perkembangan PHP

PHP (*Hypertext Preprocessor*) adalah bahasa skrip *open-source* yang dirancang khusus untuk pengembangan web. Perkembangannya dimulai pada tahun 1994 oleh Rasmus Lerdorf yang awalnya membuat serangkaian skrip Perl untuk melacak

pengunjung resume *online*-nya. Ia menamai skrip ini "*Personal Home Page Tools*". Pada tahun 1995, Rasmus Lerdorf merilis kode sumbernya ke publik. Ini memicu kolaborasi para pengembang di seluruh dunia dan melahirkan PHP/FI (*Personal Home Page*/*Forms Interpreter*) yang merupakan kombinasi dari sintaks Perl dan C. PHP/FI sudah memiliki beberapa fitur dasar yang mirip dengan PHP modern, seperti konektivitas basis data dan interpretasi formulir.

Pada tahun 1997, Andi Gutmans dan Zeev Suraski menulis ulang parser PHP/FI secara total. Versi baru ini dinamai PHP 3, mengubah akronim PHP menjadi "*Hypertext Preprocessor*" dan menjadi tonggak penting. PHP 3 adalah versi pertama yang benar-benar mirip dengan PHP yang kita kenal sekarang dengan sintaks lebih konsisten dan dukungan *database* lebih luas. Pada tahun 1999, Gutmans dan Suraski mendirikan *Zend Technologies* dan merilis PHP 4 dengan mesin eksekusi baru yang disebut *Zend Engine*. PHP 4 meningkatkan performa secara signifikan, memperkenalkan konsep sesi, *buffering output*, dan dukungan untuk objek lebih baik. Versi ini menjadi sangat populer dan mendominasi pengembangan web dinamis. Dirilis pada tahun 2004, PHP 5 membawa perubahan besar dengan memperkenalkan *Zend Engine* 2.0. Peningkatan utama pada versi ini adalah dukungan OOP yang lebih komprehensif dan modern. Fitur-fitur seperti *visibility* (*public*, *private*, *protected*), antarmuka (*interface*), kelas abstrak (*abstract class*), dan penanganan pengecualian (*exception handling*) membuat PHP menjadi pilihan yang kuat untuk pengembangan aplikasi skala besar.

Setelah PHP 6 tidak jadi dirilis, PHP 7 muncul pada tahun 2015 dan dianggap sebagai lompatan terbesar dalam sejarah PHP. *Zend Engine* 3.0 yang baru meningkatkan performa secara drastis, membuatnya dua kali lebih cepat dari PHP 5. Fitur baru seperti *type hinting* untuk tipe data skalar dan operator *null coalescing* juga ditambahkan, membuat kode lebih bersih dan aman. Versi terbaru, PHP 8, dirilis pada tahun 2020. Ini membawa banyak fitur modern seperti JIT (*Just-In-Time*) *Compiler* untuk performa yang lebih baik pada beban kerja tertentu, *named arguments*, dan *match expression*. Perkembangan PHP terus berlanjut hingga saat ini, menjadikannya salah satu bahasa pemrograman paling relevan dan kuat di dunia web.

B. Instalasi dan Konfigurasi Lingkungan PHP

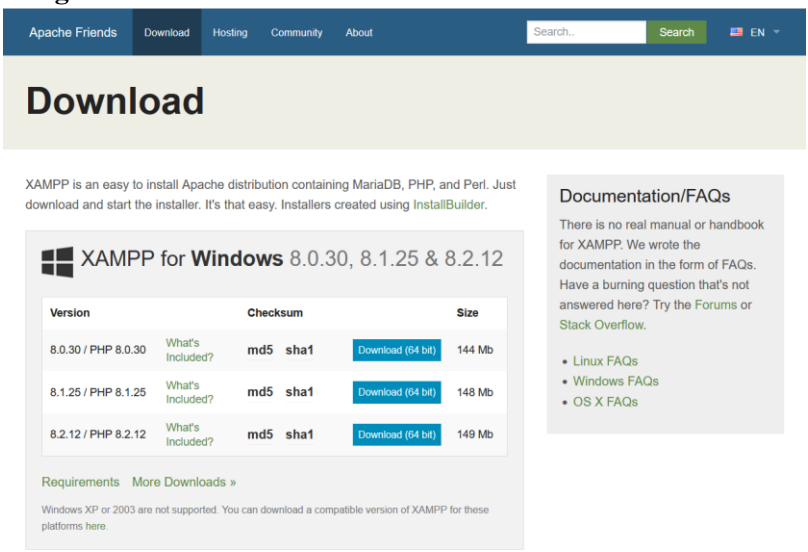
Untuk memulai pengembangan web dengan PHP, perlu menyiapkan lingkungan pengembangan yang terdiri dari beberapa komponen. PHP adalah bahasa *server-side*, jadi membutuhkan server web untuk menjalankan kode PHP. Cara termudah untuk melakukannya adalah dengan menginstal paket server lokal yang sudah mencakup semua yang dibutuhkan.

Menginstal komponen secara terpisah (Apache, PHP, MySQL) bisa rumit. Oleh karena itu, kebanyakan pengembang menggunakan paket instalasi tunggal seperti XAMPP, WAMP, atau MAMP. Berikut untuk penjelasannya:

1. XAMPP adalah singkatan dari X (*cross-platform*), Apache, MariaDB, PHP, dan Perl. pilihan yang paling populer dan serbaguna. Cocok untuk pengguna Windows, macOS, dan Linux
2. WAMP adalah singkatan dari Windows, Apache, MySQL, dan PHP. Dirancang khusus untuk pengguna Windows
3. MAMP adalah singkatan dari MacOS, Apache, MySQL, dan PHP, Dirancang khusus untuk pengguna MacOS

Berikut adalah langkah-langkah umum untuk menginstal dan mengonfigurasi XAMPP pada sistem operasi *window*, yang akan digunakan sebagai contoh:

Langkah 1: Unduh XAMPP



Gambar 1. Halaman Web Download XAMPP

Kunjungi situs web resmi Apache Friends https://www.apachefriends.org/download.html dan unduh versi XAMPP yang sesuai dengan sistem operasi Anda (Windows, Linux, atau macOS. Pada gambar 1 halaman web untuk download XAMPP menampilkan link download dan versi PHP-nya bisa disesuaikan dengan kebutuhan.

Langkah 2: Instal XAMPP

Setelah proses unduhan selesai, jalankan file installer XAMPP yang telah diunduh (biasanya berekstensi .exe). Sistem operasi Windows mungkin akan menampilkan peringatan keamanan (*User Account Control*). Klik Yes untuk melanjutkan proses instalasi.

Pada jendela awal instalasi, klik tombol Next untuk memulai. Installer akan menampilkan daftar komponen yang dapat diinstal, seperti Apache, MySQL/MariaDB, PHP, phpMyAdmin, dan komponen tambahan lainnya. Untuk kebutuhan pengembangan web dengan PHP, disarankan untuk memilih minimal Apache, MySQL (atau MariaDB), PHP, dan phpMyAdmin, kemudian klik Next.

Langkah 3: Menentukan Lokasi Instalasi

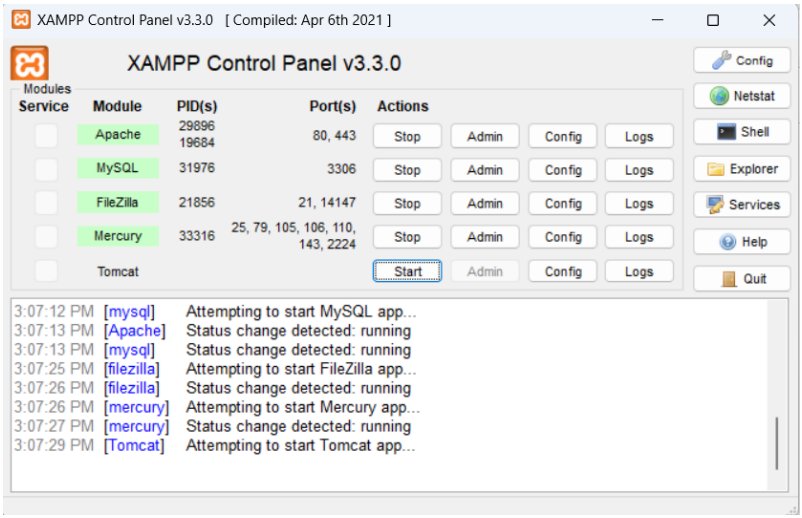
Pada tahap ini, pengguna diminta untuk menentukan direktori instalasi XAMPP. Secara default, XAMPP akan diinstal pada direktori c:\xampp. Direktori default ini disarankan karena memudahkan konfigurasi dan kompatibel dengan sebagian besar tutorial PHP. Setelah menentukan lokasi instalasi, klik Next untuk melanjutkan.

Langkah 4: Proses Instalasi

Installer akan mulai menyalin file dan mengkonfigurasi komponen yang diperlukan. Proses ini membutuhkan waktu beberapa menit tergantung pada spesifikasi komputer. Selama proses berlangsung, pengguna cukup menunggu hingga instalasi selesai.

Setelah proses instalasi selesai, akan muncul notifikasi bahwa instalasi berhasil. Klik Finish untuk mengakhiri proses instalasi. Pengguna juga dapat memilih untuk langsung menjalankan XAMPP Control Panel.

Langkah 5: Menjalankan XAMPP Control Panel

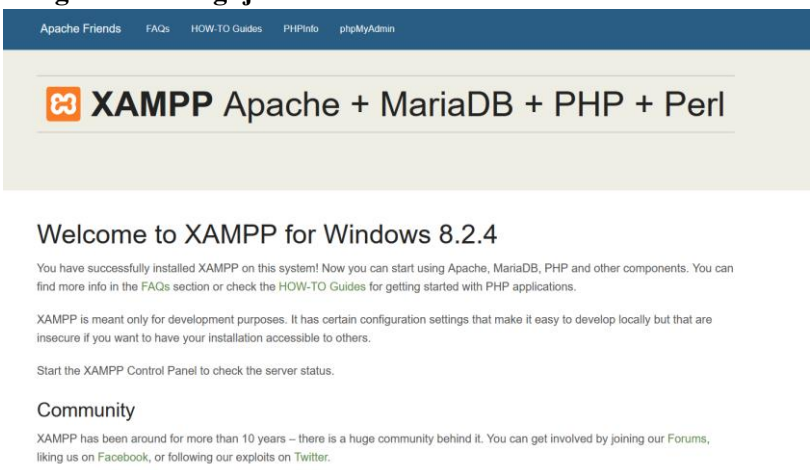


Gambar 2. Halaman XAMPP Control Panel

Setelah instalasi selesai, buka XAMPP Control Panel. Pada panel ini akan terlihat beberapa modul utama seperti Apache, MySQL, FileZilla, dan Tomcat.

Klik tombol Start pada modul Apache dan MySQL. Jika kedua modul berhasil dijalankan, indikator status akan berubah menjadi warna hijau, menandakan bahwa server web dan database telah aktif.

Langkah 6: Menguji Instalasi XAMPP



Gambar 3. Halaman XAMPP Dashboard

Untuk memastikan XAMPP telah terinstal dan berjalan dengan baik, buka browser (Google Chrome, Firefox, atau Edge), lalu ketikkan alamat berikut pada address bar: http://localhost Jika instalasi berhasil, maka akan muncul halaman XAMPP Dashboard seperti pada gambar di atas. Hal ini menandakan bahwa server Apache dan PHP telah berjalan dengan normal.

C. Struktur Dasar Program PHP

1. TAG PHP

Tag PHP adalah penanda khusus yang memberitahu web server bahwa kode di dalamnya harus diinterpretasikan sebagai PHP, bukan sebagai teks biasa atau HTML. Kode di dalam tag inilah yang akan dieksekusi di sisi server, dan hasilnya (jika ada) akan digabungkan dengan HTML dan dikirimkan ke browser. Kode PHP harus selalu diletakkan di antara tag pembuka dan tag penutup. Ada beberapa jenis tag, tetapi yang paling umum digunakan adalah:

```
<?php
 // Kode PHP ditulis di sini
?>
```

Penjelasan kode:

- `<?php` adalah tag pembuka yang memberitahu server bahwa kode selanjutnya adalah kode PHP.
- `?>` adalah tag penutup yang menandakan akhir dari blok kode PHP.

Jika file PHP hanya berisi kode PHP dan tidak ada HTML yang mengikutinya, tag penutup `?>` sering kali dihilangkan untuk menghindari masalah spasi atau *header* yang tidak diinginkan. Salah satu kekuatan terbesar PHP adalah kemampuannya untuk disematkan di dalam file HTML sehingga dapat membuka dan menutup tag PHP sebanyak yang dibutuhkan dalam satu file untuk mencampur kode PHP dengan *markup* HTML. Contoh di bawah ini:

```
<!DOCTYPE html>
<html>
<head>
    <title>Situs Saya</title>
```

```
</head>
<body>

    <h1>
        <?php
            $judul = "Selamat Datang di PHP!";
            echo $judul;
        ?>
    </h1>

    <p>
        <?php
            $tahun = date("Y");
            echo "Hak Cipta &copy; " . $tahun;
        ?>
    </p>

</body>
</html>
```

Dalam contoh di atas, menggunakan dua blok PHP yang berbeda. Blok pertama mencetak judul, dan blok kedua mencetak tahun saat ini secara dinamis.

2. Komentar Dalam PHP

Komentar adalah bagian penting dari kode yang tidak akan dieksekusi oleh mesin PHP. Fungsinya adalah untuk memberikan penjelasan atau catatan bagi *programmer*, baik untuk diri sendiri di masa mendatang maupun untuk *programmer* lain yang mungkin membaca kode tersebut. Komentar membantu meningkatkan keterbacaan dan pemahaman kode. PHP mendukung beberapa jenis komentar yang dapat digunakan sesuai kebutuhan.

a. Komentar Satu Baris

Jenis komentar ini digunakan untuk membuat catatan singkat pada satu baris kode, bisa menggunakannya di awal baris atau di akhir baris setelah pernyataan. Komentar satu baris dimulai dengan //. Teks apa pun antara // dan akhir baris akan diabaikan (tidak akan dieksekusi). Selain itu dapat menggunakan # untuk komentar satu baris. Contoh

```
<?php
```

```
// Ini adalah komentar satu baris
echo "Hello World!"; // Komentar juga bisa
diletakkan di akhir baris

# Ini juga komentar satu baris
$nama = "Mizar";
echo "Halo, " . $nama;
?>
```

b. Komentar Multi-Baris

Jenis komentar ini ideal untuk menulis penjelasan yang lebih panjang yang membutuhkan lebih dari satu baris. Dimulai dengan /* dan diakhiri dengan */. Contoh penggunaannya

```
<?php
/*
  Ini adalah blok komentar
  yang dapat mencakup beberapa baris.
  Sering digunakan untuk menjelaskan fungsi,
  logika yang kompleks, atau informasi hak
cipta.
*/
$umur = 25;
echo "Umur saya adalah " . $umur . " tahun.";
?>
```

Komentar yang efektif berfungsi sebagai panduan yang berharga dalam kode, tidak sekadar mengulang apa yang sudah jelas. Praktik terbaik menyarankan kita untuk menjelaskan alasan di balik suatu kode, bukan hanya apa yang dilakukannya. Komentar juga sangat berguna untuk mendokumentasikan algoritma yang kompleks atau trik tertentu, membantu developer lain (dan diri sendiri di masa depan) memahami logika yang ada. Selain itu, komentar adalah alat yang ampuh untuk *debugging*, di mana kita dapat menonaktifkan kode sementara tanpa menghapusnya. Untuk fungsi dan kelas, penggunaan format standar seperti DocBlock sangat dianjurkan karena membantu *tools* otomatis menghasilkan dokumentasi yang rapi, membuat kode lebih mudah dipelihara dan dipahami secara menyeluruh.

3. Variabel

Variabel adalah elemen fundamental dalam setiap bahasa

pemrograman, termasuk PHP. Variabel berfungsi sebagai "wadah" untuk menyimpan informasi atau nilai, seperti teks, angka, atau data lainnya. Dalam PHP, ada beberapa aturan dan karakteristik penting terkait variabel. Semua variabel di PHP harus diawali dengan tanda dolar ($). Setelah tanda dolar, nama variabel bisa berupa huruf atau garis bawah (_), diikuti oleh kombinasi huruf, angka, atau garis bawah. Contoh sintaks:

```
<?php
$nama_siswa = "Dzimar";
$umur = 9;
$total_harga = 15000.50;
?>
```

Untuk memastikan kode tetap bersih dan mudah dibaca, ada beberapa aturan dan konvensi yang perlu diikuti:

a. Harus diawali dengan $.
b. Karakter pertama setelah $ tidak boleh berupa angka. Contoh: $1nama tidak valid.
c. Hanya boleh berisi karakter alfanumerik (a-z, A-Z, 0-9) dan garis bawah (_).
d. Nama variabel peka terhadap huruf besar dan kecil (*case-sensitive*). Artinya, $nama berbeda dengan $Nama.
e. Disarankan untuk menggunakan konvensi camelCase atau snake_case. $namaSiswa (camelCase), $nama_siswa (snake_case)

PHP adalah bahasa yang *loosely typed*, artinya tidak perlu mendeklarasikan tipe data variabel secara eksplisit saat membuatnya. Tipe data akan ditentukan secara otomatis oleh PHP berdasarkan nilai yang diberikan. Contoh

```
<?php
$teks = "Ini adalah string";   // Tipe: String
$angka = 123;                  // Tipe: Integer
$desimal = 10.5;               // Tipe: Float
$benar = true;                 // Tipe: Boolean
$array_buah = ["Apel", "Jeruk"]; // Tipe: Array
$objek = new stdClass();      // Tipe: Object
?>
```

Ruang lingkup variabel menentukan di mana variabel tersebut dapat diakses. Ada tiga jenis ruang lingkup di PHP:

a. *Local,* yaitu variabel yang dideklarasikan di dalam sebuah fungsi hanya dapat diakses di dalam fungsi tersebut. Contoh

```
<?php
function myTest() {
    $x = 5; // Variabel lokal
    echo "Nilai x adalah: $x";
}
myTest();
// echo $x; // Ini akan menghasilkan error
karena $x tidak bisa diakses di luar fungsi
?>
```

b. *Global,* yaitu variabel yang dideklarasikan di luar fungsi mana pun dapat diakses dari mana saja, kecuali di dalam fungsi. Untuk mengaksesnya di dalam fungsi, Anda harus menggunakan kata kunci `global`. Contoh

```
<?php
$y = 10; // Variabel global
function myTest2() {
    global $y;
    echo "Nilai y adalah: $y";
}
myTest2();
?>
```

c. *Static,* yaitu ketika sebuah fungsi selesai dieksekusi, semua variabel lokalnya akan dihapus. Namun, jika Anda ingin variabel lokal tidak hilang dan menyimpan nilainya untuk panggilan fungsi berikutnya, Anda bisa menggunakan kata kunci `static`.

```
<?php
function myCounter() {
    static $counter = 0;
    $counter++;
    echo $counter . "<br>";
}
myCounter(); // Output: 1
myCounter(); // Output: 2
myCounter(); // Output: 3
?>
```

Memahami cara kerja variabel dan ruang lingkupnya sangat penting untuk menulis kode PHP yang efektif dan bebas dari kesalahan.

4. **Eho/Print**

Di PHP, `echo` dan `print` adalah dua cara utama untuk menampilkan *output* ke layar. Meskipun keduanya memiliki fungsi yang hampir sama, ada beberapa perbedaan kecil dalam penggunaan dan performa. `echo` adalah konstruksi bahasa yang digunakan untuk menampilkan satu atau lebih string. `echo` tidak mengembalikan nilai dan dapat menerima banyak argumen dengan menggunakan koma (,) untuk memisahkan *string* yang ingin ditampilkan. Contoh

```
<?php
// Menggunakan echo untuk satu string
echo "Halo, selamat datang di PHP!";

echo "<br>";

// Menggunakan echo dengan beberapa argumen
$nama = "Ghifar";
echo "Halo, ", $nama, "! Apa kabar?";
?>
```

Output:

```
Halo, selamat datang di PHP!
Halo, Ghifar! Apa kabar?
```

5. **Tipe Data**

Tipe data (*data type*) adalah jenis nilai yang dapat disimpan dalam variabel. PHP secara otomatis mendeteksi tipe data berdasarkan nilai yang diberikan (*Weakly Typed Language*), tetapi penting bagi *programmer* untuk memahami tipe data yang digunakan agar bisa memproses data dengan benar. PHP secara otomatis akan mengonversi tipe data sesuai konteks pemakaian (*type juggling*). Contoh penggunaan tipe data pada PHP

```
<?php
// 1. STRING
$nama = "Ahmad Fauzi"; // Tipe data: String

// 2. INTEGER
```

```
$umur = 21; // Tipe data: Integer

// 3. FLOAT
$ipk = 3.75; // Tipe data: Float

// 4. BOOLEAN
$isLulus = true; // Tipe data: Boolean

// 5. ARRAY
$mataKuliah = ["Algoritma", "Basis Data",
"Pemrograman Web"]; // Tipe data: Array

// 6. OBJECT (Menggunakan class)
class Mahasiswa {
    public $nama;
    public $umur;

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

    public function perkenalan() {
        return "Halo, nama saya $this->nama dan
saya berumur $this->umur tahun.";
    }
}

$mhs = new Mahasiswa($nama, $umur); // Tipe
data: Object

// 7. NULL
$nilaiSkripsi = NULL; // Tipe data: NULL

// 8. RESOURCE
$file = fopen("log.txt", "w"); // Tipe data:
Resource
fwrite($file, "Log data mahasiswa disimpan.");
fclose($file);

// OUTPUT SEMUA DATA
```

```
echo "<h3>Data Mahasiswa</h3>";
echo "Nama: " . $nama . "<br>";
echo "Umur: " . $umur . "<br>";
echo "IPK: " . $ipk . "<br>";
echo "Lulus: " . ($isLulus ? "Ya" : "Tidak") .
"<br>";

echo "Mata Kuliah: <ul>";
foreach ($mataKuliah as $mk) {
    echo "<li>$mk</li>";
}
echo "</ul>";

echo "Perkenalan: " . $mhs->perkenalan() .
"<br>";
echo "Nilai Skripsi: ";
var_dump($nilaiSkripsi); // Menampilkan NULL
?>
```

Dalam pengembangan aplikasi menggunakan PHP, pemahaman tipe data merupakan fondasi utama agar proses pengolahan data berjalan dengan benar dan efisien. PHP menyediakan berbagai jenis tipe data yang masing-masing memiliki fungsi spesifik:

a. *String* menyimpan informasi berupa teks seperti nama, alamat, atau deskripsi.
b. Integer digunakan untuk bilangan bulat, cocok untuk menyimpan umur, jumlah, atau skor.
c. *Float* (atau *double*) menyimpan angka desimal seperti nilai IPK atau suhu.
d. *Boolean* digunakan dalam logika pemrograman, bernilai *true* atau *false*.
e. *Array* menyimpan kumpulan nilai sekaligus dalam satu variabel, ideal untuk daftar seperti mata kuliah.
f. *Object* merepresentasikan entitas dari sebuah *class*, menjadi bagian penting dalam konsep *Object-Oriented Programming* (OOP).
g. NULL menunjukkan bahwa variabel belum menyimpan nilai apa pun.

h. Resource digunakan untuk mengelola akses ke sumber daya eksternal, seperti file atau koneksi *database*.

PHP secara otomatis menentukan tipe data berdasarkan nilai yang diberikan (*type juggling*). Meskipun demikian, pemahaman yang baik terhadap tipe data akan membantu dalam penulisan kode yang lebih rapi, efisien, dan bebas kesalahan.

6. PHP Casting

Casting dalam bahasa pemrograman PHP adalah proses mengubah tipe data dari satu bentuk ke bentuk lainnya. PHP merupakan bahasa pemrograman dengan tipe data *loosely typed* (longgar), artinya variabel tidak harus dideklarasikan tipe datanya sejak awal. Namun, dalam situasi tertentu, *programmer* perlu memaksa sebuah variabel agar berperilaku sesuai dengan tipe data tertentu, dan inilah fungsi dari *casting*. Berikut jenis *casting* dalam PHP, yaitu:

a. Casting ke Integer

Contoh Mengubah nilai variabel menjadi bilangan bulat:

```
<?php
$angka = "20.8";
$int = (int)$angka;
echo $int; // Output: 20
?>
```

Pada kode di atas *String* "`20.8`" dipaksa menjadi integer `20`.

b. Casting ke Float (Double/Real)

Contoh mengubah variabel menjadi bilangan desimal:

```
<?php
$angka = "20.8";
$float = (float)$angka;
echo $float; // Output: 20.8
?>
```

Pada kode di atas *String* "`20.8`" diubah menjadi *float* `20.8`

c. Casting ke String

Contoh Mengubah variabel menjadi *string* (teks):

```
<?php
$angka = 50;
$str = (string)$angka;
echo $str; // Output: "50"
?>
```

Pada kode di atas Integer 50 dipaksa menjadi *string* "50".

d. Casting ke Boolean

Contoh mengubah variabel menjadi *true* atau *false*:

```
<?php
$nilai1 = 0;
$nilai2 = 1;
$nilai3 = "";

var_dump((bool)$nilai1); // false
var_dump((bool)$nilai2); // true
var_dump((bool)$nilai3); // false
?>
```

Pada kode di atas untuk aturan umum untuk `0`, `""`, `NULL`, dan *array* kosong = `false`. Selain itu = `true`.

e. Casting ke Array

Contoh mengubah variabel menjadi *array*:

```
<?php
$nama = "Mizar";
$arr = (array)$nama;
print_r($arr);
// Output: Array ( [0] => Mizar )
?>
```

Pada kode di atas *String* `"Mizar"` menjadi *array* dengan 1 elemen

f. Casting ke Object

Contoh Mengubah variabel menjadi *object* standar (stdClass):

```
<?php
$nama = "Dzimar";
$obj = (object)$nama;
echo $obj->scalar; // Output: Dzimar
?>
```

Pada kode di atas *String* `"Dzimar"` dikonversi ke objek dengan properti `scalar`.

g. Casting ke NULL

PHP tidak mendukung casting langsung ke `NULL`. Untuk melakukan cast ke `NULL`, gunakan pernyataan (`unset`), contoh:

```
<?php
```

```
$data = "Belajar Pemrograman PHP";
$data = (unset) $data;

var_dump($data); // NULL
?>
```

Pada kode di atas *output* yang dihasilkan "`NULL`" . *Function* `var_dump()` Digunakan Untuk memverifikasi jenis objek apa pun di PHP.

Casting dalam bahasa pemrograman PHP memungkinkan *programmer* mengontrol tipe data agar sesuai kebutuhan. Salah satu manfaat dengan *casting* bisa menghindari *error* ketika melakukan operasi matematika, manipulasi *string*, maupun pengolahan data lain. Karena PHP bersifat *loosely typed*, *casting* menjadi alat penting untuk menjaga konsistensi tipe data dalam program.

7. PHP Math Functions

PHP menyediakan berbagai fungsi matematika bawaan yang memudahkan *programmer* dalam melakukan operasi perhitungan numerik. Fungsi ini dapat digunakan untuk keperluan sederhana seperti pembulatan angka, hingga operasi yang lebih kompleks seperti perhitungan trigonometri. Fungsi matematika utama dalam PHP yaitu:

a. Nilai Absolut (`abs()`)

Fungsi `abs()` untuk mengembalikan nilai positif dari suatu bilangan. Contoh:

```
<?php
echo abs(-9); // Output: 9
?>
```

Fungsi `abs()` pada contoh kode di atas mengembalikan nilai absolut (positif) suatu angka, yaitu -9 menjadi 9.

b. Pangkat (`pow()`)

Fungsi `pow()` untuk menghitung nilai dari bilangan berpangkat. Contoh:

```
<?php
echo pow(5, 2); // Output: 25 (5^2)
?>
```

Kode di atas merupakan 5 pangkat 2 hasilnya, yaitu 25.

c. Akar Kuadrat (`sqrt()`)

Fungsi `sqrt()` untuk menghitung akar kuadrat dari sebuah angka. Contoh:

```
<?php
echo sqrt(25); // Output: 5
?>
```

Kode di atas merupakan akar kuadrat dari 25 hasilnya, yaitu 5.

d. Pembulatan (`round()`)

Fungsi `round()` untuk membulatkan bilangan desimal ke bilangan bulat terdekat. Contoh:

```
<?php
echo round(5.4);  // Output: 5
echo round(5.6);  // Output: 6
?>
```

Pada kode `round(5.4)` dibulatkan menjadi 5. Sedangkan, kode `round(5.6)` dibulatkan menjadi 6 karena angka di belakang koma lebih dari 5.

e. Pembulatan ke atas (`ceil()`)

Fungsi `ceil()` untuk membulatkan angka desimal ke bilangan bulat terdekat ke atas.Contoh:

```
<?php
echo ceil(5.1); // Output: 6
?>
```

Pada kode `ceil(5.1)` membulatkan angka 5.1 ke atas, yaitu angka 6.

f. Pembulatan ke bawah (`floor()`)

Membulatkan angka desimal ke bilangan bulat terdekat ke bawah. Contoh:

```
<?php
echo floor(5.9); // Output: 5
?>
```

Pada kode `floor()` membulatkan angka 5.9 ke bilangan bulat terdekat ke bawah, yaitu angka 5.

g. Nilai Tertinggi (`max()`) dan Terendah (`min()`)

Fungsi `max()` untuk menentukan nilai terbesar dan `min()` menentukan terkecil dari sekumpulan angka. Contoh:

```
<?php
echo max(3, 8, 2, 9); // Output: 9
echo min(3, 8, 2, 9); // Output: 2
?>
```

Pada kode `max(3, 8, 2, 9)` menentukan nilai terbesarnya, yaitu 9. Dan pada kode `min(3, 8, 2, 9)` menentukan nilai terkecil, yaitu 2.

h. Angka Acak (`rand()`)

Fungsi menghasilkan bilangan acak. Untuk angka yang dihasilkan dapat diberi rentang nilai. Contoh:

```
<?php
echo rand();        // Output: angka acak
echo rand(1, 100); // Output: angka acak
antara 1-100
?>
```

Pada kode `rand()` akan menampilkan angka acak dengan angka yang berbeda-beda. Pada kode `rand(1,100)` akan menampilkan angka acak yang berbeda beda antara 1 dan 100.

i. Fungsi Trigonometri (`sin(), cos(), tan()`)

Fungsi trigonometri `sin(), cos(), tan()` Digunakan untuk perhitungan trigonometri. Contoh:

```
<?php
echo sin(deg2rad(100)); // Output: 0.98
echo "<br>";
echo cos(deg2rad(90)); // Output: 6.12
echo "<br>";
echo tan(deg2rad(80)); // Output: 5.67
?>
```

Pada kode di atas akan menghasilkan perhitungan trigonometri.

j. PHP `pi()` Function

Fungsi `pi()` untuk mengembalikan nilai konstanta Pi (π), yang merupakan sebuah angka yang merepresentasikan perbandingan keliling lingkaran dengan diameternya. Nilai pi

adalah sekitar 3.1415926535898. Fungsi ini sering digunakan dalam perhitungan matematika yang melibatkan lingkaran, seperti luas dan keliling lingkaran, serta dalam perhitungan geometri dan trigonometri lainnya. Contoh:

```
<?php
echo(pi());
?>
```

Kode di atas akan menghasilkan 3.1415926535898.

Fungsi matematika di PHP sangat membantu dalam pengolahan angka, mulai dari perhitungan sederhana hingga perhitungan ilmiah. Dengan memanfaatkan fungsi bawaan ini, *programmer* dapat menulis kode yang lebih efisien, cepat, dan akurat tanpa harus menulis algoritma matematika dari awal.

8. PHP Constants

Konstanta adalah pengenal (*identifier*) untuk menyimpan nilai tetap yang tidak dapat diubah selama kode program dijalankan. Berbeda dengan variabel, konstanta nilainya bersifat *immutable* (tidak dapat diganti atau dihapus setelah didefinisikan). Konstanta sangat berguna untuk menyimpan nilai tetap seperti nama aplikasi, versi, nilai pi (π), konfigurasi *database*, dan informasi lain yang tidak berubah sepanjang program. Nama konstanta yang valid dimulai dengan huruf atau garis bawah (tanpa tanda $ sebelum nama konstanta). Contoh sintaks untuk membuat konstanta menggunakan fungsi `define()`:

```
define(name, value);
```

Keterangan:

name : Menentukan nama konstanta

value : Menentukan nilai konstanta

contoh:

```
<?php
define("GREETING", "Selamat datang di
pemrograman PHP");
echo GREETING;
?>
```

Output dari kode di atas:

```
Selamat datang di pemrograman PHP
```

Penggunaan nama konstanta *case sensitive* atau peka huruf besar-kecil, penamaannya harus konsisten nama konstanta

yang dideklarasikan dengan nama konstanta yang di panggil. Seperti contoh di atas nama konstanta "GREETING" sama persis nama konstantanya ketika dipanggil.

Ada beberapa cara untuk membuat konstanta dalam pemrograman PHP, yaitu:

a. Menggunakan `define()`

`define()` bisa digunakan di dalam fungsi, *loop*, atau kondisi. Contoh:

```
<?php
define("judul", "Belajar PHP");
define("PI", 3.14);

echo judul;      // Output: Belajar PHP
echo "<br>";
echo PI;        // Output: 3.14
?>
```

Contoh membuat konstanta *Array* menggunakan fungsi define():

```
<?php
define("mobil", [
  "Toyota",
  "Honda",
  "Mitsubishi"
]);
echo mobil[0];//output : Toyota
?>
```

b. Menggunakan `const`

`const` hanya bisa digunakan pada level global atau dalam *class*. Contoh:

```
<?php
const mata_kuliah = "Web Programming";

echo mata_kuliah; // Output: Web Programming
?>
```

Contoh penggunaan konstanta secara otomatis bersifat global dan dapat digunakan di seluruh kode program:

```
<?php
define("hello",     "Selamat     datang     di
pemrograman PHP");
```

```
function myTest() {
  echo hello;
}

myTest();  //  Output:  Selamat  datang  di
pemrograman PHP
?>
```

Perbedaan penggunaan `const` dengan `define()` . Penggunaan `const` tidak dapat dibuat di dalam lingkup blok lain, seperti di dalam fungsi atau di dalam pernyataan if. Sedangkan penggunaan `define()` dapat dibuat di dalam lingkup blok lain.

Selain konstanta yang bisa kita buat sendiri, PHP juga menyediakan ratusan konstanta bawaan (*predefined constants*). Konstanta ini dibuat otomatis oleh PHP, ekstensi bawaan, maupun *library* yang terpasang. Nilainya sudah ditentukan dan tidak bisa diubah oleh *programmer*. *Predefined constants* biasanya digunakan untuk Mendapatkan informasi sistem (versi PHP, OS, direktori file), *Debugging* & *logging* (mengetahui baris kode, nama file, *class*, *method*), dan Ekstensi PHP tertentu (misalnya E_ALL untuk *error reporting*).

Berikut beberapa konstanta bawaan dalam tabel:

Tabel 1. PHP *Predefined Constants*

Konstanta	Deskripsi
PHP_VERSION	Versi PHP yang digunakan
PHP_OS	Sistem operasi server
__LINE__	Nomor baris dalam file
__FILE__	Path lengkap file PHP
__DIR__	Direktori file saat ini
__FUNCTION__	Nama fungsi saat ini
__CLASS__	Nama *class* saat ini
__METHOD__	Nama *method* saat ini
__NAMESPACE__	*Namespace* saat ini
PHP_INT_MAX	Nilai integer maksimum
PHP_INT_MIN	Nilai integer minimum
PHP_EOL	Karakter akhir baris (*end of line*)
E_ERROR	*Error* fatal (tidak bisa dilanjutkan)

E_WARNING	Peringatan (*script* tetap jalan)
E_NOTICE	Pemberitahuan (misalnya variabel belum didefinisikan)
E_ALL	Menampilkan semua jenis *error*

Contoh penggunaan konstanta bawaan:

```
<?php
echo "=== Informasi Sistem ===<br>";
echo "Versi PHP: " . PHP_VERSION . "<br>";
echo "Sistem Operasi: " . PHP_OS . "<br>";
echo "Integer Maksimum: " . PHP_INT_MAX .
"<br>";
echo "Integer Minimum: " . PHP_INT_MIN . "<br>";
echo "End Of Line: '" . PHP_EOL . "'<br><br>";

echo "=== Magic Constants ===<br>";
echo "File ini: " . __FILE__ . "<br>";
echo "Direktori: " . __DIR__ . "<br>";
echo "Baris kode: " . __LINE__ . "<br>";

function contohFungsi() {
    echo "Nama Fungsi: " . __FUNCTION__ . "<br>";
}
contohFungsi();

class MyClass {
    public function myMethod() {
        echo "Nama Class: " . __CLASS__ . "<br>";
          echo "Nama Method: " . __METHOD__ .
"<br>";
    }
}
$obj = new MyClass();
$obj->myMethod();
?>
```

Output dari kode program di atas:

```
=== Informasi Sistem ===
Versi PHP: 7.1.1-1
Sistem Operasi: Linux
Integer Maksimum: 9223372036854775807
Integer Minimum: -9223372036854775808
End Of Line: ' '
```

```
=== Magic Constants ===
File ini: /home/v1L4KS/prog.php
Direktori: /home/v1L4KS
Baris kode: 12
Nama Fungsi: contohFungsi
Nama Class: MyClass
Nama Method: MyClass::myMethod
```

Konstanta dalam PHP berfungsi untuk menyimpan nilai tetap yang tidak bisa diubah, sehingga membuat kode lebih aman, terstruktur, dan mudah dipelihara. Dengan memanfaatkan konstanta, *programmer* dapat menghindari kesalahan akibat perubahan nilai variabel yang seharusnya tidak diubah.

9. PHP Operators

Operator dalam PHP digunakan untuk melakukan operasi terhadap variabel maupun nilai. PHP menyediakan berbagai jenis operator yang dapat digunakan dalam perhitungan matematis, manipulasi *string*, pengambilan keputusan, hingga manipulasi bit. Dengan memahami operator, *programmer* dapat menulis kode yang lebih ringkas, jelas, dan efisien. Jenis-Jenis Operator dalam PHP, yaitu:

a. Operator Aritmetika

Digunakan untuk perhitungan matematika

Operator	Nama	Contoh	Hasil
+	Penjumlahan	$a + $b	Jumlahkan $a dan $b
-	Pengurangan	$a - $b	Kurangi $a dengan $b
*	Perkalian	$a * $b	Kalikan $a dan $b
/	Pembagian	$a / $b	Bagi $a dengan $b
%	Modulus	$a % $b	Sisa hasil bagi $a / $b
**	Pangkat	$a ** $b	$a pangkat $b

b. Operator Penugasan (*Assignment Operators*)

Digunakan untuk memberikan nilai ke variabel

Operator	Contoh	Sama Dengan
=	$x = 10	$x diberi nilai 10
+=	$x += 5	$x = $x + 5
-=	$x -= 3	$x = $x - 3
*=	$x *= 2	$x = $x * 2

/=	$x /= 4	$x = $x / 4
%=	$x %= 2	$x = $x % 2

c. Operator Perbandingan (*Comparison Operators*)
Digunakan untuk membandingkan dua nilai.

Operator	Contoh	Hasil
==	$a == $b	*True* jika $a sama dengan $b
===	$a === $b	*True* jika $a sama dengan $b **dan tipe data sama**
!=	$a != $b	*True* jika $a tidak sama dengan $b
!==	$a !== $b	*True* jika $a tidak sama dengan $b **atau tipe berbeda**
>	$a > $b	*True* jika $a lebih besar dari $b
<	$a < $b	*True* jika $a lebih kecil dari $b
>=	$a >= $b	*True* jika $a lebih besar atau sama dengan $b
<=	$a <= $b	*True* jika $a lebih kecil atau sama dengan $b
<=>	$a <=> $b	-1 jika $a<$b, 0 jika sama, 1 jika $a>$b

d. Operator Logika (*Logical Operators*)
Digunakan dalam ekspresi logika.

Operator	Contoh	Deskripsi
and	$a and $b	*True* jika keduanya *true*
or	$a or $b	*True* jika salah satu *true*
xor	$a xor $b	*True* jika salah satu *true*, tapi tidak keduanya
&&	$a && $b	Sama dengan *and* (lebih tinggi prioritas)
!	!$a	*True* jika $a *false*

e. Operator *Increment* / *Decrement*
Digunakan untuk menambah atau mengurangi nilai variabel.

Operator	Contoh	Hasil
++$a	Pre-increment	Tambah $a lalu kembalikan nilainya

$a++	Post-increment	Kembalikan nilai $a lalu tambah 1
--$a	Pre-decrement	Kurangi $a lalu kembalikan nilainya
$a--	Post-decrement	Kembalikan nilai $a lalu kurangi 1

f. Operator *String*

Digunakan untuk memanipulasi *string*.

Operator	Contoh	Hasil
.	$txt1 . $txt2	Menggabungkan *string*
.=	$txt1 .= $txt2	Menambahkan *string* ke variabel yang sudah ada

g. Operator *Array*

Digunakan untuk membandingkan dan menggabungkan *array*.

Operator	Contoh	Hasil
+	$a + $b	Menggabungkan *array*
==	$a == $b	*True* jika nilai dan kunci sama
===	$a === $b	*True* jika nilai, kunci, dan urutan sama
!=	$a != $b	*True* jika *array* berbeda
<>	$a <> $b	Sama dengan !=
!==	$a !== $b	*True* jika tidak identik

h. Operator *Bitwise*

Digunakan untuk operasi pada level biner.

Operator	Contoh	Hasil
&	$a & $b	AND *bitwise*
`	`	`$a
^	$a ^ $b	XOR *bitwise*
~	~$a	Negasi *bitwise*
<<	$a << $b	Geser bit ke kiri
>>	$a >> $b	Geser bit ke kanan

Contoh kode program penggunaan operator:

```
<?php
$a = 10;
$b = 3;
```

```
// Operator Aritmatika
echo "Aritmatika:<br>";
echo "$a + $b = " . ($a + $b) . "<br>";
echo "$a - $b = " . ($a - $b) . "<br>";
echo "$a * $b = " . ($a * $b) . "<br>";
echo "$a / $b = " . ($a / $b) . "<br>";
echo "$a % $b = " . ($a % $b) . "<br>";
echo "$a ** $b = " . ($a ** $b) . "<br><br>";

// Operator Perbandingan
echo "Perbandingan:<br>";
var_dump($a == $b); echo "<br>";
var_dump($a != $b); echo "<br>";
var_dump($a > $b); echo "<br>";
var_dump($a <=> $b); echo "<br><br>";

// Operator Logika
echo "Logika:<br>";
var_dump($a > 5 && $b < 5); echo "<br>";
var_dump($a < 5 || $b < 5); echo "<br>";
var_dump(!($a == 10)); echo "<br><br>";

// Operator String
echo "String:<br>";
$txt1 = "Hello";
$txt2 = " World!";
echo $txt1 . $txt2 . "<br>";
$txt1 .= $txt2;
echo $txt1 . "<br><br>";
?>
```

Output dari kode program di atas:

```
Aritmatika:
10 + 3 = 13
10 - 3 = 7
10 * 3 = 30
10 / 3 = 3.3333
10 % 3 = 1
10 ** 3 = 1000

Perbandingan:
```

```
bool(false)
bool(true)
bool(true)
int(1)

Logika:
bool(true)
bool(true)
bool(false)

String:
Hello World!
Hello World!
```

Operator dalam PHP sangat beragam dan mendukung berbagai kebutuhan mulai dari perhitungan matematis, perbandingan logika, manipulasi *string*, hingga operasi pada *array* dan *bitwise*. Pemahaman mendalam mengenai operator membantu *programmer* menulis kode lebih efisien, mudah dibaca, dan sesuai kebutuhan aplikasi.

10. PHP Conditional Statements

Conditional statements dalam PHP digunakan untuk mengontrol alur eksekusi program berdasarkan suatu kondisi (benar atau salah). Dengan pernyataan ini, program dapat membuat keputusan yang berbeda sesuai dengan *input*, data, atau logika yang diberikan. Jenis-jenis *Conditional Statements* di PHP, yaitu:

a. if Statement

Menjalankan kode hanya jika kondisi benar (*true*). Sintaks untuk *if statement*:

```
if (kondisi) {
  // kode yang akan dieksekusi jika kondisinya
benar;
}
```

Contoh:

```
<?php
$nilai = 82;

if ($nilai >= 80) {
    echo "Lulus";
}
```

```
?>
```

Output-nya:

```
Lulus
```

b. if...else Statement

Menjalankan kode tertentu jika kondisi benar (*true*), dan kode lain jika kondisi salah (*false*). Sintaks untuk *if else statement*:

```
if (kondisi) {
  // kode yang akan dieksekusi jika kondisi
benar (true);
} else {
  // kode yang akan dieksekusi jika kondisi
salah (false);
}
```

Contoh:

```
<?php
$nilai = 70;

if ($nilai >= 85) {
    echo "Lulus";
} else {
    echo "Tidak Lulus";
}
?>
```

Output-nya:

```
Tidak Lulus
```

c. if...elseif...else Statement

Pernyataan yang digunakan untuk mengeksekusi kode yang berbeda untuk lebih dari dua kondisi. Sintaks untuk *if elseif else statement*:

```
if (kondisi) {
  //kode akan dieksekusi jika kondisi ini
benar (true);
} elseif (kondisi) {
  // kode akan dieksekusi jika kondisi pertama
salah (false) dan kondisi ini benar (true);
} else {
  // kode akan dieksekusi jika semua kondisi
salah (false);
}
```

Contoh:

```
<?php
$nilai = 80;

if ($nilai >= 90) {
    echo "Grade: A";
} elseif ($nilai >= 75) {
    echo "Grade: B";
} elseif ($nilai >= 60) {
    echo "Grade: C";
} else {
    echo "Grade: D";
}
?>
```

Output-nya:

```
Grade: B
```

d. Switch Statement

Digunakan untuk mengganti banyak *if...elseif* dengan bentuk yang lebih sederhana, biasanya pada nilai diskrit. Penggunaan pernyataan *switch* untuk memilih salah satu dari banyak blok kode yang akan dieksekusi. Sintaks untuk *switch statement*:

```
switch (expression) {
  case label1:
    //code block
    break;
  case label2:
    //code block;
    break;
  case label3:
    //code block
    break;
  default:
    //code block
}
```

Contoh:

```
<?php
$hari = "Senin";

switch ($hari) {
    case "Senin":
```

```
        echo "Hari ini Senin";
        break;
    case "Selasa":
        echo "Hari ini Selasa";
        break;
    case "Rabu":
        echo "Hari ini Rabu";
        break;
    default:
        echo "Hari tidak dikenal";
}
?>
```

Output-nya:

```
Hari ini Senin
```

Pernyataan kondisional (`if`, `if...else`, `if...elseif`, `switch`) memungkinkan PHP membuat keputusan logis dalam menjalankan program. Dengan ini, aplikasi dapat menjadi dinamis, fleksibel, dan responsif terhadap *input*.

11. PHP Loops (Perulangan dalam PHP)

Loops (perulangan) dalam PHP digunakan untuk menjalankan blok kode berulang kali selama kondisi tertentu masih terpenuhi. Dengan perulangan, *programmer* tidak perlu menulis kode yang sama berulang-ulang, cukup satu blok kode yang akan diulang otomatis. Perulangan sangat penting dalam pemrosesan data, seperti menampilkan daftar produk, membaca isi *array*, membuat tabel otomatis, dan lain-lain. Dalam PHP terdapat beberapa jenis perulangan berikut:

a. While

Menjalankan kode selama kondisi benar (*true*). Contoh:

```
<?php
$i = 1;

while ($i <= 5) {
    echo "Angka: $i <br>";
    $i++;
}
?>
```

Output-nya:

```
Angka: 1
Angka: 2
Angka: 3
Angka: 4
Angka: 5
```

b. Do While

Perulangan *do...while* akan selalu mengeksekusi blok kode minimal satu kali, kemudian memeriksa kondisi, dan mengulang perulangan selama kondisi yang ditentukan bernilai benar. Contoh:

```
<?php
$i = 1;

do {
    echo "Angka : $i <br>";
    $i++;
} while ($i <= 5);
?>
```

Output-nya:

```
Angka : 1
Angka : 2
Angka : 3
Angka : 4
Angka : 5
```

c. For

Melakukan pengulangan pada blok kode sejumlah kali yang telah ditentukan. Biasanya digunakan ketika jumlah perulangan sudah diketahui. Sintaks-nya:

```
for (ekspresi1, ekspresi2, ekspresi3) {
  // Blok kode
}
```

Ekspresi 1: di evaluasi sekali

Ekspresi 2: di evaluasi sebelum setiap iterasi

Ekspresi 3: di evaluasi setelah setiap iterasi

Contoh:

```
<?php
for ($i = 1; $i <= 5; $i++) {
    echo "Perulangan ke-$i <br>";
```

```
}
?>
```

Output-nya:

```
Perulangan ke-1
Perulangan ke-2
Perulangan ke-3
Perulangan ke-4
Perulangan ke-5
```

d. Foreach

Melakukan perulangan melalui blok kode untuk setiap elemen dalam *array* atau setiap properti dalam suatu objek.

Contoh:

```
<?php
$buah = ["Apel", "Jeruk", "Mangga",
"Pisang"];

foreach ($buah as $b) {
    echo "Buah: $b <br>";
}
?>
```

Output-nya:

```
Buah: Apel
Buah: Jeruk
Buah: Mangga
Buah: Pisang
```

Dalam penggunaan perulangan terkadang perlu mengontrol alur perulangan yaitu menggunakan pernyataan `break` untuk menghentikan perulangan dan `continue` untuk melewati satu iterasi, lanjut ke iterasi berikutnya.

Contoh:

```
<?php
for ($i = 1; $i <= 10; $i++) {
    if ($i == 5) {
        continue; // Lewati angka 5
    }
    if ($i == 8) {
        break; // Hentikan di angka 8
    }
    echo "Nomor: $i <br>";
}
```

```
?>
```

Output-nya:

```
Nomor: 1
Nomor: 2
Nomor: 3
Nomor: 4
Nomor: 6
Nomor: 7
```

Loop dalam PHP sangat membantu untuk mengulang eksekusi kode secara otomatis, baik dalam jumlah tertentu maupun hingga kondisi tertentu terpenuhi. Ada empat jenis utama (`while`, `do...while`, `for`, dan `foreach`) ditambah kontrol `break` dan `continue` untuk fleksibilitas.

12. PHP Functions (Fungsi dalam PHP)

Fungsi adalah blok kode yang dapat dipanggil berulang kali untuk menjalankan tugas tertentu. Dengan menggunakan fungsi, kode menjadi lebih terstruktur, mudah dipelihara, dan *reusable* (dapat digunakan kembali). PHP memiliki dua jenis fungsi:

a. Built-in Functions

Fungsi bawaan PHP (misalnya strlen(), date(), strtolower()). PHP memiliki lebih dari 1000 fungsi bawaan yang dapat dipanggil langsung dari dalam skrip untuk melakukan tugas tertentu.

b. User-defined Functions

Fungsi yang dibuat sendiri oleh *programmer*. Fungsi adalah blok pernyataan yang dapat digunakan berulang kali dalam suatu program. Fungsi tidak akan dijalankan secara otomatis saat halaman dimuat. Fungsi akan dijalankan dengan memanggil fungsi tersebut.

Berikut jenis-jenis fungsinya:

1) Fungsi Sederhana (Tanpa Parameter dan Return)

Contoh:

```
<?php
function salam() {
    echo "Halo, selamat datang di PHP!<br>";
}

// Memanggil fungsi
```

```
salam();
salam();
?>
```

Output-nya:

```
Halo, selamat datang di PHP!
Halo, selamat datang di PHP!
```

2) Fungsi dengan Parameter

Contoh:

```
<?php
function sapa($nama) {
    echo "Halo, $nama!<br>";
}

// Memanggil fungsi dengan argumen
sapa("Dzimar");
sapa("Ghifar");
?>
```

Output-nya:

```
Halo, Dzimar!
Halo, Ghifar!
```

3) Fungsi dengan Nilai Balik (Return)

Contoh:

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

$hasil = tambah(6, 9);
echo "Hasil penjumlahan: $hasil";
?>
```

Output-nya:

```
Hasil penjumlahan: 15
```

4) Fungsi dengan Nilai Default Parameter

Contoh:

```
<?php
function greet($nama = "Mizar") {
    echo "Selamat datang, $nama!<br>";
}

greet("Dzimar");
```

```
greet();
?>
```

Output-nya:

```
Selamat datang, Dzimar!
Selamat datang, Mizar!
```

5) Fungsi dengan Parameter Tipe Data (Type Hinting)

Contoh:

```
<?php
function bagi(int $a, int $b): float {
    return $a / $b;
}

echo bagi(10, 2);
?>
```

Output-nya:

```
5
```

6) Fungsi Variadik (Parameter Tidak Terbatas)

Contoh:

```
<?php
function total(...$angka) {
    return array_sum($angka);
}

echo "Total: " . total(1, 2, 3, 4, 5);
?>
```

Output-nya:

```
Total : 15
```

7) Fungsi Rekursif

Contoh:

```
<?php
function faktorial($n) {
    if ($n == 0) return 1;
    return $n * faktorial($n - 1);
}

echo "5! = " . faktorial(5);
?>
```

Output-nya:

```
5! = 120
```

8) Anonymous Function (Closure)

Contoh:

```
<?php
$salam = function($nama) {
    return "Halo, $nama!";
};

echo $salam("mizar");
?>
```

Output-nya:

```
Halo, mizar!
```

Fungsi dalam PHP memungkinkan *programmer* membuat kode yang modular, efisien, dan mudah digunakan kembali. Dengan adanya parameter, *return value*, *default value*, *variadic*, hingga *rekursif*, fungsi menjadi elemen kunci dalam perancangan program berorientasi objek maupun prosedural.

13. PHP Arrays

Array adalah struktur data yang digunakan untuk menyimpan sekumpulan nilai dalam satu variabel. Berbeda dengan variabel biasa yang hanya menyimpan satu nilai, *array* bisa menyimpan banyak nilai dengan indeks tertentu. Di PHP, terdapat tiga jenis *array* utama:

a. Indexed Array

Array dengan indeks berupa angka yang dimulai dari 0 atau menggunakan indeks numerik.. Contoh:

```
<?php
$buah = array("Apel", "Mangga", "Jeruk");

// Akses elemen array
echo $buah[0]; // Output: Apel
echo "<br>";
echo $buah[1]; // Output: Mangga
echo "<br>";
echo $buah[2]; // Output: Jeruk

// Looping array
foreach($buah as $item) {
    echo $item . "<br>";
}
?>
```

Output-nya:

```
Apel
Mangga
JerukApel
Mangga
Jeruk
```

b. Associative Array

Array dengan *key* berupa *string* (nama) yang lebih bermakna atau menggunakan *key* (nama) sebagai indeks. Contoh:

```
<?php
$mahasiswa = array(
    "nama" => "Dzimar Farid",
    "umur" => 20,
    "jurusan" => "Informatika"
);

// Akses array
echo "Nama: " . $mahasiswa["nama"] . "<br>";
echo "Umur: " . $mahasiswa["umur"] . "<br>";
echo "Jurusan: " . $mahasiswa["jurusan"] .
"<br>";
?>
```

Output-nya:

```
Nama: Dzimar Farid
Umur: 20
Jurusan: Informatika
```

c. Multidimensional Array

Array yang berisi *array* lain (bertingkat) atau *Array* di dalam *array*. Contoh:

```
<?php
$matrik = array(
    array(1, 2, 3),
    array(4, 5, 6),
    array(7, 8, 9)
);

echo $matrik[0][1]; // Output: 2
echo "<br>";
echo $matrik[2][2]; // Output: 9
?>
```

Output-nya:

```
2
9
```

Array di PHP sangat penting untuk menyimpan banyak data sekaligus. *Indexed array* digunakan untuk daftar dengan urutan angka, *associative array* digunakan jika membutuhkan *key* yang lebih bermakna, sedangkan multidimensional *array* digunakan untuk data kompleks seperti tabel atau matriks. Fungsi-fungsi *array* bawaan PHP membuat pengolahan data lebih mudah dan efisien.

14. PHP Global Variables – Superglobals

Superglobals adalah variabel bawaan PHP yang tersedia di seluruh bagian *script*, tanpa perlu melakukan deklarasi global. Superglobals dapat diakses langsung di dalam fungsi, *class*, maupun file lain. Superglobals sangat penting dalam pengolahan data *form*, *request*, *session*, *cookie*, server, dan variabel global. Berikut daftar Superglobals di PHP:

a. $GLOBALS

Menyimpan semua variabel global dalam bentuk array asosiatif. Contoh:

```
<?php
$x = 10;
$y = 20;

function tambah() {
    $GLOBALS['z'] = $GLOBALS['x'] +
$GLOBALS['y'];
}

tambah();
echo $z; // Output: 30
?>
```

b. $_SERVER

Menyimpan informasi server dan environment. Contoh:

```
<?php
echo $_SERVER['PHP_SELF'];   // Nama file
yang sedang dijalankan
echo "<br>";
echo $_SERVER['SERVER_NAME']; // Nama server
```

```
echo "<br>";
echo $_SERVER['HTTP_HOST'];   // Host
echo "<br>";
echo $_SERVER['SCRIPT_FILENAME']; // Path
lengkap file
?>
```

Output-nya:

```
/index.php
localhost
localhost
C:/xampp/htdocs/index.php
```

c. $_REQUEST

Menyimpan data dari $_GET, $_POST, dan $_COOKIE.

Contoh:

```
<?php
// form.html
// <form method="post">
//   Nama: <input type="text" name="nama">
//   <input type="submit">
// </form>

if ($_REQUEST["nama"]) {
    echo "Halo, " . $_REQUEST["nama"];
}
?>
```

d. $_POST

Menyimpan data yang dikirim melalui metode POST.

Contoh:

```
<?php
if ($_POST["username"]) {
    echo "Username: " . $_POST["username"];
}
?>
```

e. $_GET

Menyimpan data yang dikirim melalui metode GET (URL).

Contoh:

```
<?php
// URL: http://localhost/index.php?nama=Budi
echo "Halo, " . $_GET["nama"]; // Output:
Halo, Budi
```

```
?>
```

f. $_FILES

Menyimpan informasi file yang di *upload* melalui *form*. Contoh:

```
<?php
// <form method="post"
enctype="multipart/form-data">
//   <input type="file" name="fileUpload">
//   <input type="submit">
// </form>

if ($_FILES["fileUpload"]["name"]) {
    echo "Nama file: " .
$_FILES["fileUpload"]["name"];
}
?>
```

g. $_ENV

Menyimpan variabel *environment* (bergantung konfigurasi server). Contoh:

```
<?php
print_r($_ENV);
?>
```

h. $_COOKIE

Menyimpan data *cookie*. Contoh:

```
<?php
setcookie("user", "Andi", time()+3600); //
buat cookie berlaku 1 jam
echo $_COOKIE["user"];
?>
```

i. $_SESSION

Menyimpan data *session* untuk identifikasi *user*. Contoh:

```
<?php
session_start();
$_SESSION["nama"] = "Siti";
echo $_SESSION["nama"]; // Output: Siti
?>
```

Superglobals di PHP adalah variabel khusus yang sangat berguna dalam pengembangan aplikasi web. Dengan superglobals, kita dapat mengakses variabel global, data *form*,

request, file, *session*, *cookie*, serta informasi server tanpa harus melakukan deklarasi tambahan.

15. PHP Regular Expressions

Regular Expressions (RegEx) adalah pola teks yang digunakan untuk pencarian, pencocokan (*matching*), dan manipulasi *string*. PHP mendukung RegEx melalui fungsi preg_* seperti preg_match(), preg_match_all(), dan preg_replace(). RegEx sangat berguna untuk Validasi *input* (email, nomor telepon, *password*), Pencarian kata atau pola tertentu dalam teks, dan mengganti teks berdasarkan pola tertentu. Berikut simbol dasar dalam *Regex*:

Simbol	Deskripsi	Contoh
.	Cocok dengan karakter apapun kecuali newline	/P.H/ cocok dengan PHP, PAH
^	Cocok di awal string	/^Halo/ cocok dengan "Halo Dunia"
$	Cocok di akhir string	/PHP$/ cocok dengan "Belajar PHP"
[]	Karakter dalam set	/[abc]/ cocok dengan a, b, atau c
[^]	Karakter selain dalam set	/[^0-9]/ cocok selain angka
*	0 atau lebih pengulangan	/go*/ cocok dengan g, go, goo
+	1 atau lebih pengulangan	/go+/ cocok dengan go, goo
?	0 atau 1 kemunculan	/colou?r/ cocok color atau colour
{n}	Tepat n kali	/[0-9]{3}/ cocok dengan 123
{n,}	n kali atau lebih	/[0-9]{2,}/ cocok 12, 1234
{n,m}	Antara n sampai m kali	/[0-9]{2,4}/ cocok 12, 123, 1234
`	`	OR (pilihan)
()	Grup pola	/(abc)+/ cocok abc, abcabc

Contoh:

```php
<?php
$teks = "Belajar PHP itu menyenangkan. PHP mudah
dipelajari.";

// 1. Cari kata "PHP"
if (preg_match("/PHP/", $teks)) {
    echo "Kata PHP ditemukan!<br>";
}

// 2. Cari semua kata "PHP"
preg_match_all("/PHP/", $teks, $hasil);
echo "Jumlah kata PHP: " . count($hasil[0]) .
"<br>";

// 3. Ganti "PHP" menjadi "JavaScript"
$baru = preg_replace("/PHP/", "JavaScript",
$teks);
echo "Teks baru: " . $baru . "<br>";
?>
```

Output:

```
Kata PHP ditemukan!
Jumlah kata PHP: 2
Teks baru: Belajar Javascript itu menyenangkan.
Javascript mudah dipelajari.
```

RANGKUMAN

1. Pengertian PHP
   - PHP (Hypertext Preprocessor) adalah bahasa pemrograman server-side scripting yang digunakan untuk membuat aplikasi web dinamis dan interaktif.
   - PHP pertama kali dikembangkan oleh Rasmus Lerdorf pada tahun 1995, dan kini terus berkembang dengan dukungan komunitas global.
2. Karakteristik dan Keunggulan PHP
   - Bersifat open-source dan gratis.
   - Mudah dipelajari bagi pemula.

- Fleksibel dan dapat berjalan di berbagai sistem operasi (Windows, Linux, macOS).
- Kompatibel dengan banyak server web (Apache, Nginx, IIS).
- Terintegrasi dengan baik dengan berbagai database (MySQL, PostgreSQL, MariaDB, SQLite, dll.).
- Memiliki banyak fungsi bawaan (built-in functions) dan mendukung OOP (Object-Oriented Programming).

3. Lingkungan Pengembangan PHP
   - PHP dapat dijalankan menggunakan web server seperti Apache atau Nginx.
   - Umumnya dipaketkan dalam software seperti XAMPP, LAMP, atau MAMP.
   - File PHP memiliki ekstensi .php dan dapat berisi kombinasi kode HTML, CSS, JavaScript, dan PHP.
4. Dasar Pemrograman PHP
   - Sintaks dasar PHP ditulis di dalam tag <?php ... ?>.
   - Komentar dapat ditulis dengan //, #, atau /* ... */.
   - Variabel diawali dengan simbol $ dan bersifat loosely typed (PHP tidak perlu deklarasi tipe variabel secara eksplisit).
   - Tipe data dalam PHP meliputi: String, Integer, Float, Boolean, Array, Object, NULL, dan Resource.
5. Operator dalam PHP
   - Aritmatika (+, -, *, /, %).
   - Penugasan (=, +=, -=, dll.).
   - Perbandingan (==, !=, <, >, <=, >=).
   - Logika (&&, ||, !).
   - Increment/Decrement (++/--).
6. Struktur Kendali
   - Percabangan (conditional statements): if, if-else, if-elseif-else, dan switch.
   - Perulangan (loops): for, while, do-while, dan foreach.
7. Fungsi (Functions)
   - PHP mendukung fungsi bawaan (built-in functions) dan fungsi buatan pengguna (user-defined functions).

- Fungsi digunakan untuk memecah program menjadi bagian-bagian kecil agar lebih modular, mudah dibaca, dan dapat digunakan kembali.

8. Array dalam PHP
   - Indexed array: array dengan indeks numerik.
   - Associative array: array dengan indeks berupa string.
   - Multidimensional array: array di dalam array.
9. Superglobals dalam PHP
   - Variabel global bawaan PHP yang dapat diakses dari mana saja, seperti:
     - $_GET, $_POST → mengambil data dari form.
     - $_SERVER → informasi server dan request.
     - $_SESSION, $_COOKIE → manajemen sesi dan cookie.
     - $_FILES → upload file.
10. Regular Expressions (Regex)
    - Digunakan untuk pencocokan pola teks.
    - Fungsi utama: preg_match(), preg_replace(), preg_split().
11. Dasar OOP dalam PHP (pengantar ke bab berikutnya)
    - PHP mendukung Object-Oriented Programming.
    - Konsep utama OOP: class, object, encapsulation, inheritance, polymorphism.
    - OOP membuat program lebih terstruktur, reusable, dan mudah dikembangkan.

LATIHAN

1. Pilihan Ganda

1. Siapa pencipta bahasa pemrograman PHP?
   a. Guido van Rossum
   b. James Gosling
   c. Rasmus Lerdorf
   d. Brendan Eich

2. Ekstensi default untuk file PHP adalah …
   a. .html
   b. .php
   c. .phtml
   d. .xml

3. Manakah tipe data berikut yang tidak tersedia di PHP?
   a. String
   b. Integer
   c. Boolean
   d. Character

4. Pernyataan if ($x > 10) { echo "besar"; } else { echo "kecil"; } akan menampilkan kata "besar" jika …
   a. Nilai $x lebih kecil dari 10
   b. Nilai $x sama dengan 10
   c. Nilai $x lebih besar dari 10
   d. Semua jawaban benar

5. Fungsi bawaan PHP untuk menghitung panjang *string* adalah …
   a. strlen()
   b. count()
   c. strcount()
   d. size()

2. Praktikum / Tugas

1. Buat program PHP untuk menampilkan biodata sederhana menggunakan variabel (misalnya: nama, umur, alamat, hobi).
2. Buat program PHP yang menampilkan angka 1 sampai 10 menggunakan perulangan for.
3. Buat fungsi PHP bernama luasPersegi($sisi) yang menghitung luas persegi, lalu uji dengan beberapa nilai sisi.
4. Buat array berisi 5 nama mahasiswa, lalu tampilkan seluruh nama menggunakan perulangan foreach.

5. Buat program PHP sederhana untuk mengecek apakah sebuah angka ganjil atau genap.

BAB 3
Kelas dan Objek dalam PHP

DESKRIPSI:

Bab ini membahas inti dari pemrograman berorientasi objek (OOP) dalam PHP, yaitu kelas (*class*) dan objek (*object*). Kelas merupakan *blueprint* atau cetak biru yang mendefinisikan atribut (properti) dan perilaku (*method*), sedangkan objek adalah instansiasi dari kelas yang nyata digunakan dalam program.

TUJUAN PEMBELAJARAN:

Setelah mempelajari bab ini, mahasiswa diharapkan mampu:

1. Menjelaskan konsep dasar kelas dan objek dalam OOP menggunakan PHP.
2. Membuat kelas sederhana dengan atribut dan *method*.
3. Menginstansiasi kelas menjadi objek dan menggunakannya dalam program.
4. Memahami penggunaan konstruktor dan destruktor untuk inisialisasi dan pembersihan objek.
5. Menerapkan prinsip pemrograman berorientasi objek secara bertahap dalam aplikasi PHP sederhana.

A. Definisi Kelas dan Objek dalam PHP

Kelas adalah *blueprint* atau cetak biru yang mendefinisikan struktur data dan perilaku objek. Di dalam kelas, dapat mendefinisikan properti (variabel) dan *method* (fungsi). Kelas tidak akan bekerja sendiri, tetapi harus di-instansiasi menjadi objek.

Objek adalah hasil instansiasi dari sebuah kelas. Objek memiliki data (properti) dan perilaku (*method*) sesuai dengan definisi yang ada di kelas. Melalui objek inilah program dijalankan. Contoh kode: kelas dan objek dalam PHP:

```php
<?php
// Membuat kelas Mahasiswa
class Mahasiswa {
    // Properti (atribut)
    public $nama;
    public $nim;
    public $jurusan;

    // Method (fungsi dalam class)
    public function tampilkanData() {
        echo "Nama: " . $this->nama . "<br>";
        echo "NIM: " . $this->nim . "<br>";
       echo "Jurusan: " . $this->jurusan . "<br>";
    }
}

// Membuat objek dari kelas Mahasiswa
$mhs1 = new Mahasiswa();
$mhs1->nama = "Mizard Farid";
$mhs1->nim = "123456";
$mhs1->jurusan = "Teknik Informatika";

// Memanggil method untuk menampilkan data
$mhs1->tampilkanData();
?>
```

Output-nya:

```
Nama: Mizard Farid
NIM: 123456
Jurusan: Teknik Informatika
```

Penjelasan kode program di atas, yaitu:

1. `class Mahasiswa { ... }`
   - Mendefinisikan sebuah kelas bernama Mahasiswa.
2. Properti `$nama, $nim, $jurusan`
   - Variabel di dalam kelas yang menyimpan data mahasiswa.
   - Kata kunci *public* berarti properti ini bisa diakses dari luar kelas.
3. Method `tampilkanData()`
   - Fungsi di dalam kelas yang digunakan untuk menampilkan data mahasiswa.

- `$this->nama` digunakan untuk mengakses properti dari objek saat ini.

4. `$mhs1 = new Mahasiswa();`
   - Membuat objek baru bernama $mhs1 dari kelas Mahasiswa.
5. `$mhs1->nama = "Mizard Farid";`
   - Memberikan nilai ke properti `$nama` dari objek `$mhs1`.
6. `$mhs1->tampilkanData();`
   - Memanggil method dari objek `$mhs1` untuk menampilkan data mahasiswa.

B. Properti dan Method dalam PHP

Dalam pemrograman berorientasi objek di PHP, sebuah kelas terdiri atas dua komponen utama, yaitu properti dan *method*. Properti adalah variabel yang didefinisikan di dalam kelas dan berfungsi untuk menyimpan data atau informasi yang dimiliki oleh objek. Properti pada dasarnya menggambarkan karakteristik atau atribut dari suatu objek. Misalnya, sebuah kelas Buku dapat memiliki properti berupa judul, penulis, dan harga. Setiap objek yang dibuat dari kelas tersebut akan memiliki nilai properti masing-masing sesuai dengan data yang dimilikinya.

Sementara itu, *method* adalah fungsi yang didefinisikan di dalam kelas dan merepresentasikan perilaku atau aksi yang dapat dilakukan oleh objek. Jika properti menggambarkan atribut, maka *method* menggambarkan apa yang bisa dilakukan oleh objek. Sebagai contoh, dalam kelas Buku terdapat *method* getInfoBuku() yang berfungsi menampilkan informasi buku, atau setBuku() yang digunakan untuk mengatur nilai properti. Baik properti maupun *method* memiliki tingkat akses (*visibility*) seperti *public*, *protected*, dan *private* yang menentukan apakah komponen tersebut dapat diakses dari luar kelas, hanya dari dalam kelas, atau juga dapat diakses oleh kelas turunan. Dengan adanya properti dan *method*, sebuah objek menjadi entitas yang lengkap karena memiliki data sekaligus perilaku.

Contoh kode properti dan *method*:

```
<?php
// Definisi kelas Buku
class Buku {
    // Properti
    public $judul;
    public $penulis;
    private $harga;

    // Method untuk mengatur nilai properti
     public function setBuku($judul, $penulis,
$harga) {
        $this->judul = $judul;
        $this->penulis = $penulis;
        $this->harga = $harga;
    }

    // Method untuk menampilkan informasi buku
    public function getInfoBuku() {
        echo "Judul: " . $this->judul . "<br>";
       echo "Penulis: " . $this->penulis . "<br>";
         echo "Harga: Rp" . number_format($this-
>harga, 0, ',', '.') . "<br>";
    }
}

// Membuat objek dari kelas Buku
$buku1 = new Buku();
$buku1->setBuku("Belajar OOP dengan PHP",
"Dzimar", 75000);
$buku1->getInfoBuku();
?>
```

Output-nya:

```
Judul: Belajar OOP dengan PHP
Penulis: Dzimar
Harga: Rp75.000
```

Penjelasan kode di atas, yaitu:

1. `class Buku {...}`
   - Mendefinisikan kelas Buku.

2. Properti
   - `$judul` dan `$penulis` dibuat *public*, artinya bisa diakses langsung dari luar kelas.
   - `$harga` dibuat *private*, artinya hanya bisa diakses melalui *method* di dalam kelas.
3. Method `setBuku()`
   - Digunakan untuk mengatur nilai properti kelas.
   - Parameter `$judul, $penulis, $harga` diisi saat pemanggilan *method*.
4. Method `getInfoBuku()`
   - Digunakan untuk menampilkan informasi dari properti kelas.
   - `$this->` digunakan untuk mengakses properti dari objek saat ini.
5. Objek `$buku1`
   - Dibuat dari kelas Buku menggunakan `new Buku()`.
   - Nilai properti diatur dengan *method* `setBuku()`.
   - Data ditampilkan dengan `getInfoBuku()`.

C. Konstruktor dan Destruktor dalam PHP

Dalam PHP, terdapat dua jenis *method* khusus yang berhubungan dengan proses pembuatan dan penghancuran objek, yaitu konstruktor dan destruktor. Konstruktor adalah *method* khusus yang otomatis dijalankan ketika sebuah objek dibuat dari sebuah kelas. *Method* ini biasanya digunakan untuk melakukan inisialisasi nilai awal pada properti objek atau untuk menyiapkan kondisi tertentu agar objek dapat digunakan dengan baik. Konstruktor di PHP didefinisikan dengan nama method __construct().

Sebaliknya, destruktor adalah *method* khusus yang otomatis dijalankan ketika sebuah objek tidak lagi digunakan atau dihapus dari memori. Destruktor biasanya dipakai untuk melakukan pembersihan (*clean up*), seperti menutup koneksi *database*, menghapus file sementara, atau melepaskan *resource* lain. Destruktor di PHP ditulis dengan nama method __destruct().Dengan memahami konstruktor dan destruktor, *programmer* dapat mengatur siklus hidup sebuah objek dengan lebih baik, mulai dari proses penciptaan (*creation*) hingga penghancuran (*destruction*).

Contoh kode program dengan konstruktor & destruktor:

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

    // Konstruktor: dijalankan saat objek dibuat
     public function __construct($nama, $nim,
$jurusan) {
        $this->nama = $nama;
        $this->nim = $nim;
        $this->jurusan = $jurusan;

        echo "Objek Mahasiswa bernama $this->nama
berhasil dibuat.<br>";
    }

    // Method untuk menampilkan data
    public function tampilkanData() {
        echo "Nama: " . $this->nama . "<br>";
        echo "NIM: " . $this->nim . "<br>";
          echo "Jurusan: " . $this->jurusan .
"<br><br>";
    }

     // Destruktor: dijalankan saat objek
dihancurkan
    public function __destruct() {
        echo "Objek Mahasiswa bernama $this->nama
telah dihapus.<br>";
    }
}

// Membuat objek Mahasiswa dengan konstruktor
$mhs1 = new Mahasiswa("Dzimar Farid", "123456",
"Teknik Informatika");
$mhs1->tampilkanData();
```

```
$mhs2 = new Mahasiswa("Ghifar Farid", "654321",
"Sistem Informasi");
$mhs2->tampilkanData();
?>
```

Output-nya:

```
Objek Mahasiswa bernama Dzimar Farid berhasil
dibuat.
Nama: Dzimar Farid
NIM: 123456
Jurusan: Teknik Informatika

Objek Mahasiswa bernama Ghifar Farid berhasil
dibuat.
Nama: Ghifar Farid
NIM: 654321
Jurusan: Sistem Informasi

Objek Mahasiswa bernama Ghifar Farid telah dihapus.
Objek Mahasiswa bernama Dzimar Farid telah dihapus.
```

Penjelasan kode program di atas:

1. `__construct($nama, $nim, $jurusan)`
   - Konstruktor otomatis dipanggil saat objek dibuat.
   - Properti `$nama, $nim, $jurusan` langsung diberi nilai.
2. `tampilkanData()`
   - Menampilkan informasi mahasiswa berdasarkan nilai properti.
3. `__destruct()`
   - Otomatis dipanggil ketika objek tidak digunakan lagi (misalnya saat script selesai dieksekusi).
4. `$mhs1` dan `$mhs2`
   - Dua objek mahasiswa dibuat dengan data berbeda

D. Keyword `$this` dalam PHP

Dalam PHP, *keyword* `$this` digunakan di dalam sebuah kelas untuk merujuk pada objek saat ini (*the current object*). Dengan kata lain, $this adalah referensi yang menunjuk ke objek yang sedang dibuat atau dijalankan. *Keyword* ini sangat penting karena

memungkinkan untuk mengakses properti dan *method* yang dimiliki oleh objek dari dalam kelas itu sendiri. Tanpa $this, PHP tidak bisa membedakan apakah variabel atau *method* yang dipanggil berasal dari dalam kelas atau dari luar. Dengan menggunakan $this, dapat Mengakses properti objek (contoh: `$this->nama`), Memanggil *method* lain dalam kelas (contoh: `$this->tampilkanData()`), dan Menjaga agar kode tetap rapi, mudah dipahami, dan sesuai dengan prinsip OOP.

Contoh kode program penggunaan $this:

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

    // Konstruktor
     public function __construct($nama, $nim, $jurusan) {
        $this->nama = $nama;        // menggunakan $this untuk akses properti
        $this->nim = $nim;
        $this->jurusan = $jurusan;
    }

    // Method untuk menampilkan data
    public function tampilkanData() {
        echo "Nama: " . $this->nama . "<br>";
        echo "NIM: " . $this->nim . "<br>";
       echo "Jurusan: " . $this->jurusan . "<br>";
    }

    // Method lain memanggil method tampilkanData dengan $this
    public function perkenalan() {
         echo "Halo, perkenalkan saya " . $this->nama . "<br>";
        echo "Berikut biodata saya:<br>";
           $this->tampilkanData();   //  memanggil method lain dengan $this
        echo "<br>";
```

```
    }
}

// Membuat objek
$mhs1 = new Mahasiswa("Mizard Farid", "220001",
"Teknik Informatika");
$mhs2 = new Mahasiswa("Nayla Fathia", "220002",
"Sistem Informasi");

// Memanggil method perkenalan
$mhs1->perkenalan();
$mhs2->perkenalan();
?>
```

Output-nya:

```
Halo, perkenalkan saya Mizard Farid
Berikut biodata saya:
Nama: Mizard Farid
NIM: 220001
Jurusan: Teknik Informatika

Halo, perkenalkan saya Nayla Fathia
Berikut biodata saya:
Nama: Nayla Fathia
NIM: 220002
Jurusan: Sistem Informasi
```

Penjelasan kode program:

1. `$this->nama = $nama;`
   - Menghubungkan parameter `$nama` dengan properti `$nama` milik objek.
2. `$this->tampilkanData();`
   - Memanggil method `tampilkanData()` dari dalam method `perkenalan()`.
3. `$this` selalu menunjuk pada objek yang sedang aktif:
   - Pada `$mhs1->perkenalan();`, `$this` menunjuk ke objek `$mhs1`.
   - Pada `$mhs2->perkenalan();`, `$this` menunjuk ke objek `$mhs2`.

RANGKUMAN

1. Kelas (`Class`) adalah cetak biru (*blueprint*) atau rancangan yang berisi properti (variabel) dan *method* (fungsi) yang mendefinisikan perilaku suatu objek.
2. Objek (`Object`) adalah instansi nyata yang dibuat dari sebuah kelas. Setiap objek dapat memiliki data dan perilaku sendiri meskipun berasal dari kelas yang sama.
3. Konstruktor (`__construct()`) adalah *method* khusus yang dijalankan otomatis saat objek dibuat. Fungsinya untuk menginisialisasi nilai awal dari properti atau menyiapkan kondisi tertentu.
4. Destruktor (`__destruct()`) adalah *method* khusus yang dijalankan otomatis saat objek dihapus dari memori. Umumnya digunakan untuk membersihkan *resource* seperti menutup koneksi *database* atau file.
5. Keyword `$this` digunakan untuk merujuk pada objek saat ini. Dengan $this, kita bisa mengakses properti maupun memanggil *method* dalam kelas yang sama.
6. Konsep kelas dan objek adalah dasar dari pemrograman berorientasi objek (OOP) yang membuat kode lebih terstruktur, *reusable*, dan mudah dikelola.

LATIHAN

1. Pertanyaan Teori

1. Jelaskan perbedaan antara kelas dan objek dalam PHP.
2. Apa fungsi dari konstruktor dan destruktor dalam sebuah kelas?
3. Mengapa *keyword* $this penting dalam OOP di PHP? Berikan contoh penggunaannya.
4. Apa keuntungan menggunakan OOP dibandingkan pemrograman prosedural?

2. Latihan Praktik

1. Buatlah sebuah kelas `Buku` dengan properti: `judul`, `penulis`, dan `tahunTerbit`.
   - Tambahkan konstruktor untuk mengisi data awal.
   - Tambahkan *method* `tampilkanInfo()` untuk menampilkan informasi buku.
   - Buat minimal 2 objek buku berbeda dan tampilkan datanya.
2. Buatlah sebuah kelas `Mobil` dengan properti: `merk`, `warna`, dan `kecepatan`.
   - Gunakan konstruktor untuk menginisialisasi data mobil.
   - Buat method `jalan()` yang menampilkan teks "`Mobil [merk] sedang berjalan dengan kecepatan [kecepatan] km/jam`".
   - Buat objek dari kelas tersebut dan jalankan method `jalan()`.
3. Buatlah kelas Mahasiswa dengan properti: `nama`, `nim`, dan `jurusan`.
   - Buat method `perkenalan()` yang menampilkan: `Halo, nama saya [nama], NIM [nim], dari jurusan [jurusan].`
   - Gunakan keyword `$this` untuk mengakses properti di dalam method.
   - Buat minimal 3 objek mahasiswa dan tampilkan hasil perkenalan mereka.

BAB 4
Enkapsulasi dan Aksesibilitas dalam PHP

DESKRIPSI:

Bab ini membahas konsep enkapsulasi (*encapsulation*) dan aksesibilitas (*access modifiers*) dalam pemrograman berorientasi objek (OOP) menggunakan PHP. Enkapsulasi adalah prinsip OOP yang menyembunyikan detail implementasi sebuah kelas agar hanya dapat diakses melalui cara tertentu. Dengan enkapsulasi, kita bisa mengontrol bagaimana data dalam sebuah objek dapat diakses atau diubah. Dalam PHP, pengaturan aksesibilitas dilakukan menggunakan *access modifiers*, seperti *public*, *private*, dan *protected*. Pemahaman tentang enkapsulasi dan aksesibilitas ini sangat penting untuk membangun program yang lebih aman, terstruktur, dan mudah dipelihara.

TUJUAN PEMBELAJARAN:

Setelah mempelajari bab ini, mahasiswa diharapkan mampu:

1. Menjelaskan konsep enkapsulasi dalam OOP.
2. Memahami perbedaan tingkat aksesibilitas (*public*, *private*, dan *protected*).
3. Menggunakan *getter* dan *setter* untuk mengakses dan memodifikasi data dalam objek secara aman.
4. Menerapkan enkapsulasi dalam pembuatan kelas PHP untuk menjaga keamanan data.
5. Membandingkan keuntungan penggunaan enkapsulasi dengan kode tanpa enkapsulasi.

A. Pengertian Enkapsulasi

Enkapsulasi (*encapsulation*) adalah salah satu pilar utama dalam pemrograman berorientasi objek (OOP). Konsep ini berarti penyembunyian detail implementasi dari sebuah kelas sehingga data hanya dapat diakses melalui cara tertentu. Dengan enkapsulasi, data atau properti dalam kelas dapat dijaga keamanannya agar tidak sembarangan diubah dari luar kelas. Dalam PHP, enkapsulasi diwujudkan dengan penggunaan *access modifiers*, seperti *public*, *private*, dan *protected*. *Public,* yaitu properti atau *method* dapat diakses dari mana saja (luar kelas, dalam kelas, maupun turunan). *Private*, yaitu hanya dapat diakses dari dalam kelas itu sendiri. *Protected*, yaitu hanya dapat diakses dari dalam kelas itu sendiri dan kelas turunannya. Dengan demikian, enkapsulasi membantu untuk melindungi data dari perubahan sembarangan, mengontrol akses ke properti dan *method*, menyediakan *interface* khusus (melalui *method getter* dan *setter*) untuk interaksi dengan data, dan membuat kode lebih rapi, aman, dan mudah dipelihara.

Contoh kode enkapsulasi:

```
<?php
class Mahasiswa {
    // Properti dengan enkapsulasi
    private $nama;
    private $nim;

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

    // Getter untuk mengakses nilai nama
    public function getNama() {
        return $this->nama;
    }

    // Setter untuk mengubah nilai nama
    public function setNama($namaBaru) {
        $this->nama = $namaBaru;
    }
```

```
    // Method untuk menampilkan data mahasiswa
    public function tampilkanData() {
        echo "Nama: " . $this->nama . "<br>";
        echo "NIM: " . $this->nim . "<br>";
    }
}

// Membuat objek
$mhs1 = new Mahasiswa("Mizard Farid", "220001");

// Akses data menggunakan method
$mhs1->tampilkanData();

// Mengubah nama dengan setter
$mhs1->setNama("Ghifar Farid");

// Menampilkan data setelah perubahan
$mhs1->tampilkanData();
?>
```

Output-nya:

```
Nama: Mizard Farid
NIM: 220001
Nama: Ghifar Farid
NIM: 220001
```

Penjelasan kode di atas:

1. Properti `$nama` dan `$nim` dibuat *private*, sehingga tidak bisa diakses langsung dari luar kelas, Misalnya, `$mhs1->nama` akan *error*.
2. Untuk mengakses dan mengubah data, digunakan *getter* (`getNama()`) dan *setter* (`setNama()`).
3. Dengan cara ini, data mahasiswa lebih aman karena hanya dapat dimanipulasi melalui *method* yang disediakan.

B. Access Modifiers

Dalam OOP, *access modifiers* digunakan untuk menentukan siapa yang boleh mengakses properti dan *method* dalam sebuah kelas. *Access modifiers* dapat membatasi akses ke data agar tidak

sembarangan diubah dari luar kelas. Di PHP terdapat tiga jenis *access modifiers* utama:

1. Public

   Properti atau *method* yang dideklarasikan sebagai *public* dapat diakses dari mana saja yaitu dari dalam kelas, luar kelas (langsung melalui objek), dan dari kelas turunan (*inheritance*). Cocok untuk *method* atau data yang memang ingin dibuka untuk umum.

2. Private

   Properti atau *method* yang dideklarasikan sebagai *private* hanya bisa diakses dari dalam kelas itu sendiri. Tidak bisa diakses dari luar kelas atau dari kelas turunan. *Private* cocok untuk data yang benar-benar harus dilindungi.

3. Protected

   Properti atau *method* yang dideklarasikan sebagai *protected* hanya bisa diakses dari yaitu dalam kelas itu sendiri, kelas turunan (*subclass*). Tidak bisa diakses langsung dari luar kelas. *Protected* cocok untuk data atau *method* yang tidak boleh diakses bebas, tapi tetap bisa dipakai oleh kelas yang mewarisi.

Contoh program *access modifiers*:

```
<?php
class Mahasiswa {
    public $nama;           // dapat diakses dari
mana saja
    private $nim;           // hanya bisa diakses
dari dalam kelas
    protected $jurusan;     // bisa diakses dari
kelas ini & kelas turunan

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

    // Public method
    public function tampilkanData() {
```

```
        echo "Nama: " . $this->nama . "<br>";
        echo "NIM: " . $this->nim . "<br>";
       echo "Jurusan: " . $this->jurusan . "<br>";
    }

     // Setter untuk mengubah NIM (karena NIM
private)
    public function setNim($nimBaru) {
        $this->nim = $nimBaru;
    }
}

// Kelas turunan
class MahasiswaAktif extends Mahasiswa {
    public function tampilkanJurusan() {
       echo "Mahasiswa jurusan: " . $this->jurusan
. "<br>"; // bisa akses protected
    }
}

// Membuat objek
$mhs1 = new Mahasiswa("Mizard Farid", "220001",
"Teknik Informatika");
$mhs1->tampilkanData();

// Mengubah NIM menggunakan setter
$mhs1->setNim("220099");
$mhs1->tampilkanData();

// Membuat objek dari kelas turunan
$mhs2 = new MahasiswaAktif("Dzimar Farid",
"220002", "Sistem Informasi");
$mhs2->tampilkanJurusan();

// Akses langsung (contoh error):
// echo $mhs1->nim;      // ERROR (karena private)
// echo $mhs1->jurusan;     // ERROR (karena
protected)
?>
```

Output-nya:

```
Nama: Mizard Farid
NIM: 220001
Jurusan: Teknik Informatika
Nama: Mizard Farid
NIM: 220099
Jurusan: Teknik Informatika
Mahasiswa jurusan: Sistem Informasi
```

Penjelasan:

1. `Public ($nama)`, yaitu bisa diakses langsung dari luar kelas.
2. `Private ($nim)`, yaitu tidak bisa diakses langsung, harus melalui *method* khusus (*getter*/*setter*).
3. `Protected ($jurusan)`, yaitu tidak bisa diakses dari luar, tapi bisa diakses dari kelas turunan (MahasiswaAktif).

C. Getter dan Setter dalam PHP

Seperti yang sudah dibahas sebelumnya, ketika properti dalam sebuah kelas diberi akses *private* atau *protected*, maka properti tersebut tidak bisa diakses langsung dari luar kelas. Untuk mengatasinya, PHP menyediakan cara tidak langsung yaitu melalui *getter* dan *setter*.

Getter adalah *method* yang digunakan untuk mengambil (get) nilai dari properti. Sedangkan, *Setter* adalah *method* yang digunakan untuk mengatur (set) atau mengubah nilai dari properti.

Fungsi *getter* dan *setter*, yaitu mengontrol akses ke data agar tetap aman, menambahkan validasi sebelum data disimpan, dan mencegah manipulasi langsung terhadap data sensitif.

Contoh program *getter* dan *setter*:

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

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

```
    // Getter untuk nama
    public function getNama() {
        return $this->nama;
    }

    // Setter untuk nama
    public function setNama($namaBaru) {
          // Tambahkan validasi agar nama tidak
kosong
        if (!empty($namaBaru)) {
            $this->nama = $namaBaru;
        } else {
            echo "Nama tidak boleh kosong!<br>";
        }
    }

    // Getter untuk NIM
    public function getNim() {
        return $this->nim;
    }

    // Setter untuk NIM
    public function setNim($nimBaru) {
        // Validasi: NIM harus angka 6 digit
                if    (is_numeric($nimBaru)    &&
strlen($nimBaru) == 6) {
            $this->nim = $nimBaru;
        } else {
            echo "Format NIM tidak valid!<br>";
        }
    }
}

// Membuat objek
$mhs1 = new Mahasiswa("Mizard Farid", "220001");

// Mengambil nilai dengan getter
echo "Nama: " . $mhs1->getNama() . "<br>";
echo "NIM: " . $mhs1->getNim() . "<br>";
```

```
// Mengubah nilai dengan setter
$mhs1->setNama("Dzimar Farid");
$mhs1->setNim("220099");

// Menampilkan hasil setelah perubahan
echo "Nama Baru: " . $mhs1->getNama() . "<br>";
echo "NIM Baru: " . $mhs1->getNim() . "<br>";

// Contoh validasi gagal
$mhs1->setNama("");          // akan menampilkan
pesan error
$mhs1->setNim("abc123");     // akan menampilkan
pesan error
?>
```

Output-nya:

```
Nama: Mizard Farid
NIM: 220001
Nama Baru: Dzimar Farid
NIM Baru: 220099
Nama tidak boleh kosong!
Format NIM tidak valid!
```

Penjelasan kode program di atas:

1. Properti `$nama` dan `$nim` dibuat *private*, sehingga tidak bisa diakses langsung.
2. Getter (`getNama(), getNim()`) digunakan untuk membaca nilai.
3. Setter (`setNama(), setNim()`) digunakan untuk mengubah nilai dengan validasi.
   - Nama tidak boleh kosong.
   - NIM harus angka dan terdiri dari 6 digit.
4. Dengan cara ini, data mahasiswa lebih terkontrol dan aman.

D. Studi Kasus: Implementasi Enkapsulasi dalam PHP

Untuk memahami penerapan enkapsulasi dalam PHP dengan menggunakan studi kasus sederhana tentang kelas Mahasiswa. Dalam kasus ini, membuat properti nama, NIM, jurusan, dan IPK yang semuanya bersifat *private*. Akses ke properti ini hanya bisa

dilakukan melalui *getter* dan *setter* dengan validasi tertentu. Berikut kode programnya:

```
<?php
class Mahasiswa {
    // Properti private
    private $nama;
    private $nim;
    private $jurusan;
    private $ipk;

    // Konstruktor
     public function __construct($nama, $nim,
$jurusan, $ipk) {
        $this->setNama($nama);
        $this->setNim($nim);
        $this->setJurusan($jurusan);
        $this->setIpk($ipk);
    }

    // Getter dan Setter Nama
    public function getNama() {
        return $this->nama;
    }
    public function setNama($nama) {
        if (!empty($nama)) {
            $this->nama = $nama;
        } else {
            echo "Nama tidak boleh kosong!<br>";
        }
    }

    // Getter dan Setter NIM
    public function getNim() {
        return $this->nim;
    }
    public function setNim($nim) {
        if (is_numeric($nim) && strlen($nim) == 6)
{
            $this->nim = $nim;
        } else {
```

```
                echo "NIM harus berupa angka 6
digit!<br>";
        }
    }

    // Getter dan Setter Jurusan
    public function getJurusan() {
        return $this->jurusan;
    }
    public function setJurusan($jurusan) {
        $this->jurusan = $jurusan;
    }

    // Getter dan Setter IPK
    public function getIpk() {
        return $this->ipk;
    }
    public function setIpk($ipk) {
        if (is_numeric($ipk) && $ipk >= 0 && $ipk
<= 4) {
            $this->ipk = $ipk;
        } else {
            echo "IPK harus berupa angka antara 0
- 4!<br>";
        }
    }

    // Method untuk menampilkan data
    public function tampilkanData() {
        echo "Nama: " . $this->getNama() . "<br>";
        echo "NIM: " . $this->getNim() . "<br>";
         echo "Jurusan: " . $this->getJurusan() .
"<br>";
            echo "IPK: " . $this->getIpk() .
"<br><br>";
    }
}

// Membuat objek Mahasiswa
$mhs1 = new Mahasiswa("Mizard Farid", "220001",
"Teknik Informatika", 3.75);
```

```
$mhs2 = new Mahasiswa("Dzimar Farid", "220002",
"Sistem Informasi", 3.90);

// Menampilkan data mahasiswa
$mhs1->tampilkanData();
$mhs2->tampilkanData();

// Mengubah data dengan setter
$mhs1->setNama("Mizard F.");
$mhs1->setIpk(4.2); // Akan gagal karena IPK > 4

$mhs1->tampilkanData();
?>
```

Ouput-nya:

```
Nama: Mizard Farid
NIM: 220001
Jurusan: Teknik Informatika
IPK: 3.75

Nama: Dzimar Farid
NIM: 220002
Jurusan: Sistem Informasi
IPK: 3.9

IPK harus berupa angka antara 0 - 4!
Nama: Mizard F.
NIM: 220001
Jurusan: Teknik Informatika
IPK: 3.75
```

Penjelasan:

1. Semua properti (`nama`, `nim`, `jurusan`, `ipk`) dibuat *private* → tidak bisa diakses langsung dari luar kelas.
2. *Getter* dan *Setter* digunakan untuk membaca dan mengubah data dengan validasi.
   - Nama tidak boleh kosong.
   - NIM harus berupa angka 6 digit.
   - IPK harus berada di antara 0–4.
3. *Method* `tampilkanData()` memanfaatkan *getter* agar data lebih aman saat ditampilkan.

RANGKUMAN

1. Enkapsulasi adalah salah satu prinsip utama dalam OOP yang berfungsi untuk menyembunyikan detail implementasi sebuah kelas agar tidak bisa diakses sembarangan dari luar kelas. Tujuan utamanya adalah menjaga keamanan data, mencegah perubahan yang tidak terkontrol, serta membuat kode lebih rapi dan mudah dipelihara.
2. Dalam PHP, konsep enkapsulasi diatur melalui *access modifiers*:
   - *public* → dapat diakses dari mana saja (di dalam maupun luar kelas).
   - *private* → hanya dapat diakses dari dalam kelas yang sama.
   - *protected* → dapat diakses dari dalam kelas yang sama dan kelas turunannya (*inheritance*).
3. Untuk mengakses atau memodifikasi properti yang bersifat *private* atau *protected*, digunakan *method* khusus berupa *getter* (mengambil nilai) dan *setter* (mengubah nilai). Dengan *getter* dan *setter*, kita dapat menambahkan validasi sebelum data disimpan, sehingga data lebih konsisten dan terjamin.
4. Enkapsulasi memberikan beberapa manfaat penting, di antaranya:
   - Keamanan data → mencegah akses langsung terhadap atribut sensitif.
   - Kontrol penuh → memudahkan penambahan validasi sebelum data diubah.
   - Pemeliharaan kode → membuat kode lebih modular dan mudah dikembangkan.
   - Fleksibilitas → memungkinkan perubahan internal tanpa mengubah cara penggunaannya.
5. Studi kasus kelas Mahasiswa menunjukkan bahwa enkapsulasi mampu membatasi akses data mahasiswa sekaligus memastikan data yang disimpan valid (misalnya NIM harus 6 digit, IPK hanya 0–4).

LATIHAN

1. Latihan Teori

1. Jelaskan dengan kata-kata sendiri apa yang dimaksud dengan enkapsulasi dalam OOP.
2. Sebutkan dan jelaskan tiga jenis *access modifiers* dalam PHP beserta contohnya.
3. Mengapa penggunaan *getter* dan *setter* dianggap lebih aman dibandingkan mengakses properti secara langsung?
4. Apa perbedaan utama antara *private* dan *protected* dalam PHP?
5. Sebutkan tiga manfaat utama penerapan enkapsulasi dalam pemrograman berorientasi objek.

2. Latihan Praktik

1. Buat sebuah kelas `AkunBank` dengan properti berikut:
   - `noRekening` (private)
   - `namaPemilik` (private)
   - `saldo` (private)

   Lengkapi dengan *getter* dan *setter* yang sesuai, serta *method* `deposit($jumlah)` dan `tarik($jumlah)`.
   - Validasi: saldo tidak boleh negatif, penarikan tidak boleh melebihi saldo.
2. Buat sebuah kelas Produk dengan properti `nama, harga`, dan `stok` (semuanya *private*).
   - Tambahkan *getter* dan *setter* dengan validasi: harga tidak boleh negatif, stok tidak boleh kurang dari 0.
   - Tambahkan *method* `tampilkanInfo()` untuk menampilkan data produk.
3. Modifikasi kelas Mahasiswa yang sudah dipelajari sebelumnya:
   - Tambahkan properti `email` (private).
   - Buat getter dan setter dengan validasi bahwa email harus mengandung tanda "@".

- Buat method `updateJurusan($jurusanBaru)` untuk mengubah jurusan mahasiswa.

3. Tugas Proyek Mini

Buat program sederhana dengan PHP menggunakan konsep enkapsulasi untuk sistem Perpustakaan.

- Buat kelas `Buku` dengan properti: `judul`, `penulis`, `tahunTerbit`, `status` (tersedia/dipinjam). Semua properti bersifat *private*.
- Tambahkan *getter* dan *setter* dengan validasi yang sesuai.
- Tambahkan *method* `pinjamBuku()` dan `kembalikanBuku()`. Jika buku sedang dipinjam, maka tidak boleh dipinjam lagi.
- Buat beberapa objek `Buku`, lalu tampilkan daftar buku beserta statusnya.

BAB 5
Pewarisan (Inheritance) dalam PHP

DESKRIPSI:

Bab ini membahas konsep pewarisan (*inheritance*) dalam pemrograman berorientasi objek menggunakan PHP. *Inheritance* adalah mekanisme yang memungkinkan sebuah kelas (*child class* / *subclass*) untuk mewarisi properti dan *method* dari kelas lain (*parent class* / *superclass*). Dengan adanya pewarisan, kita dapat menghindari duplikasi kode, membuat program lebih modular, efisien, dan mudah dikembangkan. Selain itu, *inheritance* juga menjadi dasar dari konsep polimorfisme, yang memungkinkan sebuah *method* memiliki perilaku berbeda pada kelas turunan.

TUJUAN PEMBELAJARAN:

Setelah mempelajari bab ini, mahasiswa diharapkan mampu:

1. Menjelaskan konsep *inheritance* dalam OOP.
2. Mengidentifikasi perbedaan antara kelas induk (*parent class*) dan kelas turunan (*child class*).
3. Membuat program PHP yang menerapkan *inheritance* untuk menghindari duplikasi kode.
4. Memahami penggunaan *keyword extends* dalam pewarisan PHP.
5. Menggunakan *inheritance* untuk mengembangkan aplikasi sederhana berbasis OOP dengan lebih efisien.

A. Pengertian Pewarisan

Inheritance (pewarisan) adalah salah satu pilar penting dalam *Object-Oriented Programming* (OOP). *Inheritance* memungkinkan sebuah kelas turunan (*child class* / *subclass*) untuk mewarisi properti

dan *method* dari kelas induk (*parent class* / *superclass*). Dengan *inheritance* dapat menghindari duplikasi kode, yaitu kode yang umum diletakkan di *parent class*, lalu digunakan ulang oleh *child class*. Dapat membuat kode lebih terstruktur dan modular yaitu perubahan cukup dilakukan di *parent class* tanpa perlu mengubah semua kelas turunan. Dapat mempermudah pengembangan program, yaitu *child class* dapat menggunakan sekaligus menambah atau memodifikasi perilaku *parent class*. Studi kasus dalam analogi sehari-hari bayangkan ada kelas induk "Hewan" yang memiliki sifat umum: bernapas, makan, bergerak. Dari kelas induk tersebut, bisa membuat kelas turunan "Kucing" dan "Burung". Kucing mewarisi kemampuan makan dan bernapas, tetapi juga memiliki perilaku khusus seperti mengeong. Burung juga mewarisi kemampuan makan dan bernapas, tetapi memiliki perilaku khusus seperti terbang.

Contoh kode PHP: *inheritance*

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

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

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

    public function bergerak() {
       echo $this->nama . " sedang bergerak.<br>";
    }
}

// Child class
class Kucing extends Hewan {
    public function suara() {
            echo $this->nama . " mengeong:
Meong...<br>";
```

```
    }
}

// Child class lain
class Burung extends Hewan {
    public function suara() {
        echo $this->nama . " berkicau: Cuit...
Cuit...<br>";
    }
}

// Membuat objek
$kucing = new Kucing("Garfield");
$burung = new Burung("Birdie");

// Memanggil method parent
$kucing->makan();
$burung->bergerak();

// Memanggil method child
$kucing->suara();
$burung->suara();
?>
```

Output:

```
Garfield sedang makan.
Birdie sedang bergerak.
Garfield mengeong: Meong...
Birdie berkicau: Cuit... Cuit...
```

Penjelasan kode di atas:

1. `Kucing` dan `Burung` adalah *child class* yang menggunakan *keyword* `extends` untuk mewarisi properti dan *method* dari `Hewan`.
2. Kedua kelas tersebut secara otomatis memiliki *method* `makan()` dan `bergerak()` dari `Hewan`.
3. Masing-masing *child class* juga dapat menambahkan perilaku baru (`suara()`).

B. Membuat Parent Class dan Child Class

Dalam PHP, *parent class* adalah kelas utama yang berisi properti dan *method* umum, sedangkan *child class* adalah kelas turunan yang mewarisi semua properti dan *method* dari *parent class*, serta dapat menambahkan fitur khusus. Untuk membuat pewarisan di PHP digunakan *keyword extends*. Studi kasus membuat kelas induk `Mahasiswa` yang memiliki properti dasar seperti `nama, nim`, dan `jurusan`. Lalu kita buat kelas turunan `MahasiswaBaru` yang mewarisi semua properti dan *method* dari `Mahasiswa`, tetapi juga memiliki tambahan fitur khusus seperti `registrasiMataKuliah()`.

Contoh kode program sebagai berikut:

```
<?php
// Parent class
class Mahasiswa {
    protected $nama;
    protected $nim;
    protected $jurusan;

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

    public function tampilkanData() {
        echo "Nama     : $this->nama <br>";
        echo "NIM      : $this->nim <br>";
        echo "Jurusan  : $this->jurusan <br><br>";
    }
}

// Child class
class MahasiswaBaru extends Mahasiswa {
    private $tahunMasuk;

     public function __construct($nama, $nim,
$jurusan, $tahunMasuk) {
```

```
        // memanggil constructor parent class
                parent::__construct($nama,   $nim,
$jurusan);
        $this->tahunMasuk = $tahunMasuk;
    }

    public function registrasiMataKuliah() {
          echo  $this->nama  .  "  telah  melakukan
registrasi mata kuliah.<br>";
    }

    public function tampilkanData() {
        // override method dari parent
        parent::tampilkanData();
           echo  "Tahun  Masuk:  $this->tahunMasuk
<br><br>";
    }
}

// Membuat objek dari parent class
$mhs1 = new Mahasiswa("Ghifar", "220101", "Teknik
Informatika");
$mhs1->tampilkanData();

// Membuat objek dari child class
$mhs2  =  new  MahasiswaBaru("Dzimar",  "220102",
"Sistem Informasi", 2025);
$mhs2->tampilkanData();
$mhs2->registrasiMataKuliah();
?>
```

Output:

```
Nama : Ghifar
NIM : 220101
Jurusan : Teknik Informatika

Nama : Dzimar
NIM : 220102
Jurusan : Sistem Informasi

Tahun Masuk: 2025
Dzimar telah melakukan registrasi mata kuliah.
```

Penjelasan:

1. *Parent Class* (`Mahasiswa`) → berisi properti dan *method* umum (`nama, nim, jurusan, tampilkanData`).
2. *Child Class* (`MahasiswaBaru`) → mewarisi semua properti dan *method* dari `Mahasiswa`, lalu menambahkan properti baru (`tahunMasuk`) dan *method* baru (`registrasiMataKuliah`).
3. `parent::__construct()` digunakan agar konstruktor *parent class* tetap dijalankan ketika *child class* dibuat.
4. *Overriding* dilakukan pada *method* `tampilkanData()` sehingga versi di *child class* bisa menambahkan informasi tambahan.

C. Keyword Extends dalam PHP

Dalam PHP, pewarisan (*inheritance*) direalisasikan menggunakan *keyword extends*. Dengan *extends*, sebuah *child class* dapat mewarisi semua properti dan *method* dari *parent class*. Aturan dasarnya yaitu sebagai berikut:

1. Satu arah pewarisan
   Sebuah *class* hanya dapat mewarisi dari satu *parent class* (tidak ada *multiple inheritance* di PHP).Namun, sebuah *parent class* bisa diturunkan ke banyak *child class*.
2. Aksesibilitas
   Properti/*method* dengan *modifier public* dan *protected* dapat diwarisi. Properti/*method* dengan *modifier private* tidak diwarisi langsung, tetapi tetap bisa diakses lewat *getter*/*setter* di *parent class*.
3. Override Method
   Child class dapat mengubah (*override*) *method* yang diwarisi dari *parent class* untuk menyesuaikan kebutuhan.

Contoh kode penggunaan *extends*:

```
<?php
// Parent class
class Kendaraan {
    public $merk;
    public $tahun;
```

```
    public function __construct($merk, $tahun) {
        $this->merk = $merk;
        $this->tahun = $tahun;
    }

    public function infoKendaraan() {
         echo "Merk: $this->merk, Tahun: $this-
>tahun <br>";
    }
}

// Child class
class Mobil extends Kendaraan {
    public $jenis;

     public function __construct($merk, $tahun,
$jenis) {
        // memanggil constructor parent
        parent::__construct($merk, $tahun);
        $this->jenis = $jenis;
    }

    // Override method
    public function infoKendaraan() {
        parent::infoKendaraan(); // panggil versi
parent
        echo "Jenis: $this->jenis <br>";
    }
}

// Objek
$mobil1 = new Mobil("Toyota", 2020, "SUV");
$mobil1->infoKendaraan();
?>
```

Output:

```
Merk: Toyota, Tahun: 2020
Jenis: SUV
```

Penjelasan:

1. `class Mobil extends Kendaraan` → menunjukkan bahwa `Mobil` adalah *child class* dari `Kendaraan`.

2. `Mobil` otomatis mewarisi properti `$merk` dan `$tahun`, serta *method* `infoKendaraan()`.
3. `parent::__construct()` → dipakai agar *constructor* dari *parent class* tetap dijalankan.
4. *Method overriding* → *method* `infoKendaraan()` di *child class* menambahkan informasi baru (`jenis`).

D. Overriding Method dalam Inheritance

Overriding method adalah proses ketika *child class* mendefinisikan kembali (menimpa) *method* yang sudah ada di *parent class*. Dengan *overriding*, *method* yang diwarisi bisa dimodifikasi agar memiliki perilaku yang berbeda sesuai kebutuhan *child class*. Aturan *overriding* di PHP, yaitu:

1. Nama *method* di *child class* harus sama persis dengan nama *method* di *parent class*.
2. *Method* di *parent class* bisa tetap dipanggil menggunakan `parent::namaMethod()`.
3. Modifier akses (`public, protected, private`) pada *method* yang di *override* harus sama atau lebih longgar (misalnya parent `protected`, child boleh `public`).

Contoh kode program *overriding*:

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

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

    public function bersuara() {
        return "Hewan ini mengeluarkan suara
umum.";
    }
}

// Child class
class Kucing extends Hewan {
```

```
    // Overriding method bersuara()
    public function bersuara() {
        return "Meong...!";
    }
}

// Child class lain
class Anjing extends Hewan {
    // Overriding method bersuara()
    public function bersuara() {
        return "Guk... Guk...!";
    }
}

// Objek
$hewan1 = new Hewan("Hewan");
echo $hewan1->bersuara() . "<br>";

$kucing1 = new Kucing("Kitty");
echo $kucing1->bersuara() . "<br>";

$anjing1 = new Anjing("Doggy");
echo $anjing1->bersuara() . "<br>";
?>
```

Output:

```
Hewan ini mengeluarkan suara umum.
Meong...!
Guk... Guk...!
```

Penjelasan:

1. Kelas `Hewan` memiliki *method* `bersuara()` bawaan.
2. Kelas `Kucing` dan `Anjing` melakukan *overriding* pada *method* `bersuara()`.

- Pada `Kucing`, `bersuara()` mengembalikan "Meong...!".
- Pada `Anjing`, `bersuara()` mengembalikan "Guk... Guk...!".

3. Meskipun *method* yang dipanggil sama (`bersuara()`), hasilnya berbeda tergantung objeknya.

E. Studi Kasus: Implementasi Inheritance dalam PHP

Untuk memahami konsep pewarisan (*inheritance*) secara lebih nyata, kita akan membuat studi kasus sederhana terkait data mahasiswa. Dalam contoh ini, akan ada sebuah kelas induk (*Parent Class*) bernama `Mahasiswa`, lalu dibuat beberapa kelas turunan (*Child Class*), yaitu `MahasiswaS1` dan `MahasiswaS2`. Berikut kode programnya:

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

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

    // Method umum
    public function tampilkanData() {
        echo "Nama: {$this->nama} <br>";
        echo "NIM: {$this->nim} <br>";
        echo "Jurusan: {$this->jurusan} <br>";
    }
}

// Child class untuk Mahasiswa S1
class MahasiswaS1 extends Mahasiswa {
    public $skripsi;

     public function __construct($nama, $nim,
$jurusan, $skripsi) {
        // Memanggil constructor parent class
               parent::__construct($nama, $nim,
$jurusan);
        $this->skripsi = $skripsi;
    }
```

```
    // Overriding method tampilkanData()
    public function tampilkanData() {
           parent::tampilkanData();  //  Memanggil
method parent
          echo  "Judul  Skripsi:  {$this->skripsi}
<br>";
    }
}

// Child class untuk Mahasiswa S2
class MahasiswaS2 extends Mahasiswa {
    public $tesis;

     public  function  __construct($nama,  $nim,
$jurusan, $tesis) {
               parent::__construct($nama,   $nim,
$jurusan);
        $this->tesis = $tesis;
    }

    // Overriding method tampilkanData()
    public function tampilkanData() {
        parent::tampilkanData();
        echo "Judul Tesis: {$this->tesis} <br>";
    }
}

// --- Implementasi ---

// Mahasiswa S1
$mhs1   =   new   MahasiswaS1("Dzimar   Rauhillah",
"20231001",    "Sistem    Informasi",    "Sistem
Rekomendasi Jalur Belajar");
$mhs1->tampilkanData();
echo "<hr>";

// Mahasiswa S2
$mhs2    =    new    MahasiswaS2("Ghifar    Farid",
"20252002", "Ilmu Komputer", "Model Hybrid HMM-BN
untuk Adaptive Learning");
```

```
$mhs2->tampilkanData();
?>
```

Output:

```
Nama: Dzimar Rauhillah
NIM: 20231001
Jurusan: Sistem Informasi
Judul Skripsi: Sistem Rekomendasi Jalur Belajar

Nama: Ghifar Farid
NIM: 20252002
Jurusan: Ilmu Komputer
Judul Tesis: Model Hybrid HMM-BN untuk Adaptive
Learning
```

Penjelasan:

1. Kelas Induk Mahasiswa
   - Menyimpan data umum mahasiswa (nama, NIM, jurusan).
   - Memiliki *method* tampilkanData() untuk menampilkan informasi dasar mahasiswa.
2. Kelas MahasiswaS1
   - Mewarisi semua properti dan *method* dari Mahasiswa.
   - Menambahkan properti baru yaitu $skripsi.
   - Melakukan *overriding* pada *method* tampilkanData() agar bisa menampilkan judul skripsi.
3. Kelas MahasiswaS2
   - Sama seperti S1, tetapi memiliki properti $tesis untuk menyimpan judul tesis.
   - Juga melakukan *overriding* pada *method* tampilkanData().

Studi kasus ini menunjukkan bahwa dengan *inheritance* Kode lebih efisien karena tidak perlu menulis ulang atribut umum (nama, NIM, jurusan). Memungkinkan adanya spesialisasi sesuai kebutuhan (Mahasiswa S-1 punya skripsi, Mahasiswa S-2 punya tesis). Lebih mudah dikembangkan, misalnya menambahkan Mahasiswa S-3 dengan disertasi tanpa mengubah kelas Mahasiswa.

RANGKUMAN

1. *Inheritance* (pewarisan) adalah konsep OOP yang memungkinkan sebuah kelas (*child class*) mewarisi properti dan *method* dari kelas lain (*parent class*). Dengan *inheritance*, kode dapat digunakan kembali (*code reusability*) sehingga lebih efisien dan terstruktur.
2. *Keyword extends* digunakan dalam PHP untuk menunjukkan bahwa sebuah kelas merupakan turunan dari kelas lain.
3. *Child class* dapat:
   - Menggunakan semua properti dan *method* dari *parent class*.
   - Menambahkan properti dan *method* baru sesuai kebutuhan.
   - Melakukan *overriding method*, yaitu mendefinisikan ulang *method parent class* agar memiliki perilaku yang berbeda.
4. Aturan *overriding*:
   - Nama *method* di *child class* harus sama dengan di *parent class*.
   - Modifier akses harus sama atau lebih longgar.
   - *Parent method* tetap bisa dipanggil menggunakan *parent*::methodName().
5. Manfaat *inheritance* dalam PHP:
   - Mengurangi duplikasi kode dengan memanfaatkan kembali kode dari *parent class*.
   - Mempermudah pengembangan sistem karena *class* turunan bisa disesuaikan dengan kebutuhan spesifik.
   - Mendukung konsep polimorfisme, di mana satu *method* dapat berperilaku berbeda pada *class* turunan yang berbeda.
6. Studi kasus Mahasiswa S-1 dan S-2 menunjukkan bagaimana *inheritance* dapat digunakan untuk membedakan atribut dan perilaku tiap kelas, meskipun memiliki struktur umum yang sama.

LATIHAN

1. Pertanyaan Teori

1. Jelaskan dengan kata-kata Anda sendiri apa yang dimaksud dengan *inheritance* dalam OOP PHP.
2. Apa perbedaan antara *inheritance* dengan *composition* dalam pemrograman berorientasi objek?
3. Sebutkan tiga manfaat utama penggunaan *inheritance* dalam PHP.
4. Apa yang dimaksud dengan *overriding method*? Berikan contohnya.
5. Mengapa penggunaan *parent*::methodName() sering dibutuhkan ketika melakukan *overriding*?

2. Latihan Praktik

1. Buatlah sebuah *class* `Kendaraan` dengan properti:
   - `Merek`
   - `Tahun`
   - serta *method* `infoKendaraan()` untuk menampilkan informasi kendaraan.
2. Buatlah *class* turunan `Mobil` yang mewarisi dari `Kendaraan`, dengan tambahan properti:
   - `jumlahPintu`
   - serta *method* `infoKendaraan()` yang menampilkan informasi lengkap termasuk jumlah pintu.
3. Buatlah *class* turunan `Motor` dengan tambahan properti:
   - `jenisMotor` (misalnya `matic, sport, bebek`)
   - serta *method* `infoKendaraan()` yang menampilkan informasi lengkap termasuk jenis motor.
4. Buatlah objek dari kedua *class* turunan (`Mobil` dan `Motor`) lalu tampilkan informasinya.

BAB 6
Polimorfisme dalam PHP

DESKRIPSI:

Bab ini membahas tentang konsep polimorfisme dalam pemrograman berorientasi objek (OOP) menggunakan PHP. Polimorfisme adalah kemampuan sebuah objek untuk memiliki banyak bentuk, khususnya ketika sebuah *method* yang sama dapat digunakan dengan cara yang berbeda pada *class* yang berbeda. Dengan polimorfisme, kode menjadi lebih fleksibel, dapat digunakan kembali, dan mendukung pengembangan sistem yang lebih modular. Pemahaman tentang polimorfisme sangat penting agar mahasiswa mampu merancang sistem dengan struktur yang rapi, mendukung perubahan, serta meminimalisir duplikasi kode.

TUJUAN PEMBELAJARAN:

Setelah mempelajari bab ini, mahasiswa diharapkan mampu:

1. Menjelaskan konsep polimorfisme dalam PHP.
2. Membedakan antara polimorfisme melalui *inheritance*, *abstract class*, dan *interface*.
3. Mengimplementasikan polimorfisme dalam bentuk *method overriding*.
4. Membuat *class* dengan *abstract method* dan mengimplementasikannya pada *class* turunan.
5. Menggunakan *interface* untuk menciptakan kontrak *method* yang harus diimplementasikan oleh *class* lain.
6. Menerapkan polimorfisme dalam studi kasus nyata berbasis PHP.

A. Pengertian Polimorfisme

Secara bahasa, polimorfisme berasal dari bahasa Yunani, yaitu *poly* (banyak) dan *morph* (bentuk). Dalam OOP, polimorfisme berarti

kemampuan suatu objek untuk memiliki banyak bentuk atau perilaku, meskipun memiliki nama *method* yang sama. Dengan polimorfisme, kita dapat menggunakan satu *interface* (atau kontrak) yang sama, tetapi implementasinya berbeda pada masing-masing *class*. Hal ini membuat kode lebih fleksibel, mudah dikembangkan, dan efisien. Berikut jenis polimorfisme:

1. Polimorfisme melalui Inheritance (Overriding)
   Method di *parent class* diturunkan ke *child class*, tetapi di *override* (diganti isinya) agar sesuai kebutuhan.
2. Polimorfisme melalui Abstract Class
   Class abstrak mendefinisikan kontrak *method*, lalu *class* turunannya wajib mengimplementasikan *method* tersebut.
3. Polimorfisme melalui Interface
   Interface mendefinisikan *method* tanpa implementasi, lalu *class* yang menggunakannya harus mengimplementasikan semua *method* tersebut.

Contoh polimorfisme dengan *inheritance* (*overriding*):

```
<?php
// Parent class
class Hewan {
    public function bersuara() {
        return "Hewan ini bersuara...";
    }
}

// Child class
class Kucing extends Hewan {
    public function bersuara() {
        return "Meong...!";
    }
}

// Child class lain
class Anjing extends Hewan {
    public function bersuara() {
        return "Guk... Guk...!";
    }
}
```

```
// Fungsi yang menerima parameter Hewan
function cetakSuara(Hewan $hewan) {
    echo $hewan->bersuara() . "<br>";
}

// Uji coba
cetakSuara(new Hewan());
cetakSuara(new Kucing());
cetakSuara(new Anjing());
?>
```

Output:

```
Hewan ini bersuara...
Meong...!
Guk... Guk...!
```

Dalam contoh di atas, setiap *class* (`Hewan`, `Kucing`, dan `Anjing`) memiliki *method* dengan nama yang sama yaitu `bersuara()`, namun masing-masing memberikan hasil yang berbeda sesuai dengan objeknya. Hal ini menunjukkan bahwa meskipun pemanggilan *method* sama, perilakunya bisa berbeda tergantung pada implementasi di *class* turunan. Inilah esensi dari polimorfisme dalam OOP: kemampuan satu *method* untuk memiliki banyak bentuk perilaku, sehingga kode menjadi lebih fleksibel, generik, dan mudah dikembangkan.

B. Polimorfisme dengan Abstract Class

Selain melalui *inheritance* (*overriding*), polimorfisme juga dapat dicapai dengan menggunakan *abstract class*. *Abstract class* adalah kelas yang tidak dapat dibuat objek secara langsung, melainkan harus diturunkan ke kelas lain. Di dalam *abstract class*, biasanya terdapat *abstract method*, yaitu *method* yang hanya dideklarasikan tanpa implementasi. Setiap *class* turunan wajib memberikan implementasi konkret untuk *abstract method* tersebut. Dengan cara ini, kita dapat memastikan bahwa setiap *class* turunan memiliki *method* dengan nama yang sama, tetapi dengan perilaku yang berbeda sesuai kebutuhan masing-masing *class* → inilah bentuk lain dari polimorfisme.

Contoh kode program:

```
<?php
// Abstract class
abstract class Mahasiswa {
    protected $nama;
    protected $nim;

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

    // Getter agar properti protected bisa dibaca
dari luar
    public function getNama() {
        return $this->nama;
    }

    public function getNim() {
        return $this->nim;
    }

    // Abstract method (harus diimplementasikan
oleh child class)
    abstract public function tugasAkhir();
}

// Child class S1
class MahasiswaS1 extends Mahasiswa {
    public function tugasAkhir() {
        return "Skripsi";
    }
}

// Child class S2
class MahasiswaS2 extends Mahasiswa {
    public function tugasAkhir() {
        return "Tesis";
    }
}
```

```
// Child class S3
class MahasiswaS3 extends Mahasiswa {
    public function tugasAkhir() {
        return "Disertasi";
    }
}

// Fungsi untuk menampilkan informasi — gunakan
getter, bukan akses properti langsung
function infoMahasiswa(Mahasiswa $mhs) {
    echo "Nama: " . $mhs->getNama() .
         ", NIM: " . $mhs->getNim() .
         ", Tugas Akhir: " . $mhs->tugasAkhir() .
"<br>";
}

// Uji coba
$mhs1 = new MahasiswaS1("Mizard", "20231001");
$mhs2 = new MahasiswaS2("Dzimar", "20242002");
$mhs3 = new MahasiswaS3("Ghifar", "20253003");

infoMahasiswa($mhs1);
infoMahasiswa($mhs2);
infoMahasiswa($mhs3);
?>
```

Output:

```
Nama: Mizard, NIM: 20231001, Tugas Akhir: Skripsi
Nama: Dzimar, NIM: 20242002, Tugas Akhir: Tesis
Nama: Ghifar, NIM: 20253003, Tugas Akhir: Disertasi
```

Pada contoh di atas, *class* `Mahasiswa` didefinisikan sebagai *abstract class* yang memiliki *abstract method* `tugasAkhir()`. Method ini tidak memiliki implementasi di *class* induk, tetapi wajib diimplementasikan pada setiap *class* turunan (`MahasiswaS1`, `MahasiswaS2`, `MahasiswaS3`). Meskipun nama *method* yang dipanggil sama (`tugasAkhir()`), hasil yang ditampilkan berbeda sesuai *class* turunannya: mahasiswa S-1 memiliki skripsi, mahasiswa S-2 memiliki tesis, dan mahasiswa S-3 memiliki disertasi. Inilah yang menunjukkan bagaimana

polimorfisme bekerja melalui *abstract class*: satu kontrak *method* dengan banyak bentuk perilaku.

C. Polimorfisme dengan Interface

Selain menggunakan *abstract class*, polimorfisme juga dapat diterapkan melalui *interface*. *Interface* dalam PHP berfungsi sebagai kontrak yang harus dipatuhi oleh setiap *class* yang mengimplementasikannya. Semua metode yang dideklarasikan dalam *interface* wajib diimplementasikan oleh *class* yang menggunakannya. Dengan cara ini, berbagai *class* yang berbeda dapat memiliki perilaku yang seragam meskipun implementasinya berbeda.

Contoh kode implementasi polimorfisme dengan *interface* pada data mahasiswa:

```
<?php
// Mendefinisikan Interface
interface MahasiswaInterface {
    public function getNama();
    public function getNim();
    public function getJenjang();
}

// Class Mahasiswa S1
class MahasiswaS1 implements MahasiswaInterface {
    private $nama;
    private $nim;

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

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

    public function getNim() {
        return $this->nim;
    }
```

```
    public function getJenjang() {
        return "Sarjana (S1)";
    }
}

// Class Mahasiswa S2
class MahasiswaS2 implements MahasiswaInterface {
    private $nama;
    private $nim;

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

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

    public function getNim() {
        return $this->nim;
    }

    public function getJenjang() {
        return "Magister (S2)";
    }
}

// Fungsi polimorfisme
function infoMahasiswa(MahasiswaInterface $mhs) {
    echo "Nama: " . $mhs->getNama() . "<br>";
    echo "NIM: " . $mhs->getNim() . "<br>";
     echo "Jenjang: " . $mhs->getJenjang() .
"<br><br>";
}

// Implementasi
$mhs1 = new MahasiswaS1("Dzimar Rauhillah",
"20231001");
```

```
$mhs2 = new MahasiswaS2("Ghifar Dzikrullah",
"20252001");

// Pemanggilan fungsi yang sama untuk objek berbeda
infoMahasiswa($mhs1);
infoMahasiswa($mhs2);
?>
```

Output:

```
Nama: Dzimar Rauhillah
NIM: 20231001
Jenjang: Sarjana (S1)

Nama: Ghifar Dzikrullah
NIM: 20252001
Jenjang: Magister (S2)
```

Dalam contoh di atas, dibuat sebuah *interface* `MahasiswaInterface` yang mendefinisikan kontrak berupa tiga metode: `getNama()`, `getNim()`, dan `getJenjang()`. Dua class, yaitu `MahasiswaS1` dan `MahasiswaS2`, mengimplementasikan *interface* tersebut dengan cara mereka sendiri. Fungsi `infoMahasiswa()` menerima parameter bertipe `MahasiswaInterface`, sehingga dapat memproses objek dari *class* mana pun yang mengimplementasikan *interface* tersebut. Inilah bentuk nyata polimorfisme dengan *interface*: meskipun objek berasal dari *class* yang berbeda, keduanya dapat diperlakukan secara seragam karena mematuhi kontrak yang sama.

D. Studi Kasus: Implementasi Polimorfisme dalam PHP

Untuk memahami polimorfisme secara lebih nyata, mari kita terapkan dalam studi kasus sistem pembayaran mahasiswa. Dalam sistem ini, terdapat berbagai metode pembayaran, misalnya transfer bank dan pembayaran melalui *e-wallet*. Meskipun cara implementasi berbeda, semua metode pembayaran memiliki fungsi yang sama, yaitu melakukan pembayaran. Dengan polimorfisme, kita bisa menggunakan satu *interface* atau *abstract class* untuk mendefinisikan kontrak pembayaran, lalu setiap metode pembayaran mengimplementasikan sesuai kebutuhannya.

Contoh kode program:

```
<?php
// Interface sebagai kontrak umum
interface MetodePembayaran {
    public function bayar($jumlah);
}

// Implementasi pembayaran via Transfer Bank
class TransferBank implements MetodePembayaran {
    public function bayar($jumlah) {
            return "Pembayaran sebesar Rp" .
number_format($jumlah, 0, ',', '.') . " berhasil
melalui Transfer Bank.";
    }
}

// Implementasi pembayaran via E-Wallet
class EWallet implements MetodePembayaran {
    public function bayar($jumlah) {
            return "Pembayaran sebesar Rp" .
number_format($jumlah, 0, ',', '.') . " berhasil
melalui E-Wallet.";
    }
}

// Implementasi pembayaran via Kartu Kredit
class KartuKredit implements MetodePembayaran {
    public function bayar($jumlah) {
            return "Pembayaran sebesar Rp" .
number_format($jumlah, 0, ',', '.') . " berhasil
menggunakan Kartu Kredit.";
    }
}

// Fungsi polimorfisme
function prosesPembayaran(MetodePembayaran
$metode, $jumlah) {
    echo $metode->bayar($jumlah) . "<br>";
}

// Penggunaan polimorfisme
```

```
$bank = new TransferBank();
$ewallet = new EWallet();
$kartu = new KartuKredit();

// Semua objek diproses dengan cara yang sama
prosesPembayaran($bank, 1500000);
prosesPembayaran($ewallet, 500000);
prosesPembayaran($kartu, 2500000);
?>
```

Output:

```
Pembayaran sebesar Rp1.500.000 berhasil melalui Transfer Bank.
Pembayaran sebesar Rp500.000 berhasil melalui E-Wallet.
Pembayaran sebesar Rp2.500.000 berhasil menggunakan Kartu Kredit.
```

Dalam studi kasus ini, polimorfisme ditunjukkan dengan penggunaan *interface* `MetodePembayaran` sebagai kontrak umum. Masing-masing *class* (`TransferBank`, `EWallet`, dan `KartuKredit`) mengimplementasikan metode `bayar()` dengan cara yang berbeda, sesuai karakteristik masing-masing metode pembayaran. Namun, ketika dipanggil melalui fungsi `prosesPembayaran()`, semua objek dapat diproses dengan cara yang sama. Hal ini menunjukkan kekuatan polimorfisme, yaitu memberikan fleksibilitas dan konsistensi dalam memproses berbagai jenis objek yang berbeda namun berbagi perilaku yang sama. Dengan pendekatan ini, sistem lebih mudah dikembangkan, misalnya jika ingin menambahkan metode pembayaran baru, cukup membuat *class* baru tanpa mengubah kode yang sudah ada.

RANGKUMAN

Pada bab ini telah dibahas secara menyeluruh mengenai polimorfisme dalam PHP, yaitu kemampuan suatu *method* yang sama digunakan pada objek yang berbeda dengan hasil atau perilaku yang disesuaikan dengan implementasinya. Konsep ini membuat

kode lebih fleksibel, sederhana, dan mudah dikembangkan. Polimorfisme dapat diimplementasikan melalui beberapa cara:

1. Polimorfisme dengan *Class* Turunan, yaitu *method* yang sama di *class* induk dapat didefinisikan ulang pada *class* turunan sehingga memberikan perilaku berbeda sesuai kebutuhan.
2. Polimorfisme dengan *Abstract Class*, yaitu *class* abstrak digunakan sebagai kerangka dasar, di mana *method* abstrak harus diimplementasikan pada *class* turunannya dengan cara yang spesifik.
3. Polimorfisme dengan *Interface*, yaitu *interface* mendefinisikan kontrak yang harus diikuti oleh *class*, sehingga beberapa *class* dapat memiliki *method* yang sama namun dengan implementasi berbeda.

Dari studi kasus yang dibahas, terlihat bahwa polimorfisme sangat bermanfaat dalam membangun sistem yang kompleks. Dengan polimorfisme, pengembang dapat membuat kode yang konsisten, mudah diperluas, dan mendukung prinsip OOP seperti enkapsulasi dan pewarisan. Hal ini menjadikan polimorfisme sebagai salah satu pilar penting dalam pemrograman berorientasi objek, termasuk dalam PHP.

LATIHAN

1. Soal Teori

1. Jelaskan dengan kata-kata Anda sendiri apa yang dimaksud dengan polimorfisme dalam OOP.
2. Sebutkan perbedaan implementasi polimorfisme menggunakan *abstract class* dan *interface* dalam PHP.
3. Mengapa polimorfisme dapat membuat kode program lebih fleksibel dan mudah dikembangkan?
4. Apa yang akan terjadi jika sebuah *class* turunan tidak mengimplementasikan *method* dari *interface* yang digunakannya?
5. Bagaimana hubungan antara polimorfisme dan prinsip *Open*/*Closed Principle* dalam OOP?

2. Soal Praktik (Kode Program)

1. Buatlah sebuah *abstract class* bernama Kendaraan dengan *method* abstrak bergerak(). Lalu buat *class* Mobil dan Motor yang mengimplementasikan *method* tersebut dengan perilaku berbeda. Panggil semua objek menggunakan satu fungsi yang sama.
2. Buatlah sebuah *interface* bernama BangunDatar dengan *method* hitungLuas(). Implementasikan *interface* tersebut pada *class* Persegi dan Lingkaran. Tampilkan hasil luas dari masing-masing objek.
3. Kembangkan studi kasus pembayaran mahasiswa pada bab ini dengan menambahkan metode pembayaran baru, misalnya *Virtual Account*, tanpa mengubah kode pada fungsi prosesPembayaran().
4. Buatlah *class* Dosen dengan *method* mengajar(), lalu turunkan menjadi DosenTetap dan DosenHonorer. Implementasikan polimorfisme sehingga kedua *class* turunan dapat dipanggil dengan cara yang sama namun menghasilkan *output* berbeda.
5. Jelaskan hasil *output* dari soal nomor 4 dan analisis bagaimana polimorfisme bekerja pada kasus tersebut.

BAB 7
Abstraksi dan Interface dalam PHP

DESKRIPSI:

Bab ini membahas konsep abstraksi dan *interface* dalam pemrograman berorientasi objek menggunakan PHP. Abstraksi merupakan salah satu pilar utama OOP yang memungkinkan pengembang untuk menyembunyikan detail implementasi dan hanya menampilkan hal-hal penting yang relevan dengan pengguna *class*. Sementara itu, *interface* berfungsi sebagai kontrak yang harus dipatuhi oleh *class* yang mengimplementasikannya. Dengan mempelajari bab ini, mahasiswa dapat memahami bagaimana abstraksi dan *interface* dapat digunakan untuk membangun sistem yang fleksibel, konsisten, dan mudah diperluas.

TUJUAN PEMBELAJARAN:

Setelah mempelajari bab ini, mahasiswa diharapkan mampu:

1. Menjelaskan pengertian dan konsep dasar abstraksi dalam OOP PHP.
2. Membuat dan menggunakan *abstract class* serta *abstract method* dalam PHP.
3. Menjelaskan perbedaan antara *abstract class* dan *interface*.
4. Mengimplementasikan *interface* dalam PHP untuk mendefinisikan kontrak perilaku.
5. Menggunakan abstraksi dan *interface* dalam studi kasus sederhana.
6. Menerapkan prinsip OOP untuk membangun sistem yang terstruktur, efisien, dan mudah dikembangkan.

A. Pengertian Abstraksi dan Interface

Dalam pemrograman berorientasi objek (OOP), abstraksi dan *interface* merupakan dua konsep penting yang digunakan untuk menyembunyikan detail implementasi sekaligus menyediakan kerangka kerja yang konsisten bagi *class* turunan. Abstraksi adalah proses menyembunyikan detail teknis dari sebuah *class* dan hanya menampilkan fitur penting yang relevan dengan pengguna. Abstraksi biasanya diterapkan dengan menggunakan *abstract class* dan *abstract method*. *Abstract class* tidak dapat di instansiasi (tidak bisa dibuat objek langsung). *Abstract method* hanya memiliki deklarasi (nama dan parameter), tanpa isi, dan harus diimplementasikan pada *class* turunan. Abstraksi membantu menciptakan kerangka dasar (*blueprint*) sehingga *class* turunan dapat mengembangkan detail implementasi sesuai kebutuhan.

Interface adalah kontrak yang berisi sekumpulan *method* tanpa implementasi. Semua *class* yang mengimplementasikan *interface* wajib menuliskan *method-method* tersebut sesuai aturan *interface*. Berbeda dengan *abstract class*, sebuah *class* dapat mengimplementasikan lebih dari satu *interface*, sehingga lebih fleksibel. *Interface* berguna untuk menyediakan standar perilaku yang konsisten bagi berbagai *class*. Kemudian mendukung *multiple inheritance* di PHP (karena PHP tidak mendukung pewarisan ganda dengan *class*, tetapi bisa dengan *interface*). Selanjutnya mempermudah penerapan polimorfisme.

Contoh kode program: abstraksi

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

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

    abstract public function bersuara();
}

class Kucing extends Hewan {
    public function bersuara() {
```

```
        return $this->nama . " berkata: Meong!";
    }
}

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

$kucing = new Kucing("Mimi");
$anjing = new Anjing("Doggy");

echo $kucing->bersuara();
echo "<br>";
echo $anjing->bersuara();
?>
```

Output:

```
Mimi berkata: Meong!
Doggy berkata: Guk guk!
```

Contoh kode program: *interface*

```
<?php
interface Bentuk {
    public function hitungLuas();
}

class Persegi implements Bentuk {
    private $sisi;

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

    public function hitungLuas() {
         return "Luas Persegi: " . ($this->sisi *
$this->sisi);
    }
}

class Lingkaran implements Bentuk {
    private $jari;
```

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

    public function hitungLuas() {
        return "Luas Lingkaran: " . (3.14 * $this-
>jari * $this->jari);
    }
}

$persegi = new Persegi(4);
$lingkaran = new Lingkaran(7);

echo $persegi->hitungLuas();
echo "<br>";
echo $lingkaran->hitungLuas();
?>
```

Output:

```
Luas Persegi: 16
Luas Lingkaran: 153.86
```

Dari contoh di atas, dapat dilihat bahwa *abstract class* (`Hewan`) memberikan kerangka dasar dengan *method* abstrak `bersuara()` yang wajib diimplementasikan pada *class* turunan (`Kucing` dan `Anjing`). Sedangkan, *interface* (`Bentuk`) berfungsi sebagai kontrak yang mengharuskan *class* `Persegi` dan `Lingkaran` memiliki *method* `hitungLuas()`. Keduanya sama-sama menekankan pada ide "apa yang harus dilakukan", bukan "bagaimana cara melakukannya", namun *interface* lebih fleksibel karena mendukung penerapan pada banyak *class* sekaligus.

B. Abstract Class dan Abstract Method

Abstract class adalah *class* yang tidak dapat di instansiasi secara langsung, artinya kita tidak bisa membuat objek dari *class* tersebut. *Abstract class* biasanya digunakan sebagai kerangka dasar (*blueprint*) bagi *class* turunan. Di dalam *abstract class*, kita dapat mendefinisikan *property*, *method* biasa, maupun *abstract method*. Pengertian *Abstract method* adalah *method* yang hanya didefinisikan

nama dan parameternya tanpa implementasi (isi *body*). *Abstract method* harus diimplementasikan (di *override*) oleh setiap *class* turunan yang mewarisinya. Dengan menggunakan *abstract class* dan *abstract method*, kita bisa memastikan bahwa setiap *class* turunan memiliki perilaku (*method*) tertentu, meskipun cara implementasinya berbeda-beda.

Contoh kode program: *abstract class* dan *abstract method*:

```
<?php
// Abstract class
abstract class Kendaraan {
    protected $merk;

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

    // Abstract method (harus diimplementasikan
oleh class turunan)
    abstract public function bergerak();
}

// Class turunan
class Mobil extends Kendaraan {
    public function bergerak() {
         return "Mobil merk " . $this->merk . "
berjalan di jalan raya.";
    }
}

class Pesawat extends Kendaraan {
    public function bergerak() {
        return "Pesawat merk " . $this->merk . "
terbang di udara.";
    }
}

// Pemanggilan
$mobil = new Mobil("Toyota");
$pesawat = new Pesawat("Boeing");
```

```
echo $mobil->bergerak();
echo "<br>";
echo $pesawat->bergerak();
?>
```

Output:

```
Mobil merk Toyota berjalan di jalan raya.
Pesawat merk Boeing terbang di udara.
```

Dalam contoh di atas, *class* `Kendaraan` adalah *abstract class* yang memiliki *abstract method* `bergerak()`. *Abstract method* tersebut tidak memiliki isi dan harus diimplementasikan oleh *class* turunannya (`Mobil` dan `Pesawat`). Setiap *class* turunan mengimplementasikan `bergerak()` sesuai karakteristik masing-masing, sehingga walaupun *method* yang dipanggil sama, hasilnya berbeda. Dengan demikian, *abstract class* membantu menciptakan kerangka umum, sementara detail implementasi diserahkan pada masing-masing *class* turunan.

C. Perbedaan Abstract Class dan Interface

Meskipun *abstract class* dan *interface* sama-sama digunakan untuk mendukung abstraksi dalam OOP, keduanya memiliki perbedaan penting baik dari sisi penggunaan maupun fungsionalitas. Berikut perbedaannya:

1. Abstract Class
   - Didefinisikan dengan kata kunci *abstract*.
   - Dapat memiliki *property* (variabel) dan *method* (baik *abstract* maupun non-*abstract*).
   - Tidak bisa dibuat objek langsung, hanya bisa diturunkan.
   - Cocok digunakan ketika terdapat beberapa *method* umum yang bisa langsung diwarisi oleh *class* turunan, dan beberapa *method* harus diimplementasikan sendiri.
2. Interface
   - Didefinisikan dengan kata kunci *interface*.
   - Hanya boleh berisi deklarasi *method* (tanpa implementasi).
   - Tidak bisa memiliki *property* dengan nilai, tetapi bisa menggunakan konstanta (*const*).
   - Sebuah *class* bisa mengimplementasikan lebih dari satu *interface*, sehingga mendukung *multiple inheritance*.

- Cocok digunakan untuk mendefinisikan kontrak yang harus diikuti oleh banyak *class* dengan cara yang konsisten.

Untuk lebih jelasnya:

Tabel Perbedaan *Abstract Class* dan *Interface*

Aspek	Abstract Class	Interface
Kata kunci	*abstract*	*interface*
Objek	Tidak bisa dibuat objek langsung	Tidak bisa dibuat objek langsung
Method	Bisa berisi abstract method dan method biasa	Hanya bisa berisi deklarasi method
Property	Bisa memiliki property dengan nilai awal	Tidak bisa punya property (hanya konstanta)
Pewarisan	Satu class hanya bisa extend satu abstract class	Satu class bisa implement banyak interface
Tujuan utama	Memberikan kerangka dasar dengan sebagian implementasi	Memberikan kontrak perilaku yang wajib diikuti

Contoh perbandingan:

```
<?php
// Abstract class
abstract class Kendaraan {
    protected $merk;
    public function __construct($merk) {
        $this->merk = $merk;
    }
    abstract public function bergerak();
    public function info() {
       return "Ini adalah kendaraan merk " . $this-
>merk;
    }
}

// Interface
interface Mesin {
    public function nyalakan();
}

// Class turunan
```

```
class Mobil extends Kendaraan implements Mesin {
    public function bergerak() {
         return "Mobil merk " . $this->merk . "
berjalan di jalan.";
    }
    public function nyalakan() {
        return "Mesin mobil dinyalakan.";
    }
}

// Pemanggilan
$mobil = new Mobil("Toyota");
echo $mobil->info() . "<br>";
echo $mobil->bergerak() . "<br>";
echo $mobil->nyalakan();
?>
```

Output:

```
Ini adalah kendaraan merk Toyota
Mobil merk Toyota berjalan di jalan.
Mesin mobil dinyalakan.
```

Dalam contoh tersebut, `Kendaraan` adalah *abstract class* yang memberikan kerangka dasar dengan *method* `info()` yang sudah memiliki implementasi, dan `bergerak()` yang wajib diimplementasikan oleh *class* turunan. Sedangkan `Mesin` adalah *interface* yang hanya mendefinisikan *method* `nyalakan()` tanpa implementasi. *Class* `Mobil` dapat mewarisi *abstract class* sekaligus mengimplementasikan *interface* sehingga lebih fleksibel.

D. Studi Kasus: Implementasi Abstraksi dan Interface dalam PHP

Untuk memahami penerapan abstraksi dan *interface* secara nyata, mari kita lihat studi kasus sistem pembayaran universitas. Dalam sistem ini, setiap mahasiswa dapat melakukan pembayaran dengan berbagai metode, seperti transfer bank, *e-wallet*, atau kartu kredit. Masing-masing metode pembayaran memiliki cara kerja berbeda, tetapi semuanya memiliki perilaku umum yaitu melakukan pembayaran. Dengan abstraksi, kita bisa membuat *abstract class* Pembayaran sebagai kerangka dasar. Sedangkan dengan *interface*, kita bisa membuat kontrak Notifikasi yang harus diimplementasikan

oleh semua metode pembayaran agar dapat mengirim notifikasi setelah transaksi berhasil.

Contoh kode program:

```
<?php
// Abstract class
abstract class Pembayaran {
    protected $jumlah;

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

    // Abstract method
    abstract public function prosesPembayaran();
}

// Interface
interface Notifikasi {
    public function kirimNotifikasi($pesan);
}

// Implementasi pembayaran via Bank
class TransferBank extends Pembayaran implements
Notifikasi {
    public function prosesPembayaran() {
            return "Pembayaran sebesar Rp" .
number_format($this->jumlah, 0, ',', '.') . "
berhasil via Transfer Bank.";
    }
    public function kirimNotifikasi($pesan) {
        return "Notifikasi Bank: " . $pesan;
    }
}

// Implementasi pembayaran via E-Wallet
class EWallet extends Pembayaran implements
Notifikasi {
    public function prosesPembayaran() {
            return "Pembayaran sebesar Rp" .
number_format($this->jumlah, 0, ',', '.') . "
berhasil via E-Wallet.";
```

```
    }
    public function kirimNotifikasi($pesan) {
        return "Notifikasi E-Wallet: " . $pesan;
    }
}

// Implementasi pembayaran via Kartu Kredit
class KartuKredit extends Pembayaran implements
Notifikasi {
    public function prosesPembayaran() {
            return "Pembayaran sebesar Rp" .
number_format($this->jumlah, 0, ',', '.') . "
berhasil via Kartu Kredit.";
    }
    public function kirimNotifikasi($pesan) {
          return "Notifikasi Kartu Kredit: " .
$pesan;
    }
}

// Pemanggilan
$bank = new TransferBank(1500000);
echo $bank->prosesPembayaran() . "<br>";
echo $bank->kirimNotifikasi("Transaksi Anda telah
diproses.") . "<br><br>";

$ewallet = new EWallet(500000);
echo $ewallet->prosesPembayaran() . "<br>";
echo  $ewallet->kirimNotifikasi("Saldo berhasil
terpotong.") . "<br><br>";

$kartu = new KartuKredit(2500000);
echo $kartu->prosesPembayaran() . "<br>";
echo $kartu->kirimNotifikasi("Tagihan Anda sudah
masuk.") . "<br>";
?>
```

Output:

```
Pembayaran sebesar Rp1.500.000 berhasil via
Transfer Bank.
Notifikasi Bank: Transaksi Anda telah diproses.
```

```
Pembayaran sebesar Rp500.000 berhasil via E-
Wallet.
Notifikasi E-Wallet: Saldo berhasil terpotong.

Pembayaran sebesar Rp2.500.000 berhasil via Kartu
Kredit.
Notifikasi Kartu Kredit: Tagihan Anda sudah masuk.
```

Dari studi kasus di atas, dapat dilihat bahwa *abstract class* Pembayaran digunakan untuk menyediakan kerangka dasar berupa *property* `$jumlah` dan *method* abstrak `prosesPembayaran()`. Setiap metode pembayaran (`TransferBank`, `EWallet`, `KartuKredit`) wajib mengimplementasikan *method* ini sesuai karakteristik masing-masing. Sementara itu, *interface* Notifikasi memastikan bahwa semua metode pembayaran memiliki kemampuan untuk mengirim notifikasi setelah transaksi berhasil. Dengan menggabungkan abstraksi dan *interface*, sistem menjadi lebih fleksibel, konsisten, dan mudah diperluas. Jika suatu saat ditambahkan metode pembayaran baru (misalnya *Virtual Account*), cukup membuat *class* baru yang mewarisi *abstract class* `Pembayaran` dan mengimplementasikan *interface* `Notifikasi`, tanpa perlu mengubah kode yang sudah ada.

RANGKUMAN

Pada bab ini telah dibahas mengenai Abstraksi dan Interface dalam PHP sebagai bagian penting dari paradigma Pemrograman Berorientasi Objek (OOP).

1. Abstraksi adalah konsep penyembunyian detail implementasi dan hanya menampilkan bagian penting kepada pengguna. Dengan menggunakan *abstract class*, kita dapat membuat kerangka dasar yang memaksa kelas turunan untuk mengimplementasikan metode tertentu. Hal ini bermanfaat untuk menciptakan struktur kode yang konsisten dan memudahkan pengembangan aplikasi yang kompleks.
2. *Interface* merupakan kontrak yang mendefinisikan metode yang harus diimplementasikan oleh kelas tanpa menyediakan

implementasi apa pun. *Interface* mendukung prinsip *multiple inheritance* dalam PHP, sehingga sebuah kelas dapat mengimplementasikan lebih dari satu *interface* sekaligus.

3. Perbedaan utama:
   - *Abstract class* dapat memiliki properti dan metode (baik abstrak maupun konkret), sedangkan *interface* hanya mendefinisikan metode tanpa implementasi.
   - Satu kelas hanya bisa mewarisi satu *abstract class* (*single inheritance*), tetapi bisa mengimplementasikan banyak *interface*.
4. Dengan menggabungkan abstraksi dan *interface*, kita bisa membangun sistem yang fleksibel, terstruktur, dan mudah diperluas. Studi kasus sistem pembayaran menunjukkan bagaimana *abstract class* Pembayaran digunakan sebagai kerangka dasar, sementara *interface* Notifikasi memastikan setiap metode pembayaran memiliki mekanisme pengiriman notifikasi.

Secara keseluruhan, abstraksi dan *interface* membantu menjaga kode agar bersih, konsisten, dan lebih mudah dikelola dalam pengembangan aplikasi berbasis PHP OOP.

LATIHAN

1. Latihan 1: Abstract Class
   Buatlah sebuah *abstract class* bernama `Hewan` yang memiliki:
   - Properti `nama`.
   - *Abstract method* `suara()`.

   Buat dua kelas turunan, yaitu `Kucing` dan `Anjing`, yang mengimplementasikan *method* `suara()`. Tampilkan hasil pemanggilan suara hewan tersebut.

2. Latihan 2: Interface
   Buatlah sebuah *interface* bernama `Kendaraan` yang memiliki *method* `jalan()`.

- Buat dua *class*, `Sepeda` dan `Motor`, yang mengimplementasikan *interface* tersebut.
- Tampilkan hasil pemanggilan `jalan()` pada kedua objek.

3. Latihan 3: Kombinasi Abstract Class dan Interface
   - Buat *abstract class* bernama `AlatElektronik` dengan *abstract method* `fungsi()`.
   - Buat *interface* `Garansi` dengan *method* `lamaGaransi()`.
   - Implementasikan keduanya pada *class* `Televisi` dan `Kulkas`.
   - Tampilkan fungsi dan lama garansi dari masing-masing objek.

4. Latihan 4: Studi Kasus Mini Project
   Buatlah sebuah sistem sederhana untuk Metode Pembayaran:
   - Buat *abstract class* `Pembayaran` dengan *abstract method* `prosesPembayaran($jumlah)`.
   - Buat *interface* `Notifikasi` dengan *method* `kirimNotifikasi($pesan)`.
   - Buat *class* `TransferBank` dan `EWallet` yang mengimplementasikan keduanya.
   - Buat objek dari masing-masing *class*, kemudian simulasikan pembayaran dan pengiriman notifikasi.

BAB 8
Trait dan Namespace dalam PHP OOP

DESKRIPSI:

Bab ini membahas dua fitur penting dalam PHP OOP, yaitu *Trait* dan *Namespace*. *Trait* digunakan untuk mendukung konsep *code reusability* dengan cara menggabungkan kembali potongan kode ke dalam kelas yang berbeda, sedangkan *Namespace* digunakan untuk menghindari konflik nama (*name collision*) antara *class*, *function*, atau *constant* dalam aplikasi yang besar. Dengan memahami keduanya, mahasiswa akan mampu mengorganisasi kode dengan lebih baik, efisien, dan terstruktur.

TUJUAN PEMBELAJARAN:

Setelah mempelajari bab ini, mahasiswa diharapkan dapat:

1. Menjelaskan konsep *Trait* dan kegunaannya dalam OOP PHP.
2. Mengimplementasikan *Trait* ke dalam *class* untuk mendukung *code reusability*.
3. Memahami pengertian *Namespace* dan manfaatnya dalam mengelola kode skala besar.
4. Menggunakan *Namespace* untuk menghindari konflik nama antar *class* atau *function*.
5. Menerapkan kombinasi *Trait* dan *Namespace* dalam sebuah studi kasus sederhana.

A. Pengertian Trait dalam PHP

Dalam pemrograman berorientasi objek, *Trait* adalah sebuah mekanisme di PHP yang memungkinkan kita untuk menyusun ulang kode dan membagikannya ke beberapa *class* tanpa harus

menggunakan pewarisan (*inheritance*). *Trait* diciptakan untuk mengatasi keterbatasan PHP yang hanya mendukung *single inheritance* (satu *class* hanya bisa mewarisi dari satu *class* induk). Dengan *Trait*, dapat membuat sekumpulan *method* yang bisa digunakan kembali (*reusable*) oleh banyak *class* yang berbeda, meskipun *class-class* tersebut tidak memiliki hubungan pewarisan.
Contoh kode program:

```
<?php
trait Pesan {
    public function kirimPesan($pesan) {
        echo "Pesan: " . $pesan . "<br>";
    }
}

trait Log {
    public function catatLog($aktivitas) {
         echo "Log aktivitas: " . $aktivitas .
"<br>";
    }
}

class User {
    use Pesan, Log; // menggunakan 2 trait
}

class Admin {
    use Pesan; // hanya menggunakan trait Pesan
}

// Pemanggilan
$user = new User();
$user->kirimPesan("Halo, ini pesan dari User.");
$user->catatLog("User login ke sistem.");

$admin = new Admin();
$admin->kirimPesan("Pesan dari Admin.");
?>
```

Output:

```
Pesan: Halo, ini pesan dari User.
Log aktivitas: User login ke sistem.
```

```
Pesan: Pesan dari Admin.
```

Pada contoh program di atas, *Trait* digunakan untuk memisahkan fungsionalitas tertentu agar dapat digunakan kembali oleh beberapa *class*. *Trait* Pesan menyediakan *method* `kirimPesan()`, sedangkan *Trait* `Log` menyediakan *method* `catatLog()`. *Class* `User` menggunakan kedua *trait* tersebut sehingga memiliki kemampuan untuk mengirim pesan sekaligus mencatat log aktivitas. Sementara itu, *class* `Admin` hanya menggunakan *Trait* `Pesan`, sehingga hanya dapat mengirim pesan. Hal ini menunjukkan bahwa *Trait* sangat berguna untuk membagi fungsionalitas ke berbagai *class* tanpa harus menduplikasi kode atau terikat pada pewarisan tunggal, sehingga kode menjadi lebih fleksibel, efisien, dan mudah dipelihara.

B. Pengertian Namespace dalam PHP

Namespace dalam PHP adalah sebuah cara untuk mengelompokkan *class*, *interface*, *function*, dan *constant* ke dalam sebuah ruang lingkup tertentu sehingga tidak terjadi konflik nama (*name collision*). *Namespace* sangat berguna ketika kita membangun aplikasi berskala besar atau menggunakan berbagai *library* pihak ketiga yang mungkin memiliki *class* atau *function* dengan nama yang sama. Dengan namespace, kita bisa mengorganisasi kode agar lebih rapi, terstruktur, dan mudah dipahami. PHP memperkenalkan *namespace* sejak versi PHP 5.3 untuk mendukung pengembangan aplikasi modern.

Contoh kode program:

1. Perpustakaan/Buku.php

```
<?php
// File: Perpustakaan/Buku.php
namespace Perpustakaan;

class Buku {
    public function info() {
          return "Ini adalah class Buku dari
namespace Perpustakaan.";
    }
}
```

2. Toko/Buku.php

```
<?php
// File: Toko/Buku.php
namespace Toko;

class Buku {
    public function info() {
          return "Ini adalah class Buku dari
namespace Toko.";
    }
}
```

3. Indeks.php

```
<?php
// File utama
require 'Perpustakaan/Buku.php';
require 'Toko/Buku.php';

// Menggunakan namespace Perpustakaan
$perpusBuku = new \Perpustakaan\Buku();
echo $perpusBuku->info();
echo "<br>";

// Menggunakan namespace Toko
$tokoBuku = new \Toko\Buku();
echo $tokoBuku->info();
?>
```

Output:

```
Ini adalah class Buku dari namespace Perpustakaan.
Ini adalah class Buku dari namespace Toko.
```

Dalam contoh tersebut terdapat dua *class* dengan nama sama, yaitu `Buku`, tetapi keduanya berada di *namespace* yang berbeda: `Perpustakaan` dan `Toko`. Dengan adanya *namespace*, PHP dapat membedakan kedua *class* tersebut sehingga tidak terjadi *error* akibat konflik nama. Pada saat membuat objek, kita menggunakan *prefix* `\Perpustakaan\Buku` atau `\Toko\Buku` untuk menentukan *namespace* mana yang dimaksud. Dengan demikian, *namespace* membantu pengembang dalam mengelola kode yang kompleks dan menghindari tabrakan nama antar *class* atau *library*.

C. Menggunakan Trait dan Namespace Bersamaan

Dalam pengembangan aplikasi skala besar, sering kali kita menggunakan *Trait* untuk mengelola fungsionalitas yang dapat digunakan kembali (*reusable*) dan *Namespace* untuk mengatur struktur kode agar lebih rapi serta menghindari konflik nama. PHP memungkinkan penggunaan *Trait* di dalam sebuah *namespace*, sehingga kode menjadi lebih modular dan mudah dipelihara.

Contoh kode program:

1. Library/Helper/Pesan.php

```
<?php
// File: Library/Helper/Pesan.php
namespace Library\Helper;

trait Pesan {
    public function kirimPesan($pesan) {
        echo "Pesan: " . $pesan . "<br>";
    }
}
?>
```

2. App/User.php

```
<?php
// File: App/User.php
namespace App;

use Library\Helper\Pesan; // menggunakan Trait
dari namespace lain

class User {
    use Pesan;

    public function infoUser($nama) {
        echo "User: " . $nama . "<br>";
    }
}
?>
```

3. index.php

```
<?php
// File utama: index.php
require 'Library/Helper/Pesan.php';
require 'App/User.php';
```

```
$user = new \App\User();
$user->infoUser("Mizard");
$user->kirimPesan("Halo, selamat datang di sistem!");
?
```

Output:

```
User: Mizard
Pesan: Halo, selamat datang di sistem!
```

Penjelasan Dalam contoh di atas, *Trait* Pesan didefinisikan di dalam *namespace* `Library\Helper`. Kemudian, *class* `User` yang berada pada namespace `App` menggunakan Trait tersebut dengan perintah `use Library\Helper\Pesan;`. Hal ini menunjukkan bagaimana *Trait* dan *Namespace* dapat dikombinasikan: *Namespace* digunakan untuk mengelompokkan kode berdasarkan fungsi atau modul, sementara *Trait* digunakan untuk membagikan fungsionalitas tertentu. Dengan cara ini, kode menjadi terstruktur, modular, dan mudah diperluas.

D. Studi Kasus: Implementasi Trait dan Namespace dalam PHP

Untuk memahami penggunaan *Trait* dan *Namespace* secara nyata, mari kita buat studi kasus sederhana: Sistem Manajemen Pengguna. Sistem ini terdiri dari modul *Helper* yang menyediakan *Trait* untuk *logging* aktivitas, serta modul App yang berisi *class* `User` dan `Admin`. Dengan *Namespace*, setiap modul dipisahkan ke dalam ruang lingkup berbeda, sehingga kode lebih terorganisir.

Struktur folder:

```
/Library
    /Helper
        Log.php
/App
    User.php
    Admin.php
index.php
```

Kode program:

1. Trait Log (Library/Helper/Log.php)

```
<?php
namespace Library\Helper;
```

```
trait Log {
    public function catatLog($aktivitas) {
        echo "[LOG] " . $aktivitas . "<br>";
    }
}
?>
```

2. Class User (App/User.php)

```
<?php
namespace App;

use Library\Helper\Log;

class User {
    use Log;

    private $nama;

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

    public function login() {
           $this->catatLog("User  {$this->nama} berhasil login.");
    }
}
?>
```

3. Class Admin (App/Admin.php)

```
<?php
namespace App;

use Library\Helper\Log;

class Admin {
    use Log;

    private $nama;

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

```
    }

    public function hapusUser($user) {
          $this->catatLog("Admin  {$this->nama}
menghapus user {$user}.");
    }
}
?>
```

4. File Utama (index.php)

```
<?php
require 'Library/Helper/Log.php';
require 'App/User.php';
require 'App/Admin.php';

$user = new \App\User("Mizard");
$user->login();

$admin = new \App\Admin("Dzimar");
$admin->hapusUser("Mizard");
?>
```

Output:

```
[LOG] User Mizard berhasil login.
[LOG] Admin Dzimar menghapus user Mizard.
```

Dalam studi kasus ini, *Trait Log* berada pada *namespace* `Library\Helper` dan digunakan oleh dua *class* berbeda (`User` dan `Admin`) yang berada di *namespace* `App`. Dengan cara ini, kedua *class* dapat berbagi fungsionalitas pencatatan log tanpa harus menuliskan ulang kode. *Namespace* membuat struktur kode lebih teratur, sehingga modul *Helper* dan App terpisah jelas sesuai tanggung jawabnya. Studi kasus ini menunjukkan bahwa kombinasi *Trait* dan *Namespace* sangat efektif untuk membangun aplikasi yang modular, *reusable*, dan terorganisir.

RANGKUMAN

Pada bab ini telah dibahas mengenai *Trait* dan *Namespace* dalam OOP PHP yang menjadi solusi untuk membangun aplikasi berskala besar agar tetap rapi, modular, dan mudah dikelola.

1. *Trait* adalah mekanisme di PHP yang digunakan untuk berbagi fungsionalitas ke berbagai *class* tanpa perlu menggunakan pewarisan. Dengan *Trait*, kode dapat ditulis sekali lalu digunakan kembali oleh banyak *class* yang berbeda, sehingga mendukung *code reusability* dan mengatasi keterbatasan *single inheritance*.
2. *Namespace* adalah ruang lingkup yang digunakan untuk mengelompokkan *class*, *function*, dan *constant* agar tidak terjadi konflik nama (*name collision*). *Namespace* sangat penting ketika aplikasi melibatkan banyak modul atau *library* eksternal.
3. PHP memungkinkan penggunaan *Trait* di dalam *Namespace*, sehingga kode dapat sekaligus terorganisir secara struktur (*namespace*) dan efisien dalam pemakaian ulang (*trait*).
4. Studi kasus Sistem Manajemen Pengguna menunjukkan bagaimana *Trait Log* yang berada di *namespace Library*/*Helper* dapat digunakan oleh *class User* dan Admin di *namespace* App. Hasilnya, fungsionalitas pencatatan log dapat digunakan oleh kedua *class* tanpa menduplikasi kode, sementara *namespace* menjaga agar struktur kode tetap terpisah sesuai tanggung jawab modul.

Secara keseluruhan, pemahaman mengenai *Trait* dan *Namespace* sangat membantu dalam menciptakan aplikasi PHP berbasis OOP yang terstruktur, modular, *reusable*, dan mudah dikembangkan.

LATIHAN

1. Latihan 1: Trait Sederhana
   Buat sebuah *Trait* bernama Cetak yang memiliki *method* `cetakData($data)`. Gunakan *Trait* tersebut di dalam *class* `Produk` untuk mencetak informasi produk.

2. Latihan 2: Trait Multi-Class
   Buat *Trait* bernama HitungLuas yang berisi *method* `luasPersegi($sisi)` dan `luasLingkaran($jari2)`.

Gunakan *Trait* tersebut pada dua *class* berbeda: `Persegi` dan `Lingkaran`, lalu tampilkan hasil perhitungannya.

3. Latihan 3: Namespace Dasar
   Buat dua class dengan nama sama yaitu Buku.
   - *Class* pertama ada di *namespace* `Perpustakaan` dengan *method* `info()` yang menampilkan "Buku di perpustakaan".
   - *Class* kedua ada di *namespace* `Toko` dengan *method* `info()` yang menampilkan "Buku di toko".
   - Buat file utama untuk menampilkan keduanya tanpa terjadi konflik nama.

4. Latihan 4: Trait + Namespace
   Buat *Trait* bernama Log di *namespace* `Library\Helper` dengan *method* `catat($aktivitas)`.
   Buat *class* `Mahasiswa` dan `Dosen` di *namespace* App yang sama-sama menggunakan *Trait* tersebut.
   - `Mahasiswa` memiliki *method* `belajar()`.
   - `Dosen` memiliki *method* `mengajar()`.

   Jalankan kedua *class* tersebut pada file utama dan tampilkan hasil log aktivitasnya.

BAB 9
Exception Handling dalam OOP PHP

DESKRIPSI:

Bab ini membahas tentang *Exception Handling* dalam pemrograman berorientasi objek (OOP) menggunakan PHP. *Exception Handling* adalah mekanisme untuk menangani kesalahan (*error*) atau kondisi tidak normal dalam program tanpa menghentikan eksekusi secara tiba-tiba. Dengan menggunakan *exception*, program dapat mengantisipasi *error*, memberikan pesan yang lebih informatif, serta menjaga alur eksekusi agar tetap terkendali. Dalam OOP PHP, *exception* ditangani menggunakan *class Exception* dan blok *try*, *catch*, serta *finally*.

TUJUAN PEMBELAJARAN:

Setelah mempelajari bab ini, mahasiswa diharapkan mampu:

1. Menjelaskan konsep *Exception* dalam PHP.
2. Memahami cara kerja blok *try*, *catch*, dan *finally* untuk menangani *error*.
3. Membuat dan menggunakan *custom exception* dengan mendefinisikan *class exception* sendiri.
4. Menggunakan *Exception Handling* untuk meningkatkan keamanan dan stabilitas program.
5. Menerapkan *Exception Handling* pada studi kasus sederhana dalam PHP berbasis OOP.

A. Pengertian Exception dalam PHP

Exception adalah suatu peristiwa tidak normal (*error*) yang terjadi saat eksekusi program dan dapat mengganggu jalannya aplikasi. Dalam PHP, *exception* merupakan objek dari *class Exception* yang

mewakili kondisi kesalahan. Jika tidak ditangani, *exception* akan menyebabkan program berhenti dan menampilkan pesan *error*. Untuk menghindari hal tersebut, PHP menyediakan mekanisme *Exception Handling*, yaitu cara untuk menangkap dan menangani *error* menggunakan blok *try*, *catch*, dan *finally*. Dengan mekanisme ini, *programmer* dapat mengontrol apa yang harus dilakukan ketika terjadi *error*, misalnya menampilkan pesan khusus, menulis log *error*, atau memberikan alternatif solusi, sehingga program tetap berjalan dengan baik.

Contoh kode sederhana *exception handling*:

```
<?php
try {
    // Mencoba kode yang berpotensi error
    $angka1 = 10;
    $angka2 = 0;

    if ($angka2 == 0) {
        throw new Exception("Pembagian dengan nol
tidak diperbolehkan!");
    }

    $hasil = $angka1 / $angka2;
    echo "Hasil: " . $hasil;
} catch (Exception $e) {
    // Menangkap error dan menampilkan pesan
    echo "Terjadi error: " . $e->getMessage();
} finally {
    // Bagian ini selalu dijalankan, error atau
tidak
    echo "<br>Proses selesai.";
}
?>
```

Output:

```
Terjadi error: Pembagian dengan nol tidak
diperbolehkan!
Proses selesai.
```

Pada contoh di atas, blok `try` digunakan untuk menampung kode yang berpotensi menimbulkan *error*, yaitu pembagian dengan angka nol. Jika kondisi *error* terpenuhi, maka `throw new`

`Exception()` akan melemparkan sebuah *exception* dengan pesan tertentu. *Exception* tersebut ditangkap oleh blok `catch`, sehingga program tidak berhenti mendadak, melainkan menampilkan pesan *error* yang lebih ramah. Sementara itu, blok `finally` akan tetap dijalankan baik terjadi *error* maupun tidak, misalnya untuk menutup koneksi *database* atau menulis log. Dengan demikian, *Exception Handling* membantu menjaga agar program tetap stabil, aman, dan mudah dipelihara.

B. Struktur Dasar Exception Hnadling

Dalam PHP, *exception handling* dilakukan dengan tiga blok utama, yaitu *try*, *catch*, dan *finally*. Struktur dasarnya sebagai berikut:

1. *try* → berisi kode yang berpotensi menimbulkan *error*.
2. *catch* → digunakan untuk menangkap dan menangani *exception* yang dilempar dari blok *try*.
3. *finally* → bagian opsional yang akan selalu dijalankan, baik terjadi *exception* maupun tidak. Biasanya digunakan untuk membersihkan *resource*, menutup koneksi *database*, atau tindakan akhir lainnya.

Contoh kode program:

```
<?php
try {
    echo "Memulai proses...<br>";

    // Contoh kode yang bisa error
    $nilai = 0;
    if ($nilai === 0) {
          throw new Exception("Nilai tidak boleh
nol!");
    }

    echo "Nilai: " . $nilai;
} catch (Exception $e) {
    echo "Terjadi exception: " . $e->getMessage();
} finally {
    echo "<br>Bagian finally selalu dijalankan.";
}
?>
```

Output:

```
Memulai proses...
Terjadi exception: Nilai tidak boleh nol!
Bagian finally selalu dijalankan.
```

Dari contoh di atas, blok *try* berisi kode yang memeriksa nilai variabel `$nilai`. Karena nilainya nol, maka `throw new Exception()` digunakan untuk melempar pesan error. Pesan ini kemudian ditangkap oleh blok *catch*, sehingga program tidak berhenti mendadak, melainkan memberikan informasi yang lebih jelas kepada pengguna. Sementara itu, blok *finally* tetap dijalankan meskipun terjadi *exception*, sehingga dapat dipastikan bahwa proses akhir, seperti pembersihan *resource*, tetap dilakukan. Struktur ini membuat aplikasi lebih terkendali, aman, dan profesional.

C. Membuat Exception Kustom (Custom Exception)

Dalam pemrograman berorientasi objek, *Exception Handling* berfungsi untuk menangani *error* agar program tidak langsung berhenti, melainkan memberikan mekanisme penanganan yang lebih aman. Meskipun PHP sudah menyediakan *class Exception* bawaan, terkadang kebutuhan aplikasi menuntut penanganan *error* yang lebih spesifik dan kontekstual. Misalnya, ketika kita membuat sistem akademik, kesalahan terkait nilai mahasiswa tentu berbeda dengan kesalahan *log in* pengguna atau kesalahan koneksi *database*. Untuk itu, dapat membuat *Exception Kustom* dengan cara mendefinisikan *class* baru yang mewarisi (*extends*) *class Exception*. Dengan pendekatan ini, bisa membuat berbagai jenis *error* sesuai domain aplikasi, sehingga ketika terjadi kesalahan, program tidak hanya memberi tahu "*Error* umum", tetapi bisa memberikan pesan yang jelas, spesifik, dan mudah ditindaklanjuti.

Contoh kode program:

```
<?php
// Membuat Exception Kustom untuk validasi nilai
class NilaiException extends Exception {}

// Fungsi untuk memeriksa nilai
function cekNilai($nilai) {
    if ($nilai < 0) {
```

```
        throw new NilaiException("Nilai tidak boleh
negatif!");
    } elseif ($nilai > 100) {
        throw new NilaiException("Nilai tidak boleh
lebih dari 100!");
    } else {
        return "Nilai valid: $nilai";
    }
}

// Pengujian
try {
    echo cekNilai(120); // Input nilai lebih dari
100
} catch (NilaiException $e) {
    echo "Terjadi Error: " . $e->getMessage();
} finally {
    echo "<br>Proses pengecekan nilai selesai.";
}
?>
```

Output:

```
Terjadi Error: Nilai tidak boleh lebih dari 100!
Proses pengecekan nilai selesai.
```

Penjelasan kode:

1. Deklarasi Exception Kustom
   - `class NilaiException extends Exception {}` Membuat kelas `NilaiException` yang merupakan turunan dari *class* `Exception`. Dengan ini, kita bisa menambahkan pesan *error* khusus terkait validasi nilai.
2. Fungsi `cekNilai($nilai)`
   - Fungsi ini bertugas memvalidasi nilai.
   - Jika nilai < 0 → lempar *exception* dengan pesan "Nilai tidak boleh negatif!".
   - Jika nilai > 100 → lempar *exception* dengan pesan "Nilai tidak boleh lebih dari 100!".
   - Jika nilai valid (0–100) → kembalikan pesan "Nilai valid".
3. Blok `try...catch...finally`
   - `try` digunakan untuk mengeksekusi fungsi yang berpotensi

menimbulkan *error*.

- `catch (NilaiException $e)` menangkap *error* khusus `NilaiException`, lalu menampilkan pesan *error*.
- `finally` selalu dijalankan, baik terjadi *error* maupun tidak. Dalam contoh ini, digunakan untuk menampilkan pesan bahwa proses pengecekan selesai.

Kode di atas memperlihatkan keunggulan penggunaan *Exception Kustom*. Dengan membuat kelas `NilaiException`, *error* yang ditangani menjadi lebih spesifik sesuai konteks permasalahan, yaitu validasi nilai mahasiswa. Tanpa *exception kustom*, *error* biasanya hanya berupa pesan umum dari sistem yang kurang informatif. Dengan pendekatan ini, kita bisa membedakan jenis kesalahan — misalnya kesalahan *input*, kesalahan autentikasi, atau kesalahan koneksi *database* — masing-masing dengan *class exception*-nya sendiri. Selain itu, blok *try-catch-finally* memungkinkan program tetap berjalan meskipun terjadi *error*. Hal ini penting agar program tidak berhenti secara tiba-tiba, melainkan tetap memberikan pesan yang jelas kepada pengguna sekaligus menjaga alur eksekusi tetap aman. Dengan *exception kustom*, pengembang juga bisa memperluas fungsionalitas *class exception*, seperti menambahkan atribut tambahan (contoh: kode *error*, tingkat keparahan, atau kategori kesalahan). Kesimpulannya, penggunaan *Custom Exception* dalam OOP PHP membantu menciptakan kode yang lebih terstruktur, mudah dipelihara, serta lebih komunikatif dalam memberikan informasi *error* kepada pengguna maupun developer.

D. Studi Kasus: Exception Handling dalam PHP

Dalam pengembangan aplikasi nyata, *Exception Handling* sangat penting untuk menjaga agar sistem tetap stabil ketika terjadi kesalahan. Salah satu kasus yang sering ditemui adalah sistem *log in* pengguna dan validasi nilai mahasiswa. Dengan *Exception Handling*, kesalahan *input* dapat ditangani secara tepat tanpa menghentikan jalannya program.

Contoh studi kasus: sistem *log in* dengan *exception handling*:

```
<?php
// Membuat Exception Kustom untuk Login
class LoginException extends Exception {}

// Fungsi untuk login
function login($username, $password) {
    $userDB = "admin";
    $passDB = "12345";

    if ($username !== $userDB) {
         throw new LoginException("Username tidak
ditemukan!");
    }

    if ($password !== $passDB) {
           throw  new  LoginException("Password
salah!");
    }

     return  "Login  berhasil,  selamat  datang
$username!";
}

// Pengujian
try {
    echo login("admin", "1234"); // Salah password
} catch (LoginException $e) {
    echo "Error Login: " . $e->getMessage();
} finally {
    echo "<br>Proses login selesai.";
}
?>
```

Output:

```
Error Login: Password salah!
Proses login selesai.
```

Dalam contoh di atas, dibuat *class* `LoginException` untuk menangani kesalahan autentikasi. Fungsi `login()` melakukan pengecekan *username* dan *password*. Jika salah satu

tidak sesuai, maka dilemparkan *exception kustom* dengan pesan spesifik seperti "*Username* tidak ditemukan" atau "*Password* salah". Dengan cara ini, program bisa tetap berjalan dan memberikan pesan kesalahan yang jelas kepada pengguna. Blok *finally* memastikan bahwa pesan "Proses *log in* selesai" selalu ditampilkan, baik *log in* berhasil maupun gagal.

Keunggulan pendekatan ini adalah:

1. Pesan *error* lebih terarah sesuai konteks masalah.
2. Program tetap berjalan stabil meski terjadi *error*.
3. Mudah diperluas dengan membuat lebih banyak *exception kustom*, misalnya `DatabaseException` atau `ValidationException`.
4. Membantu pengembang melakukan *debugging* lebih cepat karena pesan *error* lebih spesifik.

RANGKUMAN

Pada bab ini telah dibahas mengenai konsep *Exception Handling* dalam OOP PHP, yaitu mekanisme untuk menangani kesalahan (*error*) agar program tidak berhenti secara tiba-tiba. *Exception Handling* memberikan cara yang lebih terstruktur dalam mengelola *error* dibandingkan penggunaan `if-else` biasa. Beberapa poin penting yang telah dipelajari antara lain:

1. Pengertian Exception Handling
   Exception adalah kondisi abnormal atau kesalahan yang terjadi selama program berjalan. Dengan *Exception Handling*, kesalahan tersebut dapat ditangani dengan baik tanpa menghentikan eksekusi program.
2. Blok Try, Catch, dan Finally
   - *try* → tempat menuliskan kode yang berpotensi menimbulkan *error*.
   - *catch* → menangkap *exception* yang terjadi dan memberikan penanganan yang sesuai.

- *finally* → bagian yang selalu dieksekusi, baik *error* terjadi maupun tidak, biasanya untuk menutup koneksi atau menampilkan pesan akhir.

3. Custom Exception (Exception Kustom)
   Pengembang dapat membuat *class exception* sendiri yang mewarisi *class Exception*. Dengan cara ini, pesan *error* dapat dibuat lebih spesifik dan sesuai dengan kebutuhan aplikasi, misalnya `LoginException`, `NilaiException`, atau `DatabaseException`.
4. Studi Kasus Implementasi
   Exception Handling dapat diterapkan pada berbagai skenario nyata, seperti:
   - Sistem *Log in* → memberikan pesan *error* ketika *username* atau *password* salah.
   - Validasi Nilai Mahasiswa → menolak *input* nilai yang kurang dari 0 atau lebih dari 100.
5. Keunggulan Exception Handling
   - Menjaga stabilitas program meskipun terjadi *error*.
   - Memberikan pesan *error* yang jelas dan terarah.
   - Memudahkan *debugging* dan pemeliharaan kode.
   - Memungkinkan penggunaan berbagai jenis *exception* sesuai konteks.

Secara keseluruhan, *Exception Handling* merupakan fitur penting dalam OOP PHP yang mendukung pembuatan aplikasi lebih *robust*, aman, dan profesional.

LATIHAN

Untuk memperdalam pemahaman mengenai Exception Handling dalam PHP, kerjakan soal-soal berikut:

1. Latihan Teori

1. Jelaskan perbedaan antara *error* dan *exception* dalam PHP.
2. Sebutkan dan jelaskan fungsi dari blok *try*, *catch*, dan *finally*.
3. Apa keuntungan menggunakan *Custom Exception* dibandingkan *exception* bawaan PHP?

4. Mengapa penting untuk selalu menggunakan *Exception Handling* dalam aplikasi skala besar?

2. Latihan Praktik

1. Buatlah program PHP yang melakukan pembagian dua bilangan. Jika penyebut bernilai nol, program harus melemparkan *exception* dengan pesan "Tidak dapat membagi dengan nol!".
2. Buatlah *class* LoginException dan fungsi *log in* ($username, $password). Jika *username* salah, tampilkan pesan "*Username* tidak ditemukan". Jika *password* salah, tampilkan pesan "*Password* salah". Jika keduanya benar, tampilkan pesan "*Log in* berhasil". Gunakan *Exception Handling* untuk mengatur alurnya.
3. Buatlah *class* BankException untuk validasi transaksi bank. Jika saldo kurang dari jumlah penarikan, lemparkan *exception* dengan pesan "Saldo tidak mencukupi". Jika jumlah penarikan negatif, lemparkan *exception* dengan pesan "Jumlah penarikan tidak valid". Jika valid, kurangi saldo dan tampilkan sisa saldo.
3. Tugas Mandiri
   Modifikasi salah satu program latihan di atas dengan menambahkan blok *finally* yang selalu menampilkan pesan "Transaksi selesai" atau "Proses pengecekan selesai" meskipun terjadi *error*.

BAB 10
Penggunaan Database dalam OOP PHP

DESKRIPSI:

Bab ini membahas bagaimana mengintegrasikan konsep Pemrograman Berorientasi Objek (OOP) dengan *database* dalam PHP. *Database* merupakan komponen penting dalam aplikasi web untuk menyimpan, mengelola, dan mengambil data secara efisien. Dengan menggabungkan OOP dan *database*, pengembang dapat membangun aplikasi yang lebih terstruktur, modular, dan mudah dipelihara. Pada bab ini, akan dipelajari cara membuat koneksi ke *database* menggunakan PDO (PHP Data *Objects*), implementasi *class* untuk operasi CRUD (*Create*, *Read*, *Update*, *Delete*), serta penerapan konsep OOP dalam mengelola data secara lebih rapi dan aman.

TUJUAN PEMBELAJARAN:

Setelah mempelajari bab ini, mahasiswa diharapkan mampu:

1. Memahami konsep dasar integrasi *database* dalam PHP berbasis OOP.
2. Menggunakan PDO sebagai metode koneksi *database* yang aman dan fleksibel.
3. Membuat *class* PHP untuk menangani koneksi *database*.
4. Mengimplementasikan *class* untuk operasi CRUD (*Create*, *Read*, *Update*, *Delete*).
5. Menerapkan prinsip OOP dalam pengelolaan data pada aplikasi berbasis PHP.
6. Mengembangkan studi kasus sederhana dengan menghubungkan PHP OOP ke *database* MySQL.

A. Koneksi Database Menggunakan PDO

PDO (PHP Data *Objects*) adalah ekstensi PHP yang menyediakan antarmuka untuk mengakses berbagai jenis *database* menggunakan metode yang seragam, fleksibel, dan aman. Dengan PDO, kita bisa menghubungkan PHP ke MySQL, PostgreSQL, SQLite, dan banyak *database* lain dengan sedikit perubahan kode. Salah satu keunggulan utama PDO adalah dukungan untuk *Prepared Statements*, yang dapat membantu mencegah serangan SQL *Injection*.

Contoh kode: membuat koneksi ke database MySQL dengan PDO:

```
<?php
class Database {
    private $host = "localhost";
    private $db_name = "kampus";
    private $username = "root";
    private $password = "";
    private $conn;

    // Fungsi untuk koneksi ke database
    public function connect() {
        $this->conn = null;
        try {
            $this->conn = new PDO(
                    "mysql:host=" . $this->host .
";dbname=" . $this->db_name,
                $this->username,
                $this->password
            );
                                        $this->conn-
>setAttribute(PDO::ATTR_ERRMODE,
PDO::ERRMODE_EXCEPTION);
            echo "Koneksi berhasil!";
        } catch (PDOException $e) {
                 echo "Koneksi gagal: " . $e-
>getMessage();
        }
        return $this->conn;
    }
}
```

```
// Pengujian koneksi
$db = new Database();
$koneksi = $db->connect();
?>
```

Output jika berhasil koneksi ke *database*:

```
Koneksi berhasil!
```

Output jika gagal koneksi *database*:

```
Koneksi gagal: SQLSTATE[HY000] [1045] Access denied for user 'root'@'localhost'
```

Penjelasan kode:

1. Class Database
   - Menyimpan informasi koneksi *database* seperti *host*, nama *database*, *username*, dan *password*.
   - Menyediakan *method* `connect()` untuk membuat koneksi menggunakan PDO.
2. Try-Catch Block
   - `try` digunakan untuk mencoba membuat koneksi.
   - Jika gagal, `catch` menangkap *error* dengan pesan yang jelas dari `PDOException`.
3. `setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION)`
   - Digunakan agar *error* ditampilkan sebagai *exception*, sehingga lebih mudah ditangani.
4. Pengujian koneksi
   - Objek `$db` dibuat dari class `Database`.
   - Fungsi `connect()` dipanggil untuk menguji apakah koneksi berhasil atau tidak.

B. Membuat Class CRUD dengan PDO

CRUD adalah singkatan dari *Create*, *Read*, *Update*, dan *Delete*, yang merupakan operasi dasar dalam mengelola data pada *database*. Dalam OOP PHP, operasi CRUD dapat diimplementasikan menggunakan *class* sehingga kode menjadi lebih terstruktur, mudah dipelihara, dan dapat digunakan kembali (*reusable*). Dengan bantuan PDO, operasi CRUD juga menjadi lebih aman karena mendukung *prepared statements* untuk mencegah serangan SQL *Injection*.

Contoh kode: implementasi *class* CRUD mahasiswa:

```
<?php
class Mahasiswa {
    private $conn;
    private $table = "mahasiswa";

    // Properti
    public $id;
    public $nama;
    public $nim;
    public $jurusan;

    // Konstruktor: menerima koneksi database
    public function __construct($db) {
        $this->conn = $db;
    }

    // CREATE (Menambahkan data baru)
    public function create() {
        $query = "INSERT INTO " . $this->table . "
(nama,   nim,   jurusan)   VALUES   (:nama,   :nim,
:jurusan)";
        $stmt = $this->conn->prepare($query);

        // Binding parameter
        $stmt->bindParam(":nama", $this->nama);
        $stmt->bindParam(":nim", $this->nim);
             $stmt->bindParam(":jurusan",   $this-
>jurusan);

        if ($stmt->execute()) {
            return true;
        }
        return false;
    }

    // READ (Mengambil semua data)
    public function read() {
        $query = "SELECT * FROM " . $this->table;
        $stmt = $this->conn->prepare($query);
        $stmt->execute();
```

```
        return $stmt;
    }

    // UPDATE (Mengubah data berdasarkan ID)
    public function update() {
        $query = "UPDATE " . $this->table . " SET
nama = :nama, nim = :nim, jurusan = :jurusan WHERE
id = :id";
        $stmt = $this->conn->prepare($query);

        $stmt->bindParam(":nama", $this->nama);
        $stmt->bindParam(":nim", $this->nim);
             $stmt->bindParam(":jurusan",   $this-
>jurusan);
        $stmt->bindParam(":id", $this->id);

        if ($stmt->execute()) {
            return true;
        }
        return false;
    }

    // DELETE (Menghapus data berdasarkan ID)
    public function delete() {
        $query = "DELETE FROM " . $this->table . "
WHERE id = :id";
        $stmt = $this->conn->prepare($query);

        $stmt->bindParam(":id", $this->id);

        if ($stmt->execute()) {
            return true;
        }
        return false;
    }
}
?>
```

Contoh penggunaan:

```
<?php
require_once "Database.php"; // File koneksi PDO
```

```
// Koneksi ke database
$database = new Database();
$db = $database->connect();

// Objek Mahasiswa
$mhs = new Mahasiswa($db);

// CREATE
$mhs->nama = "Budi Santoso";
$mhs->nim = "2025001";
$mhs->jurusan = "Informatika";
if ($mhs->create()) {
    echo "Data berhasil ditambahkan.<br>";
}

// READ
$result = $mhs->read();
while ($row = $result->fetch(PDO::FETCH_ASSOC)) {
    extract($row);
    echo "ID: $id - Nama: $nama - NIM: $nim -
Jurusan: $jurusan <br>";
}

// UPDATE
$mhs->id = 1;
$mhs->nama = "Andi Wijaya";
$mhs->nim = "2025002";
$mhs->jurusan = "Sistem Informasi";
if ($mhs->update()) {
    echo "Data berhasil diubah.<br>";
}

// DELETE
$mhs->id = 2;
if ($mhs->delete()) {
    echo "Data berhasil dihapus.<br>";
}
?>
```

Penjelasan dari kode program di atas Implementasi *class* `Mahasiswa` dengan operasi CRUD menggunakan PDO menunjukkan bagaimana konsep OOP dapat digunakan untuk

mengelola *database* secara lebih terstruktur. Dengan pendekatan ini, setiap operasi seperti menambahkan data (*Create*), mengambil data (*Read*), memperbarui data (*Update*), dan menghapus data (*Delete*) ditangani melalui *method* khusus sehingga kode menjadi lebih terorganisir dan mudah dipelihara. Penggunaan *prepared statements* dengan `bindParam` juga meningkatkan keamanan program karena dapat mencegah serangan SQL *Injection*, yang sering menjadi celah pada aplikasi berbasis *database*. Selain itu, pemisahan logika antara koneksi *database* (class `Database`) dan pengelolaan data mahasiswa (*class* `Mahasiswa`) mencerminkan prinsip *separation of concerns*, sehingga aplikasi lebih fleksibel untuk dikembangkan. Dengan adanya *class* ini, developer dapat dengan mudah memperluas fungsionalitas, misalnya menambahkan fitur pencarian atau filter data tanpa harus menulis ulang logika dasar CRUD. Secara keseluruhan, pendekatan ini menjadikan aplikasi lebih modular, aman, efisien, dan sesuai standar pengembangan perangkat lunak modern.

C. Studi Kasus: Sistem Manajemen Mahasiswa dengan OOP dan Database

Sebuah kampus membutuhkan sistem sederhana untuk mengelola data mahasiswa. Sistem tersebut harus bisa melakukan operasi CRUD (*Create*, *Read*, *Update*, *Delete*) terhadap tabel mahasiswa di *database*. Dengan menerapkan OOP dan PDO, sistem ini dapat dibuat lebih modular, mudah dipelihara, dan aman dari serangan SQL *Injection*. Berikut langkah-langkahnya:

1. Struktur Database (MySQL)

```
CREATE DATABASE kampus;

USE kampus;

CREATE TABLE mahasiswa (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nama VARCHAR(100) NOT NULL,
    nim VARCHAR(20) NOT NULL UNIQUE,
    jurusan VARCHAR(50) NOT NULL
);
```

2. **Class Database untuk koneksi (Database.php)**

```
<?php
class Database {
    private $host = "localhost";
    private $db_name = "kampus";
    private $username = "root";
    private $password = "";
    private $conn;

    public function connect() {
        $this->conn = null;
        try {
            $this->conn = new PDO(
                  "mysql:host=" . $this->host .
";dbname=" . $this->db_name,
                $this->username,
                $this->password
            );
                                        $this->conn-
>setAttribute(PDO::ATTR_ERRMODE,
PDO::ERRMODE_EXCEPTION);
        } catch (PDOException $e) {
                echo "Koneksi gagal: " . $e-
>getMessage();
        }
        return $this->conn;
    }
}
?>
```

3. **Class Mahasiswa untuk CRUD (Mahasiswa.php)**

```
<?php
class Mahasiswa {
    private $conn;
    private $table = "mahasiswa";

    public $id;
    public $nama;
    public $nim;
    public $jurusan;

    public function __construct($db) {
```

```
        $this->conn = $db;
    }

    // CREATE
    public function create() {
        $query = "INSERT INTO " . $this->table
. " (nama, nim, jurusan) VALUES (:nama, :nim,
:jurusan)";
        $stmt = $this->conn->prepare($query);

        $stmt->bindParam(":nama", $this->nama);
        $stmt->bindParam(":nim", $this->nim);
           $stmt->bindParam(":jurusan",  $this-
>jurusan);

        return $stmt->execute();
    }

    // READ
    public function read() {
       $query = "SELECT * FROM " . $this->table;
        $stmt = $this->conn->prepare($query);
        $stmt->execute();
        return $stmt;
    }

    // UPDATE
    public function update() {
        $query = "UPDATE " . $this->table . "
SET nama=:nama, nim=:nim, jurusan=:jurusan WHERE
id=:id";
        $stmt = $this->conn->prepare($query);

        $stmt->bindParam(":nama", $this->nama);
        $stmt->bindParam(":nim", $this->nim);
           $stmt->bindParam(":jurusan",  $this-
>jurusan);
        $stmt->bindParam(":id", $this->id);

        return $stmt->execute();
    }
```

```
    // DELETE
    public function delete() {
        $query = "DELETE FROM " . $this->table
. " WHERE id=:id";
        $stmt = $this->conn->prepare($query);

        $stmt->bindParam(":id", $this->id);

        return $stmt->execute();
    }
}
?>
```

4. File Utama (index.php – Contoh Penggunaan CRUD)

```
<?php
require_once "Database.php";
require_once "Mahasiswa.php";

// Koneksi
$database = new Database();
$db = $database->connect();

// Objek Mahasiswa
$mhs = new Mahasiswa($db);

// CREATE
$mhs->nama = "Dzimar Rauhillah";
$mhs->nim = "20251001";
$mhs->jurusan = "Teknik Informatika";
if ($mhs->create()) {
    echo "Data berhasil ditambahkan.<br>";
}

// READ
echo "<h3>Daftar Mahasiswa:</h3>";
$result = $mhs->read();
while ($row = $result->fetch(PDO::FETCH_ASSOC))
{
    extract($row);
    echo "ID: $id | Nama: $nama | NIM: $nim |
Jurusan: $jurusan <br>";
```

```
}

// UPDATE
$mhs->id = 1;
$mhs->nama = "Ghifari Dzikrullah";
$mhs->nim = "20251002";
$mhs->jurusan = "Sistem Informasi";
if ($mhs->update()) {
    echo "Data berhasil diperbarui.<br>";
}

// DELETE
$mhs->id = 2;
if ($mhs->delete()) {
    echo "Data berhasil dihapus.<br>";
}
?>
```

Studi kasus sistem manajemen mahasiswa ini menunjukkan bagaimana konsep OOP dan PDO bekerja sama untuk membangun aplikasi berbasis *database* yang rapi, aman, dan mudah dipelihara. Dengan memisahkan logika koneksi (*class* `Database`) dan operasi CRUD (*class* `Mahasiswa`), kode menjadi lebih modular dan mengikuti prinsip *separation of concerns*. Implementasi CRUD melalui *method* `create()`, `read()`, `update()`, dan `delete()` memudahkan pengelolaan data mahasiswa tanpa perlu menulis *query* SQL berulang kali di file utama. Selain itu, penggunaan *prepared statements* menjadikan aplikasi lebih aman dari serangan SQL *Injection*. Dengan struktur ini, sistem dapat dengan mudah dikembangkan, misalnya dengan menambahkan fitur pencarian, filter data, atau autentikasi pengguna.

RANGKUMAN

Pada bab ini, kita telah mempelajari bagaimana OOP dalam PHP dapat diintegrasikan dengan *database* untuk membangun aplikasi yang lebih terstruktur, aman, dan mudah dipelihara. Dengan memanfaatkan PDO (PHP Data *Objects*), koneksi ke *database*

menjadi lebih fleksibel dan aman melalui penggunaan *prepared statements*. Konsep *class* dan *object* diterapkan pada pembuatan *class Database* untuk mengatur koneksi, serta *class* Mahasiswa untuk mengelola operasi CRUD (*Create*, *Read*, *Update*, *Delete*). Pemisahan ini membuat kode lebih modular, sehingga mudah dikembangkan dan dikelola sesuai kebutuhan. Melalui studi kasus Sistem Manajemen Mahasiswa, kita melihat bagaimana OOP membantu menjaga keteraturan kode dan memungkinkan pengembangan fitur tambahan di masa depan, seperti pencarian, filter, dan autentikasi. Dengan pendekatan ini, pemrograman *database* dalam PHP menjadi lebih efisien, terorganisir, dan aman.

LATIHAN

Untuk menguji pemahaman Anda mengenai penggunaan *database* dalam OOP PHP, kerjakan latihan berikut:

1. Latihan Teori

1. Jelaskan apa itu PDO dalam PHP dan sebutkan keunggulannya dibandingkan metode koneksi *database* lama (mysql_*).
2. Mengapa penggunaan *prepared statements* sangat penting dalam aplikasi berbasis *database*?
3. Apa perbedaan utama antara *class Database* dan *class* Mahasiswa dalam implementasi OOP pada bab ini?
4. Sebutkan manfaat penggunaan prinsip *separation of concerns* dalam pengelolaan kode berbasis OOP dan *database*.
5. Bagaimana konsep *inheritance* bisa diterapkan dalam pembuatan sistem manajemen data dengan OOP PHP?

2. Latihan Praktik

1. Buatlah *class* Buku yang memiliki operasi CRUD menggunakan PDO untuk tabel buku dengan atribut id, judul, penulis, dan tahun_terbit.
2. Tambahkan fitur pencarian data mahasiswa berdasarkan nama pada *class* Mahasiswa yang sudah dibuat.

3. Modifikasi *class* Mahasiswa agar dapat menampilkan daftar mahasiswa dengan filter berdasarkan program studi (prodi).
4. Buat sebuah *class User* dengan atribut id, *username*, *password*, dan implementasikan fungsi register dan *log in* menggunakan PDO dengan konsep OOP.
5. Integrasikan *class* Mahasiswa dengan halaman web sederhana (HTML + PHP) untuk menampilkan data mahasiswa dalam bentuk tabel di browser.

BAB 11
Desain dan Arsitektur dalam OOP PHP

DESKRIPSI:

Bab ini membahas bagaimana konsep desain dan arsitektur diterapkan dalam pengembangan perangkat lunak berbasis OOP menggunakan PHP. Tidak hanya sekadar membuat *class* dan *object*, tetapi juga bagaimana menyusun struktur kode agar lebih rapi, mudah dikelola, serta dapat dikembangkan dalam jangka panjang. Materi akan mencakup prinsip-prinsip desain OOP seperti SOLID *principles*, pola desain (*design patterns*), serta arsitektur aplikasi seperti MVC (*Model-View-Controller*) yang umum digunakan dalam *framework* PHP modern (misalnya *Laravel*, *CodeIgniter*, atau *Symfony*). Dengan memahami desain dan arsitektur, developer dapat membangun aplikasi yang *modular*, *scalable*, *reusable*, dan *maintainable*.

TUJUAN PEMBELAJARAN:

Setelah mempelajari bab ini, mahasiswa diharapkan mampu:

1. Menjelaskan prinsip-prinsip dasar desain dalam OOP, termasuk SOLID *principles*.
2. Memahami konsep *design patterns* (misalnya *Singleton*, *Factory*, dan *Observer*) serta penerapannya dalam PHP.
3. Menguraikan arsitektur MVC (*Model-View-Controller*) dan perannya dalam pemisahan logika program.
4. Menerapkan desain dan arsitektur OOP pada studi kasus aplikasi sederhana.
5. Mengevaluasi kelebihan dan kekurangan penggunaan desain dan arsitektur tertentu dalam pengembangan aplikasi PHP.

A. Prinsip-prinsip Desain OOP (SOLID Principles)

Dalam pengembangan perangkat lunak berbasis OOP, salah satu tantangan terbesar adalah menjaga agar kode tetap mudah dibaca, fleksibel, dan dapat diperluas tanpa harus merombak keseluruhan sistem. Untuk mencapai hal tersebut, para ahli perangkat lunak memperkenalkan prinsip desain yang dikenal dengan SOLID *Principles*.

SOLID adalah singkatan dari:

1. S (*Single Responsibility Principle*) → Setiap *class* hanya boleh memiliki satu tanggung jawab utama.
2. O (*Open*/*Closed Principle*) → *Class* harus terbuka untuk ekstensi namun tertutup untuk modifikasi.
3. L (*Liskov Substitution Principle*) → Objek dari *subclass* harus dapat menggantikan *superclass* tanpa mengubah keakuratan program.
4. I (*Interface Segregation Principle*) → *Interface* sebaiknya dibuat spesifik dan tidak memaksa *class* mengimplementasikan *method* yang tidak diperlukan.
5. D (*Dependency Inversion Principle*) → *Class* sebaiknya bergantung pada abstraksi (*interface*/*abstract class*), bukan pada implementasi konkret.

Contoh penerapan sederhana dalam PHP:

```
<?php
// S: Single Responsibility Principle
class LaporanPDF {
    public function generate($data) {
        echo "Membuat laporan PDF dari data: " .
$data;
    }
}

class KirimEmail {
    public function send($email, $message) {
         echo "Mengirim email ke $email dengan
pesan: $message";
    }
}
```

```
// O: Open/Closed Principle dengan Polimorfisme
interface Laporan {
    public function cetak($data);
}

class LaporanExcel implements Laporan {
    public function cetak($data) {
         echo "Mencetak laporan Excel dari data:
$data";
    }
}

class LaporanWord implements Laporan {
    public function cetak($data) {
          echo "Mencetak laporan Word dari data:
$data";
    }
}

// D: Dependency Inversion Principle
class LaporanService {
    private $laporan;

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

    public function prosesCetak($data) {
        $this->laporan->cetak($data);
    }
}

// Eksekusi
$laporan = new LaporanService(new LaporanExcel());
$laporan->prosesCetak("Data Mahasiswa");
?>
```

Kode di atas menunjukkan bagaimana SOLID *principles* dapat diterapkan dalam PHP:

1. SRP diterapkan dengan memisahkan tugas LaporanPDF (hanya membuat laporan) dan KirimEmail (hanya mengirim email),

sehingga masing-masing *class* punya tanggung jawab spesifik.

2. OCP diterapkan dengan membuat *interface* Laporan, sehingga jika kita ingin menambahkan format baru (misalnya LaporanCSV), kita tidak perlu mengubah *class* yang sudah ada, cukup menambah *class* baru.
3. DIP diterapkan pada LaporanService yang bergantung pada *interface* Laporan (abstraksi), bukan pada implementasi konkret (LaporanExcel atau LaporanWord).

Dengan penerapan prinsip ini, kode menjadi lebih modular, fleksibel, dan mudah dikembangkan tanpa harus mengubah logika yang sudah ada.

B. Design Patterns dalam OOP PHP

Design Patterns adalah solusi umum yang sudah teruji untuk menyelesaikan permasalahan yang sering muncul dalam desain perangkat lunak. Dalam OOP PHP, *design patterns* membantu *developer* menulis kode yang lebih terstruktur, *reusable*, dan *maintainable*. Ada banyak jenis *design patterns*, namun pada bab ini kita akan membahas tiga yang paling sering digunakan dalam pengembangan aplikasi PHP:

1. Singleton Pattern

Singleton digunakan untuk memastikan hanya ada satu *instance* dari sebuah *class* sepanjang siklus hidup aplikasi. Biasanya digunakan untuk koneksi *database*. Kelebihan *singleton* yaitu menghemat *resource*, konsisten pada penggunaan *instance* dan kekurangannya yaitu menyulitkan pengujian (unit testing) jika terlalu bergantung pada *Singleton*. Contoh:

```
<?php
class Database {
    private static $instance = null;
    private $connection;

    private function __construct() {
                $this->connection    =    new
PDO("mysql:host=localhost;dbname=test", "root",
"");
    }
```

```
    public static function getInstance() {
        if (self::$instance == null) {
            self::$instance = new Database();
        }
        return self::$instance;
    }

    public function getConnection() {
        return $this->connection;
    }
}

// Penggunaan
$db1 = Database::getInstance();
$db2 = Database::getInstance();

var_dump($db1 === $db2); // true, hanya ada satu
instance
?>
```

2. Factory Pattern

Factory digunakan untuk menciptakan *object* tanpa harus menentukan *class* secara eksplisit. Cocok jika kita punya banyak turunan *class* dengan perilaku berbeda. Kelebihan *factory pattern*, yaitu memudahkan pembuatan *object* secara fleksibel dan kekurangannya, yaitu Jika tipe *object* terlalu banyak, kode *Factory* bisa menjadi kompleks. Contoh:

```
<?php
interface Laporan {
    public function cetak();
}

class LaporanPDF implements Laporan {
    public function cetak() {
        return "Mencetak laporan PDF";
    }
}

class LaporanExcel implements Laporan {
    public function cetak() {
        return "Mencetak laporan Excel";
```

```
    }
}

class LaporanFactory {
    public static function createLaporan($tipe)
{
        switch($tipe) {
            case 'pdf':
                return new LaporanPDF();
            case 'excel':
                return new LaporanExcel();
            default:
               throw new Exception("Tipe laporan
tidak dikenal");
        }
    }
}

// Penggunaan
$laporan                                        =
LaporanFactory::createLaporan('pdf');
echo $laporan->cetak();
?>
```

3. Observer Pattern

Observer digunakan ketika satu *object* (*subject*) ingin memberi tahu *object* lain (*observer*) saat terjadi perubahan. Kelebihan *observer pattern*, yaitu memudahkan komunikasi antar objek dan kekurangannya, yaitu bisa sulit dikelola jika jumlah *observer* terlalu banyak. Contoh:

```
<?php
interface Observer {
    public function update($message);
}

class User implements Observer {
    private $name;

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

```
    public function update($message) {
        echo $this->name . " menerima notifikasi:
" . $message . "\n";
    }
}

class Notifikasi {
    private $observers = [];

     public function tambahObserver(Observer
$observer) {
        $this->observers[] = $observer;
    }

    public function kirimNotifikasi($message) {
        foreach ($this->observers as $observer)
{
            $observer->update($message);
        }
    }
}

// Penggunaan
$notifikasi = new Notifikasi();
$user1 = new User("Mizard");
$user2 = new User("Dzimar");

$notifikasi->tambahObserver($user1);
$notifikasi->tambahObserver($user2);

$notifikasi->kirimNotifikasi("Ada update sistem
terbaru!");
?>
```

Ketiga pola desain di atas memperlihatkan bagaimana OOP PHP bisa lebih terstruktur dengan bantuan *design patterns*. *Singleton* menjaga efisiensi *resource*, Factory menyederhanakan proses pembuatan *object*, sementara *Observer* memfasilitasi komunikasi antar objek. Dengan memahami *design patterns*, *developer* dapat

membangun aplikasi PHP yang lebih modular, *scalable*, dan *reusable*.

C. Arsitektur MVC dalam PHP

MVC (*Model-View-Controller*) adalah salah satu arsitektur perangkat lunak yang paling populer dalam pengembangan aplikasi berbasis OOP PHP. Konsep ini membagi aplikasi menjadi tiga komponen utama:

1. *Model* → menangani logika bisnis dan interaksi dengan *database*.
2. *View* → bertugas menampilkan data kepada pengguna dalam bentuk antarmuka (UI).
3. *Controller* → sebagai penghubung antara Model dan View, mengatur alur logika aplikasi berdasarkan *input* pengguna.

Dengan pemisahan ini, kode menjadi lebih modular, terorganisir, dan mudah dipelihara, karena setiap bagian memiliki tanggung jawab spesifik. Berikut Struktur Dasar MVC:

```
project/
|-- index.php        # Front Controller
|-- app/
|   |-- controllers/
|   |   └── MahasiswaController.php
|   |-- models/
|   |   └── Mahasiswa.php
|   |-- views/
|   |   └── mahasiswa_view.php
```

Contoh implementasi MVC sederhana:

1. Model (Mahasiswa.php)

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

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

    public function getNama() {
```

```
        return $this->nama;
    }

    public function getNim() {
        return $this->nim;
    }
}
?>
```

2. View (mahasiswa_view.php)

```
<?php
class MahasiswaView {
    public function tampilkan($mahasiswa) {
        echo "Nama: " . $mahasiswa->getNama() .
"<br>";
         echo "NIM : " . $mahasiswa->getNim() .
"<br>";
    }
}
?>
```

3. Controller (MahasiswaController.php)

```
<?php
require_once "models/Mahasiswa.php";
require_once "views/mahasiswa_view.php";

class MahasiswaController {
    public function detailMahasiswa() {
          $mahasiswa = new Mahasiswa("Mizard",
"123456");
        $view = new MahasiswaView();
        $view->tampilkan($mahasiswa);
    }
}
?>
```

Contoh di atas memperlihatkan bagaimana MVC membantu memisahkan tanggung jawab dalam aplikasi:

1. *Model* fokus pada data dan logika bisnis (*class* Mahasiswa).
2. *View* fokus pada tampilan (*class* MahasiswaView).
3. *Controller* menghubungkan Model dan View (*class* MahasiswaController).

Dengan pola ini, perubahan di tampilan (View) tidak akan memengaruhi logika bisnis (Model), dan sebaliknya. Arsitektur MVC sangat berguna pada aplikasi berskala besar karena membuat kode lebih *modular*, *maintainable*, *scalable*, dan mudah diuji. *Framework* PHP populer seperti *Laravel*, *CodeIgniter*, dan *Symfony* juga menggunakan pendekatan MVC sebagai fondasi utamanya.

RANGKUMAN

Pada bab ini, kita telah mempelajari bagaimana desain dan arsitektur dalam OOP PHP membantu membangun aplikasi yang lebih terstruktur, modular, dan mudah dipelihara.

1. SOLID *Principles* memberikan panduan agar setiap class memiliki tanggung jawab tunggal, dapat diperluas tanpa harus dimodifikasi, dapat saling menggantikan dengan benar, menggunakan *interface* yang sesuai, serta bergantung pada abstraksi, bukan implementasi konkret. Prinsip ini menjadikan kode lebih fleksibel, *reusable*, dan *scalable*.
2. *Design Patterns* seperti *Singleton*, *Factory*, dan *Observer* merupakan solusi umum yang terbukti efektif dalam menyelesaikan permasalahan desain perangkat lunak. Singleton menjaga hanya ada satu *instance class*, *Factory* mempermudah pembuatan *object* tanpa menentukan *class* secara eksplisit, sementara *Observer* memfasilitasi komunikasi antar *object*.
3. Arsitektur MVC (*Model-View-Controller*) memisahkan aplikasi menjadi tiga komponen utama: *Model* (data dan logika bisnis), *View* (tampilan), dan *Controller* (pengatur alur). Dengan pemisahan ini, kode menjadi lebih terorganisir, sehingga perubahan pada satu bagian tidak mengganggu bagian lainnya.

Secara keseluruhan, pemahaman terhadap prinsip desain, *design patterns*, dan arsitektur MVC akan membantu developer membangun aplikasi berbasis OOP PHP yang lebih profesional, *maintainable*, dan siap dikembangkan untuk kebutuhan skala kecil maupun besar.

LATIHAN

Untuk memperdalam pemahaman Anda mengenai desain dan arsitektur dalam OOP PHP, kerjakan latihan berikut:

1. Latihan Teori

1. Jelaskan secara singkat masing-masing prinsip dalam SOLID dan berikan contoh aplikasinya dalam PHP.
2. Apa perbedaan antara *Singleton Pattern* dan *Factory Pattern* dalam konteks OOP PHP?
3. Mengapa arsitektur MVC dianggap lebih baik dibandingkan dengan pendekatan kode prosedural?
4. Sebutkan kelebihan dan kelemahan penggunaan *Observer Pattern* dalam aplikasi berskala besar.
5. Mengapa pemisahan tanggung jawab antara *Model*, *View*, dan *Controller* sangat penting dalam arsitektur aplikasi?

2. Latihan Praktik

1. Buatlah implementasi sederhana *Singleton Pattern* untuk koneksi *database* menggunakan PDO.
2. Kembangkan *Factory Pattern* untuk membuat *object* kendaraan (Mobil, Motor) dengan *method* jalan().
3. Implementasikan *Observer Pattern* untuk sistem notifikasi ketika ada *user* baru yang terdaftar.
4. Buatlah contoh aplikasi sederhana menggunakan MVC dengan data Produk (id, nama, harga) yang dapat ditampilkan di halaman web.

Ubah contoh aplikasi MVC di atas agar mendukung penambahan produk baru melalui *form* sederhana.

BAB 12
Studi Kasus dan Proyek Akhir

DESKRIPSI:

Bab ini merupakan bagian penutup dari buku ajar, yang berfokus pada penerapan seluruh konsep Pemrograman Berorientasi Objek (OOP) dalam PHP ke dalam sebuah studi kasus nyata. Mahasiswa tidak hanya mempelajari teori dan contoh sederhana, tetapi juga diajak untuk mengimplementasikan prinsip OOP, penggunaan *database*, desain arsitektur, serta penerapan *best practice* ke dalam sebuah proyek akhir aplikasi berbasis web. Dengan demikian, mahasiswa memperoleh pengalaman langsung dalam membangun aplikasi yang terstruktur, modular, dan dapat digunakan secara praktis.

TUJUAN PEMBELAJARAN:

Setelah mempelajari dan mengerjakan proyek akhir pada bab ini, mahasiswa diharapkan mampu:

1. Mengintegrasikan konsep *class*, *object*, enkapsulasi, *inheritance*, dan polimorfisme dalam aplikasi nyata.
2. Menerapkan prinsip SOLID dan *design patterns* untuk meningkatkan kualitas desain aplikasi.
3. Menggunakan PDO dan *database* dalam implementasi CRUD yang aman dan efisien.
4. Membangun aplikasi dengan arsitektur MVC sederhana menggunakan PHP.
5. Menyelesaikan proyek akhir berupa aplikasi web sederhana berbasis OOP PHP yang sesuai kebutuhan pengguna.
6. Menunjukkan kemampuan analisis, perancangan, implementasi, serta pengujian aplikasi.

A. Studi Kasus: Sistem Manajemen Perpustakaan Berbasis OOP PHP

Studi kasus ini bertujuan untuk mengintegrasikan seluruh konsep OOP PHP yang telah dipelajari ke dalam sebuah aplikasi manajemen perpustakaan. Aplikasi ini akan memiliki fitur dasar seperti manajemen buku, manajemen anggota, dan transaksi peminjaman/pengembalian. Selain itu, aplikasi akan menggunakan *database* dengan PDO, menerapkan enkapsulasi, *inheritance*, polimorfisme, serta menggunakan arsitektur MVC agar kode tetap modular dan mudah dikelola.

1. Fitur Utama Sistem

Fitur utama dalam Sistem Manajemen Perpustakaan berbasis OOP dibagi beberapa fitur yaitu:

a. Manajemen Buku
   - Tambah, lihat, ubah, dan hapus data buku.
   - Informasi buku: ID, judul, penulis, penerbit, tahun terbit.

b. Manajemen Anggota
   - Tambah, lihat, ubah, dan hapus data anggota.
   - Informasi anggota: ID, nama, alamat, email, nomor telepon.

c. Transaksi Peminjaman dan Pengembalian
   - Catat transaksi peminjaman dengan tanggal pinjam dan tanggal kembali.
   - *Update* status buku ketika dipinjam/dikembalikan.

d. Laporan
   - Menampilkan daftar buku yang tersedia dan yang sedang dipinjam.
   - Menampilkan riwayat transaksi anggota.

2. Struktur Kode (MVC Sederhana)

Struktur kode untuk Sistem Manajemen Perpustakaan berbasis OOP adalah sebagai berikut:

```
perpustakaan/
|-- index.php
|-- app/
|   |-- controllers/
|   |   ├── BukuController.php
|   |   ├── AnggotaController.php
```

```
|   |   └── TransaksiController.php
|   |-- models/
|   |   ├── Buku.php
|   |   ├── Anggota.php
|   |   └── Transaksi.php
|   |-- views/
|   |   ├── buku_view.php
|   |   ├── anggota_view.php
|   |   └── transaksi_view.php
|-- config/
|   └── Database.php
```

3. Implementasi Kode

Untuk implementasi kode dibagi dalam beberapa bagian yaitu:

a. config/Database.php

```
<?php
class Database {
    private static $instance = null;
    private $connection;

    private function __construct() {
                $this->connection    =    new
PDO("mysql:host=localhost;dbname=perpustakaa
n", "root", "");
                            $this->connection-
>setAttribute(PDO::ATTR_ERRMODE,
PDO::ERRMODE_EXCEPTION);
    }

    public static function getInstance() {
        if (self::$instance == null) {
            self::$instance = new Database();
        }
        return self::$instance->connection;
    }
}
?>
```

b. models/Buku.php

```
<?php
class Buku {
    private $db;
```

```
    public function __construct() {
        $this->db = Database::getInstance();
    }

      public function tambahBuku($judul,
$penulis, $penerbit, $tahun) {
         $sql = "INSERT INTO buku (judul,
penulis, penerbit, tahun) VALUES (?, ?, ?,
?)";
        $stmt = $this->db->prepare($sql);
          $stmt->execute([$judul, $penulis,
$penerbit, $tahun]);
    }

    public function getAllBuku() {
         $stmt = $this->db->query("SELECT *
FROM buku");
                          return       $stmt-
>fetchAll(PDO::FETCH_ASSOC);
    }
}
?>
```

c. controllers/BukuController.php

```
<?php
require_once "app/models/Buku.php";
require_once "app/views/buku_view.php";

class BukuController {
    private $model;

    public function __construct() {
        $this->model = new Buku();
    }

    public function index() {
        $data = $this->model->getAllBuku();
        $view = new BukuView();
        $view->tampilkan($data);
    }
}
?>
```

d. views/buku_view.php

```
<?php
class BukuView {
    public function tampilkan($data) {
        echo "<h2>Daftar Buku</h2>";
        echo "<table border='1'>
                <tr><th>ID</th><th>Judul</th
><th>Penulis</th><th>Penerbit</th><th>Tahun<
/th></tr>";
        foreach ($data as $row) {
            echo "<tr>
                    <td>{$row['id']}</td>
                    <td>{$row['judul']}</td>
                  <td>{$row['penulis']}</td>
                 <td>{$row['penerbit']}</td>
                    <td>{$row['tahun']}</td>
                 </tr>";
        }
        echo "</table>";
    }
}
?>
```

e. index.php

```
<?php
require_once "config/Database.php";
require_once
"app/controllers/BukuController.php";

$controller = new BukuController();
$controller->index();
?>
```

Studi kasus ini menggambarkan penerapan nyata OOP dalam PHP untuk membangun aplikasi Sistem Manajemen Perpustakaan. Dengan MVC, logika bisnis (Model), pengendali alur (*Controller*), dan tampilan (View) dipisahkan, sehingga aplikasi lebih modular dan mudah dikelola. PDO digunakan agar koneksi *database* lebih aman dengan *prepared statements*. Konsep enkapsulasi diterapkan pada model dengan menyembunyikan detail *query* dari bagian lain program, sementara polimorfisme dan *inheritance* dapat diterapkan pada pengembangan fitur lanjutan

seperti laporan. Studi kasus ini dapat dijadikan dasar untuk proyek akhir mahasiswa dalam mengembangkan aplikasi web yang lebih kompleks dengan PHP dan OOP.

B. Proyek Akhir: Aplikasi Web Sederhana Berbasis OOP PHP

Proyek akhir ini merupakan tugas komprehensif untuk mahasiswa dalam menerapkan seluruh konsep Pemrograman Berorientasi Objek (OOP) dalam PHP yang telah dipelajari. Aplikasi yang dibangun adalah Sistem Manajemen Toko Online Sederhana, dengan fitur CRUD produk, manajemen *user* (admin & *customer*), transaksi pembelian, serta integrasi *database*. Proyek ini juga mengadopsi pola MVC (*Model-View-Controller*) agar arsitektur aplikasi tetap terstruktur, modular, dan mudah dikembangkan.

1. Fitur Utama

Adapun fitur utama dalam aplikasi yang dibangun yaitu Sistem Manajemen Toko Online Sederhana yaitu:

a. Manajemen Produk
   - Admin dapat menambah, mengedit, menghapus, dan menampilkan daftar produk.
   - Setiap produk memiliki atribut: ID, nama, harga, stok, dan deskripsi.

b. Manajemen Pengguna
   - Sistem *log in* & register untuk admin dan *customer*.
   - Admin memiliki akses penuh, sedangkan *customer* hanya bisa melihat produk dan melakukan pembelian.

c. Transaksi Pembelian
   - *Customer* dapat menambahkan produk ke keranjang belanja (*cart*).
   - Sistem menyimpan transaksi ke *database* dengan detail pembelian.

d. Laporan Penjualan
   - Admin dapat melihat riwayat transaksi untuk analisis sederhana.

2. Struktur Folder

Berikut struktur folder untuk Sistem Manajemen Toko Online

```
toko_online/
```

```
|-- index.php
|-- config/
|   └── Database.php
|-- app/
|   |-- controllers/
|   |   ├── ProdukController.php
|   |   ├── UserController.php
|   |   └── TransaksiController.php
|   |-- models/
|   |   ├── Produk.php
|   |   ├── User.php
|   |   └── Transaksi.php
|   |-- views/
|   |   ├── produk_view.php
|   |   ├── user_view.php
|   |   └── transaksi_view.php
```

3. Implementasi Kode

Untuk implementasi kode dalam aplikasi yang dibangun, yaitu Sistem Manajemen Toko Online Sederhana:

a. config/Database.php

```
<?php
class Database {
    private static $instance = null;
    private $conn;

    private function __construct() {
                    $this->conn     =     new
PDO("mysql:host=localhost;dbname=toko_online
", "root", "");
                                    $this->conn-
>setAttribute(PDO::ATTR_ERRMODE,
PDO::ERRMODE_EXCEPTION);
    }

    public static function getInstance() {
        if (self::$instance == null) {
            self::$instance = new Database();
        }
        return self::$instance->conn;
    }
}
```

```
?>
```

b. models/Produk.php

```
<?php
class Produk {
    private $db;

    public function __construct() {
        $this->db = Database::getInstance();
    }

      public   function   tambahProduk($nama,
$harga, $stok, $deskripsi) {
          $sql  =  "INSERT  INTO  produk  (nama,
harga, stok, deskripsi) VALUES (?, ?, ?, ?)";
        $stmt = $this->db->prepare($sql);
        $stmt->execute([$nama, $harga, $stok,
$deskripsi]);
    }

    public function getAllProduk() {
          $stmt  =  $this->db->query("SELECT  *
FROM produk");
                            return       $stmt-
>fetchAll(PDO::FETCH_ASSOC);
    }
}
?>
```

c. controllers/ProdukController.php

```
<?php
require_once "app/models/Produk.php";
require_once "app/views/produk_view.php";

class ProdukController {
    private $model;

    public function __construct() {
        $this->model = new Produk();
    }

    public function index() {
```

```
                $produk    =    $this->model-
>getAllProduk();
        $view = new ProdukView();
        $view->tampilkan($produk);
    }
}
?>
```

d. views/produk_view.php

```
<?php
class ProdukView {
    public function tampilkan($produk) {
        echo "<h2>Daftar Produk</h2>";
        echo "<table border='1'>
                <tr><th>ID</th><th>Nama</th>
<th>Harga</th><th>Stok</th><th>Deskripsi</th
></tr>";
        foreach ($produk as $p) {
            echo "<tr>
                    <td>{$p['id']}</td>
                    <td>{$p['nama']}</td>
                    <td>{$p['harga']}</td>
                    <td>{$p['stok']}</td>
                  <td>{$p['deskripsi']}</td>
                 </tr>";
        }
        echo "</table>";
    }
}
?>
```

e. index.php

```
<?php
require_once "config/Database.php";
require_once
"app/controllers/ProdukController.php";

$produkController = new ProdukController();
$produkController->index();
?>
```

Proyek akhir ini dirancang untuk melatih mahasiswa dalam membangun aplikasi nyata berbasis OOP PHP dengan pendekatan

modular dan terstruktur. Seluruh konsep penting seperti enkapsulasi, *inheritance*, polimorfisme, abstraksi, *trait*, *namespace*, *exception handling*, dan integrasi *database* diterapkan dalam satu kesatuan sistem. Dengan menggunakan PDO, aplikasi lebih aman dari SQL *Injection*. Penerapan MVC membuat kode lebih terorganisir, mudah dikembangkan, serta dapat diperluas ke level produksi dengan menambahkan fitur autentikasi, manajemen transaksi lebih kompleks, hingga integrasi API.

RANGKUMAN

Pada bab ini, mahasiswa diajak untuk menerapkan seluruh konsep Pemrograman Berorientasi Objek (OOP) dalam PHP ke dalam sebuah studi kasus nyata dan proyek akhir. Studi kasus sederhana berupa Sistem Manajemen Perpustakaan memperlihatkan bagaimana *class*, objek, *inheritance*, dan polimorfisme bekerja dalam aplikasi. Selanjutnya, proyek akhir berupa Sistem Manajemen Toko Online Sederhana dengan arsitektur MVC (*Model-View-Controller*) memberikan pengalaman langsung dalam mengintegrasikan konsep OOP dengan *database* serta penerapan *best practice* dalam pengembangan aplikasi. Dengan adanya proyek ini, mahasiswa tidak hanya memahami teori OOP tetapi juga mampu mengaplikasikannya dalam bentuk sistem nyata. Konsep seperti enkapsulasi, *inheritance*, polimorfisme, abstraksi, *interface*, *trait*, *namespace*, *exception handling*, dan penggunaan *database* menjadi lebih jelas karena diterapkan secara langsung. Proyek akhir juga melatih keterampilan analisis, perancangan sistem, implementasi, serta pengujian aplikasi berbasis OOP PHP. Dengan demikian, mahasiswa dipersiapkan untuk menghadapi tantangan nyata dalam dunia pemrograman berbasis web modern.

DAFTAR PUSTAKA

Adiwisastra, M. F., A. B. Hikmah, dan A. I. Warnilah. (2019). Dasar Pemrograman Web. Purwodadi: Penerbit CV. Sarnu Untung.

Adiwisastra, M. F., A. B. Hikmah. (2022). Desain Halaman Web Dengan CSS. Yogyakarta: Penerbit Graha Ilmu.

Balazs, J. (2020). Object-oriented PHP best practices: Principles, design patterns, and tools. Apress.

Fowler, M. (2019). UML distilled: A brief guide to the standard object modeling language (4th ed.). Addison-Wesley.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). Design patterns: Elements of reusable object-oriented software. Addison-Wesley.

Haryanto, A. (2019). Pemrograman PHP dan MySQL untuk pemula. Andi.

Linderman, M. (2021). Modern PHP: Advanced features and good practices. O'Reilly Media.

Pressman, R. S., & Maxim, B. R. (2020). Software engineering: A practitioner's approach (9th ed.). McGraw-Hill.

Sharan, K. (2017). Beginning PHP and MySQL: From novice to professional (5th ed.). Apress.

Sommerville, I. (2016). Software engineering (10th ed.). Pearson.

Wahana Komputer. (2017). Mudah belajar PHP dan MySQL. Andi.

Welling, L., & Thomson, L. (2017). PHP and MySQL web development (5th ed.). Addison-Wesley.

PHP Documentation. (2024). PHP manual. PHP.net. Retrieved September 12, 2025, from https://www.php.net/manual/

MySQL Documentation. (2024). MySQL reference manual. Oracle. Retrieved September 12, 2025, from https://dev.mysql.com/doc/

W3Schools. (2024). PHP tutorial. W3Schools. Retrieved September 12, 2025, from https://www.w3schools.com/php/

GLOSARIUM

A

Algoritma : Teknik penyusunan langkah-langkah penyelesaian masalah dalam bentuk kalimat dengan kata terbatas tetapi tersusun secara logis dan sistematis.

Array : Himpunan data sejenis yang disimpan dalam suatu variabel dengan *index* untuk mengakses setiap data yang tersimpan.

B

Boolean : Tipe data logika yang hanya memiliki dua nilai, yaitu *true* dan *false*, digunakan dalam pengambilan keputusan program.

C

Casting : Proses mengubah tipe data suatu variabel ke tipe data lain secara eksplisit sesuai kebutuhan program.

Class : Cetak biru (*blueprint*) dari sebuah objek yang mendefinisikan properti dan metode dalam pemrograman berorientasi objek.

Constructor : Metode khusus yang dijalankan secara otomatis saat objek dibuat untuk menginisialisasi properti.

D

Data Type : Jenis nilai yang dapat disimpan dalam variabel seperti integer, string, boolean, array, dan object.

E

Echo : Perintah dalam PHP yang digunakan untuk menampilkan output ke layar.

Encapsulation : Konsep OOP yang membungkus data dan metode dalam satu kelas serta membatasi akses langsung ke data.

F

Float : Tipe data numerik yang digunakan untuk bilangan desimal.

Function : Sekumpulan instruksi yang diberi nama dan dapat dipanggil untuk menjalankan tugas tertentu.

G

Global Variable : Variabel yang dideklarasikan di luar fungsi dan dapat diakses secara global.

H

HTML : Bahasa markup standar untuk membangun struktur halaman web.

I

Inheritance : Konsep pewarisan properti dan metode dari kelas induk ke kelas turunan untuk mendukung penggunaan ulang kode.

Integer : Tipe data bilangan bulat tanpa nilai desimal.

Interface : Struktur OOP yang berisi deklarasi metode tanpa implementasi dan harus diterapkan oleh kelas yang menggunakannya.

J

JIT (Just-In-Time Compiler)	:	Fitur PHP 8 yang meningkatkan performa dengan mengompilasi kode saat runtime.

K

Konstanta	:	Identifier yang menyimpan nilai tetap dan tidak dapat diubah selama program berjalan.

L

Looping	:	Struktur kontrol yang digunakan untuk menjalankan perintah secara berulang.

M

Method	:	Fungsi yang berada di dalam kelas dan merepresentasikan perilaku suatu objek.
Modifier Akses	:	Kata kunci *public*, *private*, dan *protected* untuk mengatur hak akses properti dan metode.

N

Namespace	:	Mekanisme untuk mengelompokkan class atau fungsi agar tidak terjadi konflik nama.

O

Object	:	Instansi nyata dari sebuah class yang memiliki nilai properti dan metode.
OOP (Object Oriented Programming)	:	Paradigma pemrograman yang berfokus pada objek sebagai representasi dunia nyata.

P

PHP : Bahasa pemrograman *server-side* yang digunakan untuk membangun aplikasi web dinamis.

Polymorphism : Kemampuan objek berbeda untuk merespons metode yang sama dengan perilaku berbeda.

Property : Variabel yang didefinisikan dalam kelas untuk menyimpan data objek.

Q

Query : Perintah yang digunakan untuk mengakses atau memanipulasi data dalam database.

R

Resource : Tipe data khusus untuk merepresentasikan sumber daya eksternal seperti file atau koneksi database.

S

Scope : Ruang lingkup akses suatu variabel dalam program seperti local, global, dan static.

Static : Kata kunci untuk mempertahankan nilai variabel atau memanggil metode tanpa instansiasi objek.

String : Tipe data yang digunakan untuk menyimpan teks atau karakter.

T

Type Juggling	:	Kemampuan PHP untuk mengonversi tipe data secara otomatis sesuai konteks penggunaan.

U

Undefined Variable	:	Kondisi ketika variabel digunakan tanpa dideklarasikan terlebih dahulu.

V

Variable	:	Wadah penyimpanan data yang nilainya dapat berubah selama program berjalan.

W

Web Server	:	Perangkat lunak yang menerima dan merespons permintaan dari browser.

X

XAMPP	:	Paket server lokal yang terdiri dari Apache, MySQL/MariaDB, PHP, dan Perl untuk pengembangan web.

Y

Yield	:	Perintah pada fungsi generator untuk mengembalikan nilai secara bertahap.

Z

Zend Engine	:	Mesin inti PHP yang bertanggung jawab dalam eksekusi skrip PHP.

BIODATA PENULIS

Miftah Farid Adiwisastra, S.T., M.Kom., lahir di Tasikmalaya 10 Juni 1986 lulus Program Studi Magister Ilmu Komputer (S-2) di STMIK Nusa Mandiri pada tahun 2015 dan sedang menempuh Studi Doktoral Informatika (S-3) di Universitas Telkom Bandung. Saat ini menjadi dosen tetap di Universitas Bina Sarana Informatika Kampus Kota Tasikmalaya sejak 2013 sampai sekarang. Beberapa karya yang penulis buat di antaranya: *Buku Dasar Pemrograman Web* (2019), *Buku Desain Halaman Web dengan CSS* (2020), dan *Buku E-Business* (2022), *Buku Penerapan Konsep Dasar Logika Algoritma Dengan Pemrograman PHP* (2023). Selain itu penulis juga aktif menulis karya ilmiah lainnya.

Haerul Fatah, M.Kom., lahir di Bandung pada tanggal 02 April 1994, telah menyelesaikan pendidikan pascasarjana Magister di bidang Ilmu Komputer dan memperoleh gelar Magister Komputer (M.Kom) pada tahun 2018 di STMIK Nusa Mandiri. Saat ini menjadi Dosen tetap di Universitas Bina Sarana Informatika Kampus Tasikmalaya pada Program Studi Sistem Informasi sejak tahun 2019 sampai sekarang.

Herlan Sutisna, S.T., M.Kom., lahir di Tasikmalaya pada Desember 1987. menuntaskan pendidikan pascasarjana Magister Ilmu Komputer di STMIK Nusa Mandiri pada tahun 2015. Aktif mengajar dari tahun 2012 sampai sekarang, dan berhasil Meraih Sertifikasi Dosen pada tahun 2020. Beberapa karya yang penulis buat di antaranya: *Buku Mahir Framework Kekinian Tanpa Ribet*, dan *Buku English For Computer*. Saat ini menjadi Dosen Tetap di Universitas Bina Sarana Informatika (BSI) Kampus Tasikmalaya pada Program Studi Sistem Informasi dan giat juga mengikuti berbagai program yang diselenggarakan oleh Kemdiktisaintek.

Bambang Kelana Simpony, S.T., M.Kom., seorang penulis dan akademisi di bidang teknologi informasi. Lahir pada tanggal 12 September 1986 di Ciamis. Menyelesaikan Program Sarjana (S-1) di STMIK DCI Tasikmalaya, jurusan Teknik Informatika, pada tahun 2013. Kemudian melanjutkan pendidikan pascasarjana dan meraih gelar Magister Komputer (M.Kom.) di STMIK Nusa Mandiri pada tahun 2015.Sebagai seorang dosen tetap, saat ini mengajar di Universitas Bina Sarana Informatika Kampus Kota Tasikmalaya. Selain karier akademik, juga memiliki minat dalam menulis buku dan berkontribusi dalam pengembangan literatur pemrograman.

Dini Silvi Purnia, S.Kom., M.Kom., Lahir di Tasikmalaya, 27 Oktober 1990. Telah menyelesaikan program Magister Ilmu Komputer di STMIK Nusa Mandiri dan Lulus tahun 2016. Sejak tahun 2012 penulis telah berprofesi sebagai Dosen dan saat ini aktif mengajar di Universitas Bina Sarana Informatika Program Studi Sistem Informasi, Mengampu Mata kuliah Metode Penelitian, Sistem basis Data, E-Business dan Web Programming.

Perkembangan teknologi informasi menuntut penguasaan paradigma pemrograman yang mampu menghasilkan perangkat lunak yang terstruktur, modular, dan mudah dikembangkan. Pemrograman Berorientasi Objek dengan PHP: Teori dan Praktik hadir sebagai buku ajar yang dirancang untuk membantu mahasiswa, dosen, dan praktisi memahami serta mengimplementasikan konsep Object Oriented Programming (OOP) secara sistematis menggunakan bahasa pemrograman PHP.

Buku ini membahas secara bertahap mulai dari konsep dasar OOP dan fundamental PHP, hingga implementasi lanjutan seperti enkapsulasi, pewarisan, polimorfisme, pengelolaan database berbasis OOP, serta perancangan dan arsitektur aplikasi. Pembaca juga diajak memahami penerapan konsep melalui studi kasus dan proyek akhir yang merefleksikan kebutuhan pengembangan aplikasi web modern.

Disusun dengan pendekatan teoritis dan praktis, setiap bab dilengkapi tujuan pembelajaran, contoh kode program, rangkuman materi, dan latihan untuk memperkuat pemahaman. Dengan struktur yang sistematis dan aplikatif, buku ini diharapkan menjadi referensi yang efektif dalam membangun kompetensi pemrograman berorientasi objek serta mendukung pengembangan aplikasi web yang profesional dan berkelanjutan.

PT Insan Cendekia Mandiri Group
0813-7272-5118
@penerbiticm
Penerbit Insan Cendekia Mandiri
www.insancendekiamandiri.co.id
PENDIDIKAN
ISBN 978-634-252-250-9
9 786342 522509