



**LAPORAN  
PRAKTIKUM PEMOGRAMAN  
BERBASIS OBJEK (PBO)**

**LAPORAN PRAKTIKUM  
PEMOGRAMAN BERBASIS OBJEK**

**Disusun Oleh:  
Muhammad Hamzah Nur Fadhilah  
223443042**

**JURUSAN TEKNIK OTOMASI MANUFAKTUR DAN MEKATRONIKA  
POLITEKNIK MANUFAKTUR NEGERI BANDUNG  
BANDUNG  
SEPTEMBER 2024**



**LAPORAN PRAKTIKUM**  
**PEMOGRAMAN BERBASIS OBJEK**

Disusun oleh:  
Muhammad Hamzah Nur Fadhillah  
223443042



**JURUSAN TEKNIK OTOMASI MANUFAKTUR DAN MEKATRONIKA**  
**POLITEKNIK MANUFAKTUR NEGERI BANDUNG**  
**BANDUNG**  
**SEPTEMBER 2024**



**DAFTAR ISI**

**Sampul Dalam.....**

**Daftar Isi.....**

**Biodata.....**

**Teori.....**

**Studi Kasus.....**

**Penutup.....**



## BIODATA



### Informasi Data Diri

Nama Lengkap : Muhammad Hamzah Nur Fadhillah  
NIM : 223443042  
Jurusan : Teknik Otomasi Manufaktur dan Mekatronika  
Kelas : 2 AEC 2  
Email : 223443042@mhs.polman-bandung.ac.id  
Kontak : 081321449895

## TEORI

Dalam pengembangan aplikasi berbasis PHP, konsep Object-Oriented Programming (OOP) sangat penting untuk meningkatkan modularitas dan fleksibilitas kode. Sebagaimana dijelaskan oleh Wibowo dalam penelitiannya, OOP pada PHP memberikan kemudahan dalam hal pengelompokan fungsi-fungsi ke dalam kelas, yang memungkinkan pemrograman lebih terstruktur dan efisien. Studi tersebut mengulas berbagai aspek OOP, termasuk pewarisan, enkapsulasi, dan polimorfisme, yang menjadi dasar bagi pengembangan perangkat lunak modern.

[1]

### Daftar Pustaka

- [1] K. Wibowo AMIK Bina Sarana Informatika Jl Rs Fatmawati No, P. Labu, and J. Selatan, "ANALISA KONSEP OBJECT ORIENTED PROGRAMMING PADA BAHASA PEMROGRAMAN PHP," 2015.



## STUDI KASUS

### BAGIAN 1

Catatan:

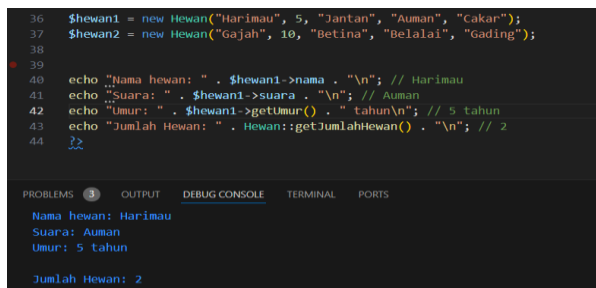
#### 1. Objek

Objek adalah hasil instansiasi dari class, dan mengandung seluruh *resource* yang telah didefinisikan pada class. [1]

#### Kode program

```
// Membuat objek Hewan
$hewan1 = new Hewan("Harimau", 5, "Jantan",
"Auman", "Cakar");
$hewan2 = new Hewan("Gajah", 10, "Betina",
"Belalai", "Gading");
```

`new Hewan(...)` adalah proses pembuatan objek. Ketika objek dibuat, konstruktor (`__construct()`) di dalam kelas `Hewan` akan dipanggil untuk menginisialisasi data pada objek tersebut. Dalam hal ini, `$hewan1` memiliki nama "Harimau", umur 5 tahun, jenis kelamin "Jantan", suara "Auman", dan pertahanan "Cakar".



```
36 $hewan1 = new Hewan("Harimau", 5, "Jantan", "Auman", "Cakar");
37 $hewan2 = new Hewan("Gajah", 10, "Betina", "Belalai", "Gading");
38
39
40 echo "Nama hewan: " . $hewan1->nama . "\n"; // Harimau
41 echo "Suara: " . $hewan1->suara . "\n"; // Auman
42 echo "Umur: " . $hewan1->getUmur() . " tahun\n"; // 5 tahun
43 echo "Jumlah Hewan: " . Hewan::getJumlahHewan() . "\n"; // 2
44 ?>
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
Nama hewan: Harimau
Suara: Auman
Umur: 5 tahun
Jumlah Hewan: 2
```

## 2. Class

Class merupakan Gambaran dari sebuah object atau bisa dikatakan output dari sebuah object. Pada Bahasa pemograman calss merupakan sekumpulan kode yang dituliskan untuk mendefinisikan property dan method yang ada pada sebuah object. [1]

### Kode Program

```
<?php
class Hewan {
public $nama;
private $umur;
protected $jenisKelamin;
public $suara;
public $pertahanan;
public static $warna;
private static $jumlahHewan = 0;
```

Kelas Hewan didefinisikan dengan berbagai properti:

- **public:** Atribut seperti \$nama, \$suara, dan \$pertahanan dapat diakses langsung dari luar kelas.
- **private:** Atribut \$umur hanya dapat diakses di dalam kelas dan tidak langsung diakses dari luar.
- **protected:** Atribut \$jenisKelamin hanya dapat diakses dari dalam kelas ini atau kelas turunannya.
- **static:** Properti seperti \$warna dan \$jumlahHewan dimiliki oleh kelas, bukan oleh objek individual. Ini berarti mereka berbagi nilai di antara semua objek yang dibuat dari kelas Hewan.

```

1 </php>
2 class Hewan {
3     public $nama;
4     private $umur;
5     protected $jenisKelamin;
6     public $suara;
7     public $pertahanan;
8     public static $warna;
9     private static $jumlahHewan = 0;
10
11     public function __construct($nama, $umur, $jenisKelamin, $suara, $pertahanan) {
12         $this->nama = $nama;
13         $this->umur = $umur;
14         $this->jenisKelamin = $jenisKelamin;
15         $this->suara = $suara;
16         $this->pertahanan = $pertahanan;
17     }
18 }

```

### 3. Akses Modifier

#### Public

**Deskripsi:** Anggota yang dideklarasikan sebagai public dapat diakses dari mana saja: baik dari dalam kelas, luar kelas, maupun dari objek yang dibuat dari kelas tersebut.

#### Private

**Deskripsi:** Anggota yang dideklarasikan sebagai private hanya dapat diakses dari dalam kelas itu sendiri. Tidak bisa diakses dari luar kelas, termasuk oleh kelas turunan.

#### Protected

**Deskripsi:** Anggota yang dideklarasikan sebagai protected dapat diakses dari dalam kelas dan oleh kelas yang mewarisi (inherit) kelas tersebut. Namun, tidak dapat diakses dari luar kelas tersebut secara langsung.

#### Penjelasan :

**Public:** public \$nama, public \$suara, dan public \$pertahanan dapat diakses dari luar kelas, memungkinkan interaksi langsung dengan data tersebut.

**Private:** private \$umur dan private static \$jumlahHewan hanya dapat diakses dari dalam kelas. Getter dan setter digunakan

untuk mengakses dan mengubah nilai private property dari luar kelas.

**Protected:** protected \$jenisKelamin hanya dapat diakses oleh kelas itu sendiri dan kelas-kelas turunan, tapi tidak dapat diakses secara langsung dari luar kelas.

## Public

```
public.php > ...
1  <?php
   14 references | 2 implementations
2  class Hewan {
   7 references
3      public $nama;
4  }
5
6  $hewan = new Hewan();
7  $hewan->nama = "Harimau";
8  ?>
```

## Private

```
1  <?php
   14 references | 2 implementations
2  class Hewan {
   8 references
3      private $umur;
4
   1 reference | 0 overrides
5      public function setUmur($umur) {
6          $this->umur = $umur;
7      }
8
   2 references | 0 overrides
9      public function getUmur() {
10         return $this->umur;
11     }
12 }
13
14 $hewan = new Hewan();
15 $hewan->setUmur(5);
16 echo $hewan->getUmur();
17
18 ?>
```

## Protected

```
1  <?php
   14 references | 3 implementations
2  class Hewan {
   6 references
3      protected $jenisKelamin;
4
   1 reference | 1 override
5      public function setJenisKelamin($jenisKelamin) {
6          $this->jenisKelamin = $jenisKelamin;
7      }
8  }
9
   1 reference | 0 implementations
10 class Kucing extends Hewan {
   1 reference | 0 overrides
11     public function getJenisKelamin() {
12         return $this->jenisKelamin;
13     }
14 }
15
16 $kucing = new Kucing();
17 $kucing->setJenisKelamin("Betina");
18 echo $kucing->getJenisKelamin();
19
20
21 ?>
```

#### 4. Construct

PHP memungkinkan pengembangan untuk menyatakan metode konstruktor untuk sebuah class. Class yang memiliki metode ini pada setiap objek yang baru dibentuk (diinstansiasi), diperlukan inialisasi sebelum objek digunakan. [1]

#### Kode Program

```
public function __construct($nama, $umur,
    $jenisKelamin, $suara, $pertahanan) {
    // Inisialisasi properti dengan nilai yang
    diterima
    $this->nama = $nama;
    $this->umur = $umur;
    $this->jenisKelamin = $jenisKelamin;
    $this->suara = $suara;
    $this->pertahanan = $pertahanan;
    // Menambah jumlah hewan setiap kali objek
    baru dibuat
    self::$jumlahHewan++;
}
```

#### Penjelasan :

- **ParameterConstructor:** (\$nama, \$umur, \$jenisKelamin, \$suara, \$pertahanan) menerima nilai yang akan digunakan untuk menginisialisasi properti objek.
- **Inisialisasi Properti:** Nilai parameter digunakan untuk menetapkan nilai awal pada properti objek.
- Ketika `new Hewan(...)` dipanggil, constructor akan dipanggil secara otomatis, dan nilai yang diberikan ("Harimau", 5, "Jantan", "Auman", "Cakar") digunakan untuk menginisialisasi properti objek \$hewan1.

```

// Constructor
6 references | 2 overrides
public function __construct($nama, $umur, $jenisKelamin, $suara, $pertahanan) {
    $this->nama = $nama;
    $this->umur = $umur;
    $this->jenisKelamin = $jenisKelamin;
    $this->suara = $suara;
    $this->pertahanan = $pertahanan;

    self::$jumlahHewan++;
}

37 // Membuat objek Hewan
38 $hewan1 = new Hewan("Harimau", 5, "Jantan", "Auman", "Cakar");
39 $hewan2 = new Hewan("Gajah", 10, "Betina", "Belalai", "Gading");
40
41 // Mengakses dan menampilkan informasi objek
42 echo "Nama hewan: " . $hewan1->nama . "\n"; // Harimau
43 echo "Suara: " . $hewan1->suara . "\n"; // Auman
44 echo "Umur: " . $hewan1->getUmur() . " tahun\n"; // 5 tahun
45 echo "Jumlah Hewan: " . Hewan::getJumlahHewan() . "\n"; // 2
46 }>

```

PROBLEMS 4 OUTPUT DEBUG CONSOLE TERMINAL PORTS

```

Nama hewan: Harimau
Suara: Auman
Umur: 5 tahun
Jumlah Hewan: 2

```

## 5. Static Properti

Adalah variabel yang dideklarasikan dengan kata kunci static. Mereka dimiliki oleh kelas itu sendiri, bukan oleh objek individual. Semua objek dari kelas yang sama berbagi nilai dari properti static tersebut. [1]

### Kode Program :

```
<?php
class Hewan {
    public static $jumlahHewan = 0;

    public function __construct() {
        self::$jumlahHewan++;
    }
}

// Membuat objek Hewan
$hewan1 = new Hewan();
$hewan2 = new Hewan();

// Mengakses static property
echo "Jumlah Hewan: " . Hewan::$jumlahHewan; //
Output: 2

?>
```

**Deklarasi:** public static \$jumlahHewan = 0;

- Properti static dideklarasikan dengan kata kunci static dan milik kelas, bukan milik objek individual. Semua objek dari kelas Hewan berbagi nilai dari \$jumlahHewan.

**Akses:** Hewan::\$jumlahHewan

- Properti static dapat diakses menggunakan sintaks NamaKelas::\$namaProperti, tanpa perlu membuat objek dari kelas tersebut.



```

1  <?php
   17 references | 2 implementations
2  class Hewan {
   6 references
3      public static $jumlahHewan = 0;
4
   6 references | 2 overrides
5      public function __construct() {
6          self::$jumlahHewan++;
7      }
8  }
9
10 // Membuat objek Hewan
11 $hewan1 = new Hewan();
12 $hewan2 = new Hewan();
13
14 // Mengakses static property
15 echo "Jumlah Hewan: " . Hewan::$jumlahHewan; // Output: 2
16
17 ?>

```

PROBLEMS

4

OUTPUT

DEBUG CONSOLE

TERMINAL

PORTS

Jumlah Hewan: 2

## 6. Static Method

Static methods adalah metode yang dideklarasikan dengan kata kunci `static`. Mereka dapat dipanggil tanpa membuat instansi objek dari kelas. Static methods hanya dapat mengakses properti static dan metode static lainnya.

```
<?php
class Hewan {
    public static $warna = "Coklat";

    public static function getWarna() {
        return self::$warna;
    }
}

echo "Warna Hewan: " . Hewan::getWarna(); // Output:
Coklat

?>
```

**Deklarasi:** `public static function getWarna()`

- Metode static dideklarasikan dengan kata kunci `static`. Metode ini tidak bergantung pada instansi objek dan hanya dapat mengakses properti static dan metode static lainnya.

**Akses:** `Hewan::getWarna()`

- Metode static dapat diakses menggunakan sintaks `NamaKelas::namaMetode()`, tanpa perlu membuat objek dari kelas tersebut.

```
coba.php > PHP > Hewan
1  <?php
   15 references | 2 implementations
2  class Hewan {
   2 references
3      public static $warna = "Coklat";
4
   1 reference | 0 overrides
5      public static function getWarna() {
6          return self::$warna;
7      }
8  }
9
10 // Mengakses static method tanpa membuat objek
11 echo "Warna Hewan: " . Hewan::getWarna(); // Output: Coklat
12
13 ~>
```

PROBLEMS 4 OUTPUT DEBUG CONSOLE TERMINAL PORTS

Warna Hewan: Coklat

## 7. Self dan Parent

Self digunakan untuk merujuk pada kelas yang sama di mana ia digunakan. Ini berguna untuk mengakses atau memodifikasi properti static dan metode static dalam kelas yang sama.

```
<?php
class Hewan {
    public static $jumlahHewan = 0;

    public static function tambahHewan() {
        self::$jumlahHewan++;
    }

    public static function getJumlahHewan() {
        return self::$jumlahHewan;
    }
}

// Memanggil metode static tanpa membuat objek
Hewan::tambahHewan();
echo Hewan::getJumlahHewan(); // Output: 1

?>
```

### Penjelasan Self :

**Konteks:** Digunakan dalam konteks kelas saat ini.

**Akses:** Mengakses atau memodifikasi properti dan metode static yang dideklarasikan di kelas yang sama.

**Contoh:** `self::$jumlahHewan` mengakses properti static `$jumlahHewan` di dalam kelas `Hewan`.

Parent digunakan untuk merujuk pada kelas induk dari kelas saat ini. Ini sangat berguna dalam situasi pewarisan, di mana Anda mungkin ingin memanggil metode atau mengakses properti dari kelas dasar (superclass) dalam kelas turunan (subclass).

```
class Kucing extends Hewan {
    public function __construct($nama) {
        parent::__construct($nama); // Memanggil
        konstruktor kelas induk
    }

    public function suara() {
        return "Meow!"; // Override metode dari kelas
        induk
    }

    public function info() {
        return parent::suara() . " Tetapi saya adalah
        kucing."; // Memanggil metode kelas induk
    }
}

$kucing = new Kucing("Tom");
echo $kucing->info(); // Output: Suara umum hewan.
Tetapi saya adalah kucing.
```

### **Penjelasan Parent :**

**Konteks:** Digunakan untuk merujuk pada kelas induk dari kelas saat ini.

**Akses:** Memanggil metode dan mengakses properti yang dideklarasikan di kelas induk.

**Contoh:** `parent::__construct($nama)` memanggil konstruktor kelas Hewan dari kelas Kucing.

## Self

```
1 <?php
17 references | 2 implementations
2 class Hewan {
3     6 references
4     public static $jumlahHewan = 0;
5     1 reference | 0 overrides
6     public static function tambahHewan() {
7         self::$jumlahHewan++;
8     }
9     5 references | 0 overrides
10    public static function getJumlahHewan() {
11        return self::$jumlahHewan;
12    }
13 }
14 // Menganggil metode static tanpa membuat objek
15 Hewan::tambahHewan();
16 echo Hewan::getJumlahHewan(); // Output: 1
17
18 ?>
```

PROBLEMS 4 OUTPUT DEBUG CONSOLE TERMINAL PORTS

1

## Parent

```
1 reference | 0 implementations
14 class Kucing extends Hewan {
15     4 references | 0 overrides | prototype
16     public function __construct($nama) {
17         parent::__construct($nama); // Menganggil konstruktor kelas induk
18     }
19     1 reference | 0 overrides | prototype
20     public function suara() {
21         return "Meow!"; // Override metode dari kelas induk
22     }
23     1 reference | 0 overrides
24     public function info() {
25         return parent::suara() . " Tetapi saya adalah kucing."; // Menganggil metode kelas induk
26     }
27 }
28 $kucing = new Kucing("Tom");
29 echo $kucing->info(); // Output: Suara umum hewan. Tetapi saya adalah kucing.
30
31
32 ?>
```

PROBLEMS 4 OUTPUT DEBUG CONSOLE TERMINAL PORTS

Suara umum hewan. Tetapi saya adalah kucing.

## 8. Final

Final adalah kata kunci dalam PHP yang digunakan untuk mengontrol pewarisan dan overriding dalam pemrograman berorientasi objek (OOP). Ini digunakan untuk mencegah kelas, metode, atau properti dari diwariskan atau di-override oleh kelas turunan.

Ketika sebuah kelas dideklarasikan sebagai final, kelas tersebut tidak dapat diwarisi oleh kelas lain. Ini mencegah kelas turunan membuat subclass dari kelas final tersebut.

### Kode Program :

```
final class Hewan {
    public function suara() {
        return "Suara umum hewan.";
    }
}

// Ini akan menghasilkan error karena kelas
`Hewan` adalah final
// class Kucing extends Hewan {
//     // ...
// }

// Kelas ini dapat diwarisi karena tidak final
class Burung {
    // Metode final tidak dapat di-override
    final public function terbang() {
        return "Burung terbang.";
    }
}

// Kelas Turunan
class Elang extends Burung {
    // Ini akan menghasilkan error karena metode
    `terbang` adalah final
    // public function terbang() {
    //     return "Elang terbang tinggi!";
    // }
```

```

        // }
    }

    // Penggunaan Kelas Final dan Metode Final
    $hewan = new Hewan();
    echo $hewan->suara(); // Output: Suara umum
    hewan.

    // Instansi dari kelas yang tidak final
    $burung = new Burung();
    echo $burung->terbang(); // Output: Burung
    terbang.

    // Instansi dari kelas turunan
    $elang = new Elang();
    echo $elang->terbang(); // Output: Burung
    terbang.

```



```

16 class Burung {
17     // Metode final tidak dapat di-override
18     // 2 references
19     final public function terbang() {
20         return "Burung terbang.";
21     }
22 }
23 // Kelas Turunan
24 // 1 reference | 0 implementations
25 class Elang extends Burung {
26     // Ini akan menghasilkan error karena metode `terbang` adalah final
27     // public function terbang() {
28     //     return "Elang terbang tinggi!";
29     // }
30 }
31 // Penggunaan Kelas Final dan Metode Final
32 $hewan = new Hewan();
33 echo $hewan->suara(); // Output: Suara umum hewan.
34
35 // Instansi dari kelas yang tidak final
36 $burung = new Burung();
37 echo $burung->terbang(); // Output: Burung terbang.
38
39 // Instansi dari kelas turunan
40 $elang = new Elang();

```

PROBLEMS 4 OUTPUT DEBUG CONSOLE TERMINAL PORTS

Suara umum hewan.Burung terbang.Burung terbang.

## 9. Overriding

Overriding adalah konsep dalam pemrograman berorientasi objek di mana sebuah metode yang dideklarasikan dalam kelas dasar (superclass) diubah implementasinya dalam kelas turunan (subclass). Ini memungkinkan subclass untuk memberikan implementasi yang lebih spesifik atau berbeda dari metode yang diwarisi dari kelas dasar.

### Bagaimana Overriding Bekerja

1. **Deklarasi Metode di Kelas Dasar:** Kelas dasar memiliki metode yang dideklarasikan dengan implementasi tertentu.
2. **Override Metode di Kelas Turunan:** Kelas turunan mendeklarasikan metode dengan nama yang sama dan tanda tangan yang sama (parameter dan tipe) dengan metode di kelas dasar. Implementasi di kelas turunan menggantikan implementasi di kelas dasar.

### Kode Program :

```
class Kucing extends Hewan {
    // Overriding method suara
    public function suara() {
        return "Meow!";
    }
}

class Anjing extends Hewan {
    // Overriding method suara
    public function suara() {
        return "Woof!";
    }
}

// Membuat objek dari kelas turunan
$ kucing = new Kucing();
echo $ kucing->suara(); // Output: Meow!
```

```
$anjing = new Anjing();
echo $anjing->suara(); // Output: Woof!
```

- Kelas Hewan memiliki metode suara yang mengembalikan string "Suara umum hewan."
- Kelas Kucing mewarisi dari Hewan dan mengoverride metode suara. Implementasinya diubah menjadi "Meow!".
- Ketika metode suara dipanggil pada objek Kucing, implementasi baru yang ada di Kucing yang akan digunakan.
- Kelas Anjing juga mewarisi dari Hewan dan mengoverride metode suara. Implementasinya diubah menjadi "Woof!".
- Ketika metode suara dipanggil pada objek Anjing, implementasi baru yang ada di Anjing yang akan digunakan.
- instansi Kucing dan Anjing masing-masing memanggil metode suara. Metode yang dioverride oleh kelas turunan Kucing dan Anjing menggantikan metode di kelas dasar Hewan.

```

1 reference | 0 implementations
9  class Kucing extends Hewan {
10     // Overriding method suara
11     2 references | 0 overrides | prototype
12     public function suara() {
13         return "Meow!";
14     }
15 }

1 reference | 0 implementations
16 class Anjing extends Hewan {
17     // Overriding method suara
18     2 references | 0 overrides | prototype
19     public function suara() {
20         return "Woof!";
21     }
22 }

23 // Membuat objek dari kelas turunan
24 $kucing = new Kucing();
25 echo $kucing->suara(); // Output: Meow!
26
27 $anjing = new Anjing();
28 echo $anjing->suara(); // Output: Woof!
29
30 ?>

```

PROBLEMS 4 OUTPUT DEBUG CONSOLE TERMINAL PORTS

Meow!Woof!

## 10. Trait

Trait adalah fitur dalam PHP yang memungkinkan Anda untuk menyusun ulang kode secara lebih fleksibel dengan cara yang berbeda dari pewarisan kelas tradisional. Traits memungkinkan Anda untuk menyertakan fungsionalitas di beberapa kelas tanpa harus menggunakan pewarisan ganda atau mengubah hierarki kelas.

Traits berguna dalam situasi di mana Anda ingin berbagi metode atau properti di berbagai kelas tanpa mengubah hierarki pewarisan atau mengulang kode. Traits bisa digunakan untuk:

- Menghindari duplikasi kode.
- Menyediakan fungsionalitas yang dapat digunakan oleh banyak kelas.
- Mengelola kode yang melibatkan beberapa aspek atau tanggung jawab.

### Kode Program :

```
<?php

trait SuaraTrait {
    public function suara() {
        return "Suara dari trait.";
    }
}

class Kucing {
    use SuaraTrait;
}

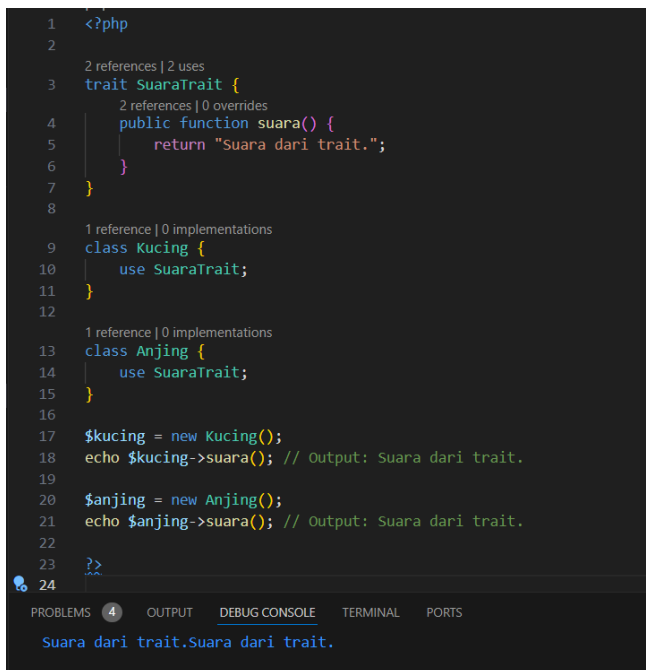
class Anjing {
    use SuaraTrait;
}

$kucing = new Kucing();
echo $kucing->suara(); // Output: Suara dari
trait.
```

```
$anjing = new Anjing();  
echo $anjing->suara(); // Output: Suara dari  
trait.
```

```
?>
```

- Trait SuaraTrait dideklarasikan dengan metode suara yang mengembalikan string "Suara dari trait."
- Kelas Kucing dan Anjing masing-masing menggunakan trait SuaraTrait dengan kata kunci use.
- Dengan menggunakan trait, kedua kelas ini mendapatkan metode suara dari SuaraTrait.
- Membuat objek dari kelas Kucing dan Anjing, dan memanggil metode suara yang dihasilkan dari trait SuaraTrait.



```
1 <?php  
2  
3 trait SuaraTrait {  
4     public function suara() {  
5         return "Suara dari trait.";  
6     }  
7 }  
8  
9 class Kucing {  
10     use SuaraTrait;  
11 }  
12  
13 class Anjing {  
14     use SuaraTrait;  
15 }  
16  
17 $kucing = new Kucing();  
18 echo $kucing->suara(); // Output: Suara dari trait.  
19  
20 $anjing = new Anjing();  
21 echo $anjing->suara(); // Output: Suara dari trait.  
22  
23 ?>  
24
```

PROBLEMS 4 OUTPUT DEBUG CONSOLE TERMINAL PORTS

Suara dari trait.Suara dari trait.

## 11. Inherit kelas dari file yang berbeda

```
Hewan.php > ...
1  <?php
2  // Mendefinisikan kelas dasar Hewan
   14 references | 3 implementations
3  class Hewan {
   8 references
4      public $nama;
5
   4 references | 2 overrides
6      public function __construct($nama) {
7          $this->nama = $nama;
8      }
9
   2 references | 1 override
10     public function suara() {
11         return "Suara umum hewan.";
12     }
13 }
14 ?>
15
```

```
Kucing.php > PHP > Kucing > suara()
1  <?php
2
3  require_once 'Hewan.php';
4
5  // Mendefinisikan kelas turunan Kucing
   2 references | 0 implementations
6  class Kucing extends Hewan {
   2 references | 0 overrides | prototype
7      public function suara() {
8          return "Meow!";
9      }
10 }
11 ?>
12
```

```

index.php > ...
1  <?php
2  // Sertakan file yang berisi kelas Kucing
3  require_once 'Kucing.php';
4
5  // Membuat objek dari kelas Kucing
6  $kucing = new Kucing("Kitty");
7  echo $kucing->nama; // Output: Kitty
8  echo $kucing->suara(); // Output: Meow!
9  ?>
10

```

#### **Hewan.php:**

- Mendefinisikan kelas Hewan dengan properti \$nama dan metode suara().
- Metode suara() memberikan output default "Suara umum hewan."

#### **Kucing.php:**

- Menggunakan require\_once 'Hewan.php'; untuk menyertakan file Hewan.php, yang mendefinisikan kelas Hewan.
- Mendefinisikan kelas Kucing yang mewarisi dari Hewan dan mengoverride metode suara() untuk memberikan output "Meow!".

#### **index.php:**

- Menggunakan require\_once 'Kucing.php'; untuk menyertakan file Kucing.php, yang pada gilirannya menyertakan Hewan.php.
- Membuat objek dari kelas Kucing dan memanggil metode suara().

## 12. Polymorphism

Polymorphism membuat objek-objek yang berasal dari subclass yang berbeda, di perlakukan sebagai objek-objek dari satu superclass. Hal ini terjadi Ketika memilih method yang sesuai untuk diimplementasikan ke objek tertentu berdasarkan pada subclass yang memiliki method bersangkutan. [1]

Kondisi yang harus di penuhi supaya polimorfisme dapat diimplementasikan adalah :

- Method yang di panggil harus melalui variable dari basis class atau superclass
- Method yang di panggil harus juga menjadi method dari basis class
- Signature (argument/parameter) method harus sama baik pada superclass maupun subclass.
- Method access attribute pada subclass tidak boleh lebih terbatas dari basic class.

Kode Program :

Polymorphism melalui overriding

```
<?php

// Kelas Dasar
class Hewan {
    public function suara() {
        return "Suara umum hewan.";
    }
}

// Kelas Turunan
class Kucing extends Hewan {
    public function suara() {
        return "Meow!";
    }
}
```



```

class Anjing extends Hewan {
    public function suara() {
        return "Woof!";
    }
}

// Fungsi untuk menunjukkan polymorphism
function tampilkanSuara(Hewan $hewan) {
    echo $hewan->suara() . "\n";
}

// Membuat objek dari kelas turunan
$kucing = new Kucing();
$anjing = new Anjing();

// Menggunakan fungsi dengan polymorphism
tampilkanSuara($kucing); // Output: Meow!
tampilkanSuara($anjing); // Output: Woof!

?>

```

- Kelas Hewan memiliki metode suara yang mengembalikan string "Suara umum hewan."
- Kelas Kucing dan Anjing masing-masing mewarisi dari Hewan dan mengoverride metode suara dengan implementasi yang spesifik.
- Fungsi ini menerima parameter bertipe Hewan. Karena Kucing dan Anjing adalah subclass dari Hewan, objek dari kelas-kelas tersebut dapat diterima oleh fungsi ini.
- Fungsi tampilkanSuara dipanggil dengan objek Kucing dan Anjing. Meskipun metode suara dipanggil dengan nama yang sama, implementasinya berbeda tergantung pada kelas objek yang diteruskan.

```

17 class Anjing extends Hewan {
18     2 references | 0 overrides | prototype
19     public function suara() {
20         return "Woof!";
21     }
22 }
23
24 // Fungsi untuk menunjukkan polymorphism
25 2 references
26 function tampilkanSuara(Hewan $hewan) {
27     echo $hewan->suara() . "\n";
28 }
29
30 // Membuat objek dari kelas turunan
31 $kucing = new Kucing();
32 $anjing = new Anjing();
33
34 // Menggunakan fungsi dengan polymorphism
35 tampilkanSuara($kucing); // Output: Meow!
36 tampilkanSuara($anjing); // Output: Woof!
37
38 }>

```

PROBLEMS 7

OUTPUT

DEBUG CONSOLE

TERMINAL

PORTS

Meow!

Woof!

### 13. Kelas dan metode Abstrak

PHP memperkenalkan kelas abstrak dan metode abstrak. Class yang mendefinisikan sebagai abstrak tidak bisa diinstansiasi, dan class yang terdiri paling tidak satu metode harus didefinisikan sebagai kelas abstrak. Kelas abstrak hanya bisa mewariskan resourcenya.

Pada kelas anaknya. Setiap kelas yang mewarisi kelas abstrak, wajib menuliskan seluruh metode abstrak yang dimiliki oleh parent kelasnya.  
[1]

#### Kode program kelas :

```
<?php

// Mendefinisikan kelas abstrak
abstract class Hewan {
    public $nama;

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

    // Metode abstrak (tidak memiliki implementasi)
    abstract public function suara();

    // Metode konkret (memiliki implementasi)
    public function makan() {
        return $this->nama . " sedang makan.";
    }
}

// Kelas Kucing mewarisi kelas Hewan dan
mengimplementasikan metode abstrak
class Kucing extends Hewan {
    // Implementasi metode abstrak
    public function suara() {
        return "Meow!";
    }
}
```

```

}

// Kelas Anjing mewarisi kelas Hewan dan
mengimplementasikan metode abstrak
class Anjing extends Hewan {
    // Implementasi metode abstrak
    public function suara() {
        return "Woof!";
    }
}

// Membuat objek dari kelas Kucing dan Anjing
$ kucing = new Kucing("Kitty");
echo $ kucing->suara() . "\n"; // Output: Meow!
echo $ kucing->makan() . "\n"; // Output: Kitty sedang
makan.

$ anjing = new Anjing("Buddy");
echo $ anjing->suara() . "\n"; // Output: Woof!
echo $ anjing->makan() . "\n"; // Output: Buddy sedang
makan.

?>

```

### Kelas Abstrak (Hewan):

- **abstract class Hewan:** Mendeklarasikan kelas Hewan sebagai kelas abstrak.
- **public function \_\_construct(\$nama):** Konstruktor untuk menginisialisasi properti nama.
- **abstract public function suara():** Metode abstrak yang harus diimplementasikan oleh kelas turunan.
- **public function makan():** Metode konkret dengan implementasi yang dapat digunakan oleh kelas turunan.

### Kelas Turunan (Kucing dan Anjing):

- **class Kucing extends Hewan** dan **class Anjing extends Hewan:** Kelas Kucing dan Anjing mewarisi dari Hewan dan menyediakan implementasi konkret untuk metode suara().

## Penggunaan:

- **Membuat Objek:** Membuat objek dari kelas Kucing dan Anjing, dan memanggil metode suara() serta makan().
- **Output:** Metode suara() memberikan output spesifik untuk masing-masing kelas, sementara metode makan() memberikan output umum dari kelas abstrak.

```
coba.php > ...
22 class Kucing extends Hewan {
23     public function suara() {
24         return "Meow!";
25     }
26 }
27
28
29 // Kelas Anjing mewarisi kelas Hewan dan mengimplementasikan metode abstrak
    1 reference | 0 implementations
30 class Anjing extends Hewan {
31     // Implementasi metode abstrak
    3 references | 0 overrides | prototype
32     public function suara() {
33         return "Woof!";
34     }
35 }
36
37 // Membuat objek dari kelas Kucing dan Anjing
38 $kucing = new Kucing("Kitty");
39 echo $kucing->suara() . "\n"; // Output: Meow!
40 echo $kucing->makan() . "\n"; // Output: Kitty sedang makan.
41
42 $anjing = new Anjing("Buddy");
43 echo $anjing->suara() . "\n"; // Output: Woof!
44 echo $anjing->makan() . "\n"; // Output: Buddy sedang makan.
45
46 ?>
47
```

PROBLEMS 7 OUTPUT DEBUG CONSOLE TERMINAL PORTS

Meow!

Kitty sedang makan.  
Woof!  
Buddy sedang makan.

### Kode program method :

```
<?php
// Mendefinisikan kelas abstrak dengan metode abstrak
abstract class Hewan {
    public $nama;

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

    // Metode abstrak yang harus diimplementasikan
    oleh kelas turunan
    abstract public function suara();

    // Metode konkret (memiliki implementasi)
    public function makan() {
        return $this->nama . " sedang makan.";
    }

    // Metode konkret dengan parameter
    public function tidur($durasi) {
        return $this->nama . " tidur selama " .
        $durasi . " jam.";
    }
}

// Kelas Kucing mewarisi kelas Hewan dan
mengimplementasikan metode abstrak
class Kucing extends Hewan {
    // Implementasi metode abstrak
    public function suara() {
        return "Meow!";
    }

    // Implementasi metode konkret tambahan jika
    diperlukan
    public function main() {
```

```

        return $this->nama . " sedang bermain.";
    }
}

// Kelas Anjing mewarisi kelas Hewan dan
mengimplementasikan metode abstrak
class Anjing extends Hewan {
    // Implementasi metode abstrak
    public function suara() {
        return "Woof!";
    }

    // Implementasi metode konkret tambahan jika
    diperlukan
    public function jalan() {
        return $this->nama . " sedang jalan-jalan.";
    }
}

// Membuat objek dari kelas Kucing dan Anjing
$kucing = new Kucing("Kitty");
echo $kucing->suara() . "\n"; // Output: Meow!
echo $kucing->makan() . "\n"; // Output: Kitty sedang
makan.
echo $kucing->tidur(5) . "\n"; // Output: Kitty tidur
selama 5 jam.
echo $kucing->main() . "\n"; // Output: Kitty sedang
bermain.

$anjing = new Anjing("Buddy");
echo $anjing->suara() . "\n"; // Output: Woof!
echo $anjing->makan() . "\n"; // Output: Buddy sedang
makan.
echo $anjing->tidur(8) . "\n"; // Output: Buddy tidur
selama 8 jam.
echo $anjing->jalan() . "\n"; // Output: Buddy sedang
jalan-jalan.

```

```
40 class Anjing extends Hewan {
42     public function suara() {
43         return "woof!";
44     }
45
46     // Implementasi metode konkret tambahan jika diperlukan
47     // 1 reference | 0 overrides
48     public function jalan() {
49         return $this->nama . " sedang jalan-jalan.";
50     }
51 }
52
53 // Membuat objek dari kelas Kucing dan Anjing
54 $kucing = new Kucing("Kitty");
55 echo $kucing->suara() . "\n"; // Output: Meow!
56 echo $kucing->makan() . "\n"; // Output: Kitty sedang makan.
57 echo $kucing->tidur(5) . "\n"; // Output: Kitty tidur selama 5 jam.
58 echo $kucing->main() . "\n"; // Output: Kitty sedang bermain.
59
60 $anjing = new Anjing("Buddy");
61 echo $anjing->suara() . "\n"; // Output: Woof!
62 echo $anjing->makan() . "\n"; // Output: Buddy sedang makan.
63 echo $anjing->tidur(8) . "\n"; // Output: Buddy tidur selama 8 jam.
64 echo $anjing->jalan() . "\n"; // Output: Buddy sedang jalan-jalan.
65
66 ?>
```

PROBLEMS 7 OUTPUT DEBUG CONSOLE TERMINAL PORTS

Kitty sedang bermain.  
Woof!  
Buddy sedang makan.  
Buddy tidur selama 8 jam.  
Buddy sedang jalan-jalan.

?>



## 14. Interface

Objek interface memungkinkan kita untuk membuat kode yang menentukan metod mana yang akan diimplementasikan tanpa harus mendefinisikan bagaimana metod tersebut akan bekerja (hanya nama metod saja). Interface didefinisikan dengan “interface” keyword, mirip dengan deklarasi class biasa, hanya saja definisi atau detail metod tidak di tuliskan. Seluruh metod yang di deklarasikan pada interface haruus memiliki modifier “public”.

Untuk mengimplementasikan sebuah interface kita dapat menggunakan “implement” keyword. Seluruh method yang ada pada interface harus diimplementasikan seluruhnya. Sebuah class bisa mengimplementasikan lebih dari satuu interface

Catatan :

- Class tidak bisa mengimplementasikan dua interface yang mempunyai nama method yang sama.
- Interface bisa diwariskan seperti class menggunakan “extends”.

Class yang mengimplementasikan interface harus menggunakan metod-metod yang ada pada interface tersebut dengan nama yang sama persis.

### Kode Program :

- File HewanInterface.php :

```
<?php

// Mendefinisikan interface
interface HewanInterface {
    // Deklarasi metode tanpa implementasi
    public function suara();
    public function makan();
}

?>
```

- File Index.php :

```
<?php

// Sertakan file yang berisi interface
require_once 'HewanInterface.php';

// Kelas Kucing mengimplementasikan
interface HewanInterface
class Kucing implements HewanInterface {
    private $nama;

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

    // Implementasi metode dari interface
    public function suara() {
        return "Meow!";
    }

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

// Kelas Anjing mengimplementasikan
interface HewanInterface
class Anjing implements HewanInterface {
    private $nama;

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

    // Implementasi metode dari interface
```

```

        public function suara() {
            return "Woof!";
        }

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

    // Membuat objek dari kelas Kucing dan
    Anjing
    $kucing = new Kucing("Kitty");
    echo $kucing->suara() . "\n"; // Output:
    Meow!
    echo $kucing->makan() . "\n"; // Output:
    Kitty sedang makan.

    $anjing = new Anjing("Buddy");
    echo $anjing->suara() . "\n"; // Output:
    Woof!
    echo $anjing->makan() . "\n"; // Output:
    Buddy sedang makan.

    ?>

```

#### File HewanInterface.php:

- Mendefinisikan interface HewanInterface yang hanya mendeklarasikan metode suara() dan makan() tanpa implementasi. File ini harus diletakkan di tempat yang dapat diakses oleh file yang memerlukannya.

#### File index.php:

- **require\_once 'HewanInterface.php';** Mengimpor file interface sehingga dapat digunakan dalam kelas.
- **class Kucing implements HewanInterface:** Kelas Kucing harus mengimplementasikan metode suara() dan makan() yang didefinisikan dalam interface HewanInterface.

- **class Anjing implements HewanInterface:** Kelas Anjing juga mengimplementasikan metode suara() dan makan().
- **Membuat objek dan memanggil metode:** Objek dari kelas Kucing dan Anjing dibuat, dan metode suara() serta makan() dipanggil untuk masing-masing objek.

```

26  class Anjing implements HewanInterface {
32  }
33
34      // Implementasi metode dari interface
35      2 references | 0 overrides
36      public function suara() {
37          return "Woof!";
38      }
39
40      2 references | 0 overrides
41      public function makan() {
42          return $this->nama . " sedang makan.";
43      }
44  }
45
46      // Membuat objek dari kelas Kucing dan Anjing
47      $kucing = new Kucing("Kitty");
48      echo $kucing->suara() . "\n"; // Output: Meow!
49      echo $kucing->makan() . "\n"; // Output: Kitty sedang makan.
50
51      $anjing = new Anjing("Buddy");
52      echo $anjing->suara() . "\n"; // Output: Woof!
53      echo $anjing->makan() . "\n"; // Output: Buddy sedang makan.
54  }>

```

PROBLEMS 7

OUTPUT

DEBUG CONSOLE

TERMINAL

PORTS

```

Meow!
Kitty sedang makan.
Woof!
Buddy sedang makan.

```

```
HewanInterface.php > ...
1  <?php
2
3  // Mendefinisikan interface
   2 references | 3 implementations
4  interface HewanInterface {
5      // Deklarasi metode tanpa implementasi
   2 references | 3 overrides
6      public function suara();
   2 references | 2 overrides
7      public function makan();
8  }
9
10 ?>
11
```

## 15. Namespace

Namespace di PHP adalah fitur yang digunakan untuk mengelompokkan kode ke dalam ruang lingkup tertentu, mencegah konflik nama dan mempermudah pengelolaan kode yang besar. Namespace memungkinkan Anda untuk mendefinisikan kelas, interface, dan fungsi di dalam konteks yang berbeda tanpa konflik nama.

### Kode program index.php :

```
<?php

// Menggunakan autoloader atau sertakan file yang
diperlukan
require_once 'Hewan.php';

// Mengimpor kelas dari namespace
use HewanNamespace\Kucing;
use HewanNamespace\Anjing;

// Membuat objek dari kelas Kucing dan Anjing
$ kucing = new Kucing("Kitty");
echo $ kucing->suara() . "\n"; // Output: Meow!
echo $ kucing->makan() . "\n"; // Output: Kitty
sedang makan.

$ anjing = new Anjing("Buddy");
echo $ anjing->suara() . "\n"; // Output: Woof!
echo $ anjing->makan() . "\n"; // Output: Buddy
sedang makan.

?>
```

### Kode program Hewan.php :

```
<?php

namespace HewanNamespace;
```

```

// Sertakan file yang berisi interface
require_once 'HewanInterface.php';

// Kelas Kucing mengimplementasikan interface
HewanInterface
class Kucing implements HewanInterface {
    private $nama;

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

    // Implementasi metode dari interface
    public function suara() {
        return "Meow!";
    }

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

// Kelas Anjing mengimplementasikan interface
HewanInterface
class Anjing implements HewanInterface {
    private $nama;

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

    // Implementasi metode dari interface
    public function suara() {
        return "Woof!";
    }

    public function makan() {

```

```

        return $this->nama . " sedang makan.";
    }
}

?>

```

### **Kode Program HewanNamespace.php :**

```

<?php

namespace HewanNamespace;

// Mendefinisikan interface dengan namespace
interface HewanInterface {
    // Deklarasi metode tanpa implementasi
    public function suara();
    public function makan();
}

?>

```

**Namespace:** Mengelompokkan kelas, interface, dan fungsi dalam ruang lingkup tertentu untuk menghindari konflik nama.

**Deklarasi Namespace:** Menentukan namespace di awal file dengan namespace.

**Penggunaan Namespace:** Mengimpor kelas atau interface dari namespace dengan use di file utama.



```

Hewan.php > ...
1  <?php
2
3  namespace HewanNamespace;
4
5  // Sertakan file yang berisi interface
6  require_once 'HewanInterface.php';
7
8  // Kelas Kucing mengimplementasikan interface HewanInterface
9  class Kucing implements HewanInterface {
10     private $nama;
11
12     // Konstruktor
13     public function __construct($nama) {
14         $this->nama = $nama;
15     }
16
17     // Implementasi metode dari interface
18     public function suara() {
19         return "Meow!";
20     }
21
22     public function makan() {
23         return $this->nama . " sedang makan.";

```

```

HewanInterface.php > ...
1  <?php
2
3  namespace HewanNamespace;
4
5  // Mendefinisikan interface dengan namespace
6  interface HewanInterface {
7      // Deklarasi metode tanpa implementasi
8      public function suara();
9      public function makan();
10 }
11
12 }
13

```

```
index.php > ...
1  <?php
2
3  // Menggunakan autoloader atau sertakan file yang diperlukan
4  require_once 'Hewan.php';
5
6  // Mengimpor kelas dari namespace
7  use HewanNamespace\Kucing;
8  use HewanNamespace\Anjing;
9
10 // Membuat objek dari kelas Kucing dan Anjing
11 $kucing = new Kucing("Kitty");
12 echo $kucing->suara() . "\n"; // Output: Meow!
13 echo $kucing->makan() . "\n"; // Output: Kitty sedang makan.
14
15 $anjing = new Anjing("Buddy");
16 echo $anjing->suara() . "\n"; // Output: Woof!
17 echo $anjing->makan() . "\n"; // Output: Buddy sedang makan.
18
19 >>
20
```

PROBLEMS 7 OUTPUT DEBUG CONSOLE TERMINAL PORTS

Meow!  
Kitty sedang makan.  
Woof!  
Buddy sedang makan.

## BAGIAN 2

Project Akhir Minggu :

### CRUD index.php

```
<?php

require_once "app/Buku.php";
require_once "views/show.php";

$buku =new Buku();
Show::bookDisplay($buku->getData());
```

### buku.php

Saya memiliki sebuah kelas bernama Buku yang berfungsi untuk mengelola data buku dalam database. Kelas ini berisi beberapa metode yang memungkinkan saya untuk melakukan operasi dasar seperti membaca, menambah, memperbarui, dan menghapus data buku. Berikut penjelasan setiap bagian dari kode ini:

#### Koneksi Database:

```
php
public function __construct() {
    try {
        $this->db = new
PDO("mysql:host=localhost;
dbname=perpustakaan", "root", "");
        $this->db-
>setAttribute(PDO::ATTR_ERRMODE,
PDO::ERRMODE_EXCEPTION);
    } catch (PDOException $e) {
        die("Error: " . $e->getMessage());
    }
}
```

- Pada bagian ini, saya membuat koneksi ke database MySQL menggunakan PDO.

- Saya menghubungkan ke database bernama perpustakaan dengan username root dan tanpa password.
- Jika koneksi gagal, saya menangkap pengecualian (PDOException) dan menampilkan pesan kesalahan.

### **Menghapus Buku:**

```
php
public function deleteBuku($index) {
    $stmt = $this->db->prepare("DELETE FROM buku
WHERE `index` = ?");
    $stmt->execute([$index]);
}
```

- Metode ini digunakan untuk menghapus buku dari tabel buku berdasarkan index buku.
- Saya mempersiapkan perintah SQL untuk menghapus baris yang memiliki index tertentu dan mengeksekusinya.

### **Membaca Data Buku:**

```
php
public function getData() {
    $stmt = $this->db->query("SELECT * FROM buku");
    return $stmt->fetchAll(PDO::FETCH_ASSOC);
}
```

- Metode ini digunakan untuk mengambil semua data dari tabel buku.
- Saya menjalankan query SQL untuk memilih semua kolom dari tabel buku dan mengembalikan hasilnya dalam bentuk array asosiatif.

### Menambah Buku:

```
php
public function addBuku($judul, $pengarang,
$tebal_buku, $penerbit, $stok) {
    $stmt = $this->db->prepare("INSERT INTO buku
(judul, pengarang, tebal_buku, penerbit, stok) VALUES
(?, ?, ?, ?, ?)");
    $stmt->execute([$judul, $pengarang, $tebal_buku,
$penerbit, $stok]);
}
```

- Metode ini digunakan untuk menambahkan buku baru ke dalam tabel buku.
- Saya mempersiapkan perintah SQL untuk memasukkan data buku ke dalam tabel, lalu mengeksekusi perintah dengan parameter yang diberikan.

### Mendapatkan Buku Berdasarkan Index:

```
php
public function getBukuByIndex($index) {
    $stmt = $this->db->prepare("SELECT * FROM buku
WHERE `index` = ?");
    $stmt->execute([$index]);
    return $stmt->fetch(PDO::FETCH_ASSOC);
}
```

- Metode ini digunakan untuk mengambil data buku tertentu berdasarkan index.
- Saya mempersiapkan perintah SQL untuk memilih buku dengan index yang spesifik dan mengembalikan hasilnya dalam bentuk array asosiatif.

### Mengimpor Kelas Buku:

```
php
require_once 'buku.php'; // Pastikan file buku.php ada di
direktori yang sama
```

- Di bagian ini, saya mengimpor kelas Buku dari file buku.php. Kelas ini berisi metode untuk mengelola data buku di database.

### Instansiasi Kelas Buku:

php

```
$buku = new Buku();
```

- Saya membuat objek Buku yang akan digunakan untuk memanggil metode-metode dari kelas Buku.

### Menangani Operasi CRUD:

php

```
if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    if (isset($_POST['add'])) {
        // Menambah buku
        $buku->addBuku($_POST['judul'],
            $_POST['pengarang'], $_POST['tebal_buku'],
            $_POST['penerbit'], $_POST['stok']);
    } elseif (isset($_POST['update'])) {
        // Memperbarui buku
        $buku->updateBuku($_POST['index'],
            $_POST['judul'], $_POST['pengarang'],
            $_POST['tebal_buku'], $_POST['penerbit'],
            $_POST['stok']);
    } elseif (isset($_POST['delete'])) {
        // Menghapus buku
        $buku->deleteBuku($_POST['index']);
    }
}
```

- Bagian ini memeriksa apakah permintaan HTTP adalah POST, yang berarti formulir telah dikirimkan.

- Tergantung pada tombol yang diklik (add, update, atau delete), saya memanggil metode yang sesuai dari objek Buku untuk menambah, memperbarui, atau menghapus buku.

### **Mendapatkan Data Buku:**

php

```
$dataBuku = $buku->getData();
```

- Saya mengambil semua data buku dari database dengan memanggil metode `getData()` dari objek Buku.

### **Mendapatkan Buku untuk Diperbarui:**

php

```
$bukuUntukUpdate = null;
```

```
if (isset($_GET['edit'])) {
```

```
    $bukuUntukUpdate
```

```
=
```

```
$buku-
```

```
>getBukuByIndex($_GET['edit']);
```

```
}
```

### **Bagian HTML dan CSS:**

- Bagian HTML ini menyediakan antarmuka pengguna untuk menambahkan, memperbarui, dan menampilkan data buku.

- **Formulir Tambah Buku:**

```
<form method="POST">
```

```
    <label>Judul: <input type="text" name="judul"
required></label>
```

```
    <label>Pengarang: <input type="text"
name="pengarang" required></label>
```

```
    <label>Tebal Buku: <input type="text"
name="tebal_buku" required></label>
```

```

        <label>Penerbit:      <input          type="text"
name="penerbit" required></label>
        <label>Stok: <input type="number" name="stok"
required></label>
        <input type="submit" name="add" value="Tambah
Buku">
</form>

```

- o Formulir ini memungkinkan saya untuk menambahkan buku baru dengan mengisi berbagai detail buku dan mengklik tombol "Tambah Buku".

### Formulir Update Buku:

```

<?php if ($bukuUntukUpdate): ?>
<form method="POST">
    <input type="hidden" name="index" value="<?php
echo    htmlspecialchars($bukuUntukUpdate['index']);
?>">
    <label>Judul: <input type="text" name="judul"
value="<?php          echo
htmlspecialchars($bukuUntukUpdate['judul']);    ?>"
required></label>
    <label>Pengarang:      <input          type="text"
name="pengarang"          value="<?php          echo
htmlspecialchars($bukuUntukUpdate['pengarang']); ?>"
required></label>
    <label>Tebal    Buku:      <input          type="text"
name="tebal_buku"          value="<?php          echo

```



```

htmlspecialchars($bukuUntukUpdate['tebal_buku']);
?>" required></label>
    <label>Penerbit:          <input          type="text"
name="penerbit"              value="<?php          echo
htmlspecialchars($bukuUntukUpdate['penerbit']);  ?>"
required></label>
    <label>Stok: <input type="number" name="stok"
value="<?php                      echo
htmlspecialchars($bukuUntukUpdate['stok']);      ?>"
required></label>
    <input type="submit" name="update" value="Update
Buku">
</form>
<?php endif; ?>

```

- Formulir ini memungkinkan saya untuk memperbarui buku yang sudah ada dengan mengisi detail buku yang ingin diperbarui dan mengklik tombol "Update Buku".

### **Tabel Daftar Buku:**

```

<table>
  <thead>
    <tr>
      <th>Index</th>
      <th>Judul</th>
      <th>Pengarang</th>
      <th>Tebal Buku</th>
      <th>Penerbit</th>
      <th>Stok</th>
      <th>Aksi</th>
    </tr>
  </thead>

```

```

<tbody>
  <?php
  $index = 1;
  foreach ($dataBuku as $bukuItem): ?>
    <tr>
      <td><?php echo $index++; ?></td>
      <td><?php
      htmlspecialchars($bukuItem['judul']); ?></td>
      <td><?php
      htmlspecialchars($bukuItem['pengarang']); ?></td>
      <td><?php
      htmlspecialchars($bukuItem['tebal_buku']); ?></td>
      <td><?php
      htmlspecialchars($bukuItem['penerbit']); ?></td>
      <td><?php
      htmlspecialchars($bukuItem['stok']); ?></td>
      <td>
        <a href="?edit=<?php
        htmlspecialchars($bukuItem['index']); ?>">Edit</a>
        <form
        method="POST"
        style="display:inline;">
          <input
          type="hidden"
          name="index"
          value="<?php
          htmlspecialchars($bukuItem['index']); ?>">
          <input
          type="submit"
          name="delete"
          value="Hapus"
          onclick="return
          confirm('Anda yakin ingin menghapus buku ini?');">
        </form>
      </td>
    </tr>
  <?php endforeach; ?>
</tbody>
</table>

```

- Tabel ini menampilkan daftar semua buku dengan kolom untuk index, judul, pengarang, tebal buku, penerbit, dan stok.
- Setiap baris juga menyediakan tautan untuk mengedit buku dan formulir untuk menghapus buku.

## Mengubah/Mereset Auto-Increment

```
$index = 1;  
<td><?php echo $index++; ?></td>
```

## Gambar Visual Studio Code

```
<table>  
  <thead>  
    <tr>  
      <th>Index</th>  
      <th>Judul</th>  
      <th>Pengarang</th>  
      <th>Tebal Buku</th>  
      <th>Penerbit</th>  
      <th>Stok</th>  
      <th>Aksi</th>  
    </tr>  
  </thead>  
  <tbody>  
    <?php foreach ($dataBuku as $bukuItem): ?>  
      <tr>  
        <td><?php echo htmlspecialchars($bukuItem['index']); ?></td>  
        <td><?php echo htmlspecialchars($bukuItem['judul']); ?></td>  
        <td><?php echo htmlspecialchars($bukuItem['pengarang']); ?></td>  
        <td><?php echo htmlspecialchars($bukuItem['tebal_buku']); ?></td>  
        <td><?php echo htmlspecialchars($bukuItem['penerbit']); ?></td>  
        <td><?php echo htmlspecialchars($bukuItem['stok']); ?></td>  
        <td>  
          <a href="#edit"><?php echo htmlspecialchars($bukuItem['index']); ?><Edit</a>  
          <form method="POST" style="display:inline">  
            <input type="hidden" name="index" value="#<?php echo htmlspecialchars($bukuItem['index']); ?>">  
            <input type="submit" name="delete" value="hapus" onclick="return confirm('Anda yakin ingin menghapus buku ini?');">  
          </form>  
        </td>  
      </tr>  
    <?php endforeach; ?>  
  </tbody>  
</table>
```

```
31  
32 // DELETE  
33 1 reference | 0 overrides  
34 public function deleteBuku($index) {  
35     $stmt = $this->db->prepare("DELETE FROM buku WHERE 'index' = ?");  
36     $stmt->execute([$index]);  
37 }  
38  
39 // Get a single book by index  
40 1 reference | 0 overrides  
41 public function getBukuByIndex($index) {  
42     $stmt = $this->db->prepare("SELECT * FROM buku WHERE 'index' = ?");  
43     $stmt->execute([$index]);  
44     return $stmt->fetch(PDO::FETCH_ASSOC);  
45 }  
46
```

```

app > index.php > ...
1 <?php
2 require_once 'buku.php'; // Pastikan file buku.php ada di direktori yang sama
3
4 $buku = new Buku();
5
6 // Menangani operasi CREATE dan UPDATE
7 if ($_SERVER['REQUEST_METHOD'] === 'POST') {
8     if (isset($_POST['add'])) {
9         // Menambah buku
10         $buku->addBuku($_POST['judul'], $_POST['pengarang'], $_POST['tebal_buku'], $_POST['penerbit'], $_POST['stok']);
11     } elseif (isset($_POST['update'])) {
12         // Memperbarui buku
13         $buku->updateBuku($_POST['index'], $_POST['judul'], $_POST['pengarang'], $_POST['tebal_buku'], $_POST['penerbit'], $_POST['stok']);
14     } elseif (isset($_POST['delete'])) {
15         // Menghapus buku
16         $buku->deleteBuku($_POST['index']);
17     }
18 }
19
20 // Mendapatkan data buku
21 $dataBuku = $buku->getData();
22
23 // Mendapatkan buku untuk diupdate jika ada
24 $bukuuntukupdate = null;
25 if (isset($_GET['edit'])) {
26     $bukuuntukupdate = $buku->getBukuByIndex($_GET['edit']);
27 }
28 ?>

```

```

<body>
<div class="container">
<h1>Daftar Buku</h1>

<!-- Formulir untuk menambahkan buku -->
<h2>Tambah Buku</h2>
<form method="POST">
    <label>Judul: <input type="text" name="judul" required></label>
    <label>Pengarang: <input type="text" name="pengarang" required></label>
    <label>Tebal Buku: <input type="text" name="tebal_buku" required></label>
    <label>Penerbit: <input type="text" name="penerbit" required></label>
    <label>Stok: <input type="number" name="stok" required></label>
    <input type="submit" name="add" value="Tambah Buku">
</form>

<!-- Formulir untuk memperbarui buku -->
<?php if ($bukuuntukupdate): ?>
<h2>Update Buku</h2>
<form method="POST">
    <input type="hidden" name="index" value="php echo htmlspecialchars($bukuuntukupdate['index']); ?">
    <label>Judul: <input type="text" name="judul" value="php echo htmlspecialchars($bukuuntukupdate['judul']); ?" required></label>
    <label>Pengarang: <input type="text" name="pengarang" value="php echo htmlspecialchars($bukuuntukupdate['pengarang']); ?" required></label>
    <label>Tebal Buku: <input type="text" name="tebal_buku" value="php echo htmlspecialchars($bukuuntukupdate['tebal_buku']); ?" required></label>
    <label>Penerbit: <input type="text" name="penerbit" value="php echo htmlspecialchars($bukuuntukupdate['penerbit']); ?" required></label>
    <label>Stok: <input type="number" name="stok" value="php echo htmlspecialchars($bukuuntukupdate['stok']); ?" required></label>
    <input type="submit" name="update" value="Update Buku">
</form>
<?php endif; ?>

```

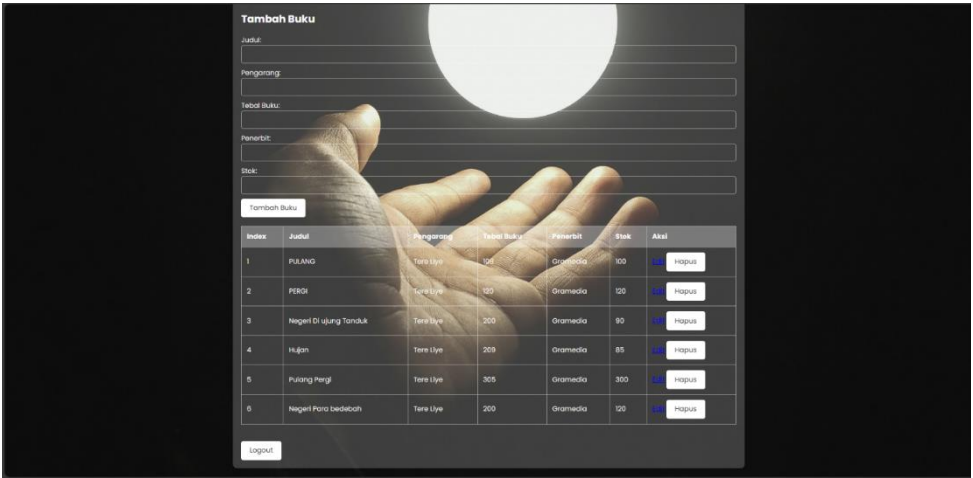
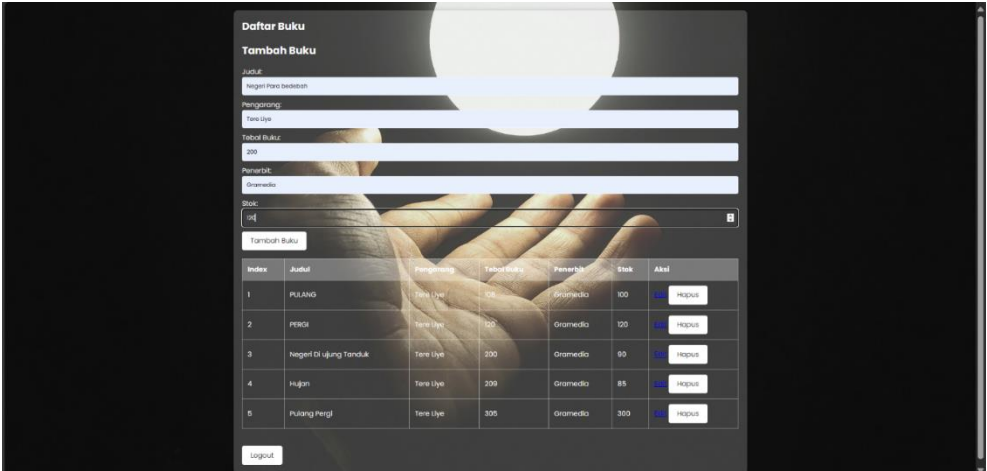
```

app > buku.php > PHP > Buku
1 <?php
1 reference | 0 implementations
2 class Buku {
7 references
3     private $db;
4
1 reference | 0 overrides
5     public function __construct() {
6         try {
7             $this->db = new PDO("mysql:host=localhost; dbname=perpustakaan", "root", "");
8             $this->db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
9         } catch (PDOException $e) {
10             die("Error: " . $e->getMessage());
11         }
12     }
13
14     // READ
1 reference | 0 overrides
15     public function getData() {
16         $stmt = $this->db->query("SELECT * FROM buku");
17         return $stmt->fetchAll(PDO::FETCH_ASSOC);
18     }
19
20     // CREATE
1 reference | 0 overrides
21     public function addBuku($judul, $pengarang, $tebal_buku, $penerbit, $stok) {
22         $stmt = $this->db->prepare("INSERT INTO buku (judul, pengarang, tebal_buku, penerbit, stok) VALUES (?, ?, ?, ?, ?)");
23         $stmt->execute([$judul, $pengarang, $tebal_buku, $penerbit, $stok]);
24     }
25
26     // UPDATE
1 reference | 0 overrides
27     public function updateBuku($index, $judul, $pengarang, $tebal_buku, $penerbit, $stok) {
28         $stmt = $this->db->prepare("UPDATE buku SET judul = ?, pengarang = ?, tebal_buku = ?, penerbit = ?, stok = ? WHERE `index` = ?");
29         $stmt->execute([$judul, $pengarang, $tebal_buku, $penerbit, $stok, $index]);
30     }
31

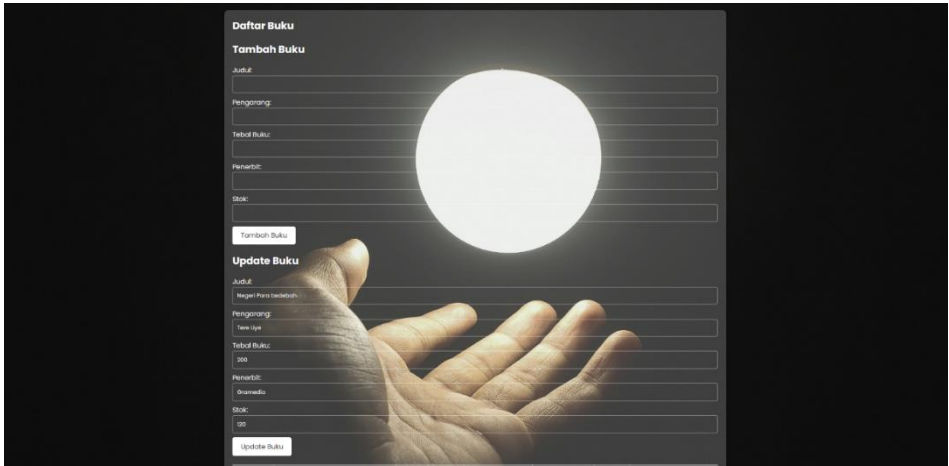
```

Visual Web

Create



Update



Index	Judul	Pengarang	Tebal Buku	Penerbit	Stok	Aksi
1	PULANG	Tere Liye	108	Gramedia	100	<a href="#">100</a> <button>Hapus</button>
2	PERGI	Tere Liye	120	Gramedia	120	<a href="#">120</a> <button>Hapus</button>
3	Negeri Di ujung Tanduk	Tere Liye	200	Gramedia	90	<a href="#">100</a> <button>Hapus</button>
4	Hujan	Tere Liye	209	Gramedia	85	<a href="#">100</a> <button>Hapus</button>
5	Pulang Pergi	Tere Liye	305	Gramedia	300	<a href="#">100</a> <button>Hapus</button>
6	Negeri Para bedebah	Tere Liye	200	Gramedia	300	<a href="#">100</a> <button>Hapus</button>

## Read

### Daftar Buku

#### Tambah Buku

Judul:

Pengarang:

Tebal Buku:

Penerbit:

Stok:

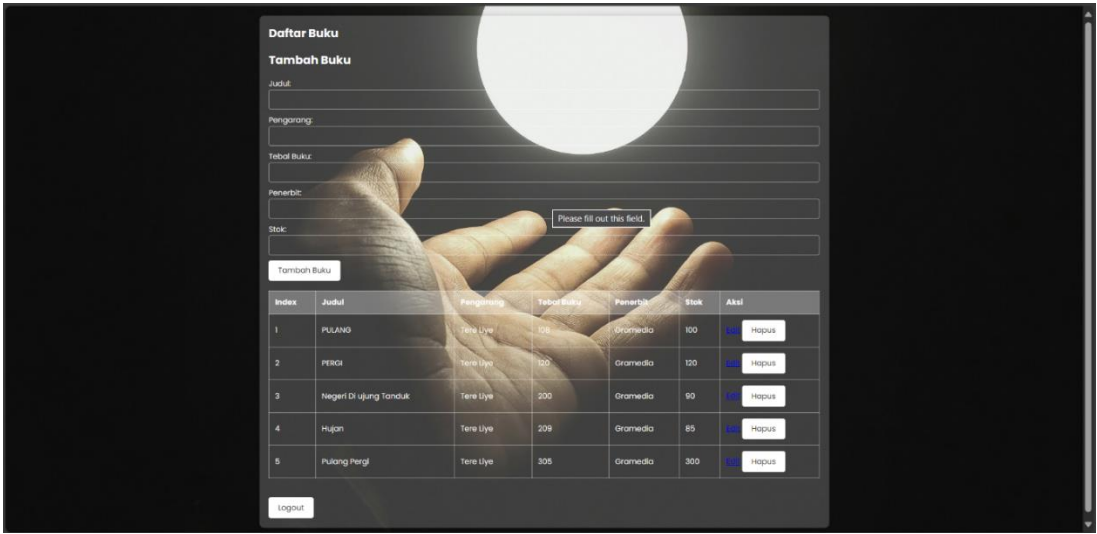
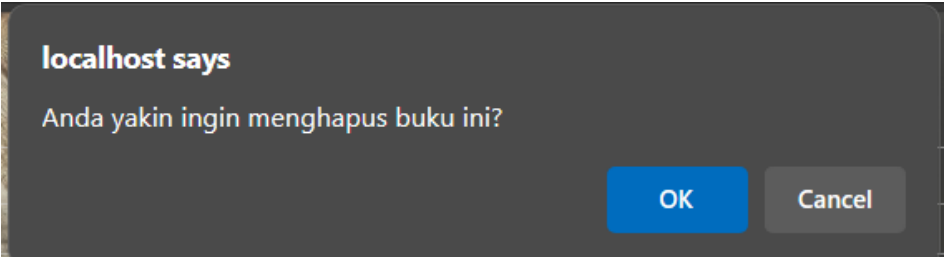
Tambah Buku

Index	Judul	Pengarang	Tebal Buku	Penerbit	Stok	Aksi
1	PULANG	Tere Liye	105	Cramedia	100	<a href="#">Edit</a> <a href="#">Hapus</a>
2	PRIDE	Tere Liye	100	Cramedia	120	<a href="#">Edit</a> <a href="#">Hapus</a>
3	Negeri Di Ujung Tanduk	Tere Liye	200	Cramedia	90	<a href="#">Edit</a> <a href="#">Hapus</a>
4	Hujan	Tere Liye	309	Cramedia	85	<a href="#">Edit</a> <a href="#">Hapus</a>
5	Pulang Pergi	Tere Liye	305	Cramedia	300	<a href="#">Edit</a> <a href="#">Hapus</a>

Logout



Delete



## PENUTUP

### III.1 Kesimpulan

Dalam laporan ini, telah dibahas berbagai konsep penting dalam pemrograman berorientasi objek (OOP) dengan menggunakan PHP, termasuk kelas, objek, pengakses (access modifiers), konstruktor, metode statis, dan interface. Berikut adalah beberapa poin kunci yang disimpulkan dari pembahasan:

1. **Kelas dan Objek:** Kelas adalah cetak biru untuk membuat objek, sedangkan objek adalah instansi dari kelas yang memiliki atribut dan metode untuk memodelkan entitas dalam dunia nyata.
2. **Pengakses (Access Modifiers):** PHP menyediakan tiga jenis pengakses - public, protected, dan private - yang mengontrol visibilitas dan aksesibilitas atribut serta metode dalam kelas.
3. **Konstruktor:** Konstruktor adalah metode khusus dalam kelas yang dipanggil saat objek dibuat, digunakan untuk inisialisasi nilai properti objek.
4. **Metode Statis dan Properti Statis:** Metode dan properti statis adalah anggota kelas yang tidak tergantung pada instansi objek, melainkan pada kelas itu sendiri. Mereka berguna untuk data atau fungsionalitas yang umum untuk semua objek dari kelas tersebut.
5. **Interface:** Interface mendefinisikan kontrak yang harus diimplementasikan oleh kelas yang menggunakannya, memungkinkan pengembangan kode yang lebih modular dan fleksibel.
6. **Namespace:** Namespace membantu mengorganisir kode dengan mencegah konflik nama dan membuat kode lebih terstruktur dalam proyek yang besar.

### III.2 Saran

Untuk meningkatkan pemahaman dan penggunaan konsep OOP dalam proyek PHP, berikut beberapa saran yang dapat diterapkan:

1. **Penggunaan Namespace:** Selalu gunakan namespace untuk mengorganisir kode dalam proyek besar. Ini akan membantu menghindari konflik nama dan mempermudah pemeliharaan kode.

2. **Implementasi Interface:** Manfaatkan interface untuk mendefinisikan kontrak yang harus diikuti oleh kelas-kelas dalam aplikasi Anda. Ini akan memudahkan pengembangan dan pengujian kode.
3. **Dokumentasi Kode:** Dokumentasikan kode Anda secara menyeluruh, termasuk penjelasan tentang kelas, metode, dan properti, agar memudahkan pemahaman dan kolaborasi dengan pengembang lain.
4. **Praktik Baik dalam OOP:** Terapkan prinsip-prinsip desain OOP seperti enkapsulasi, pewarisan, dan polimorfisme secara konsisten untuk menghasilkan kode yang bersih dan mudah dikelola.
5. **Pengujian dan Debugging:** Selalu lakukan pengujian menyeluruh pada kode Anda untuk memastikan semua metode dan fungsionalitas berfungsi seperti yang diharapkan. Gunakan alat debugging untuk mengidentifikasi dan memperbaiki masalah yang mungkin muncul.