Nama : Ahmad Dwi Cahyadi

Npm : G1F022007

Kelas : A

Responsi

- conflict.php

data > conflict.php > ...

```php
<?php

// buat namespace data\satu
namespace data\satu {
// dengan class conflict
class conflict {
}
// class sample
class sample {
}
// class dummy
class dummy {
}
};
// buat namespace data\dua
namespace data\dua{
// dengan class conflict
class conflict{
}
};
```

Penjelasanya:

Pada Kode di atas membuat dua namespace, `data\satu` dan `data\dua`, masing-masing memiliki kelas `conflict` di dalamnya. Selain itu, `data\satu` juga memiliki kelas tambahan yaitu `sample` dan `dummy`. Jadi, secara singkat, kode ini mendefinisikan beberapa namespace dan kelas dalam struktur yang terorganisir.

- helper.php

```php
<?php

namespace Helper;

function helpMe()
{
    echo "HELP ME" . PHP_EOL;
}

const APPLICATION = "Belajar PHP OOP Ahmadl";
```

Penjelasanya:

Kode ini mengorganisir fungsionalitas terkait dalam namespace `Helper`. Di dalamnya, terdapat fungsi `helpMe()`, yang ketika dipanggil, akan menampilkan pesan "HELP ME" diikuti dengan karakter baris baru. Selain itu, ada konstanta `APPLICATION` dengan nilai "Belajar PHP OOP Ahmadl". Namespace ini dapat digunakan sebagai modul bantuan yang dapat diakses dari berbagai bagian program PHP, menyediakan keterbacaan dan pemeliharaan yang lebih baik.

- manager.php

```php
<?php

// buat kelas manager dengan properti nama dan function sayHello
class Manager
{
    var string $nama;

    function sayHello(string $nama)
    {
        echo "Hi $nama, my name is {$this->nama}" . PHP_EOL;
    }
}

// buat kelas VicePresident dengan extends manager
class VicePresident extends Manager
{

}
```

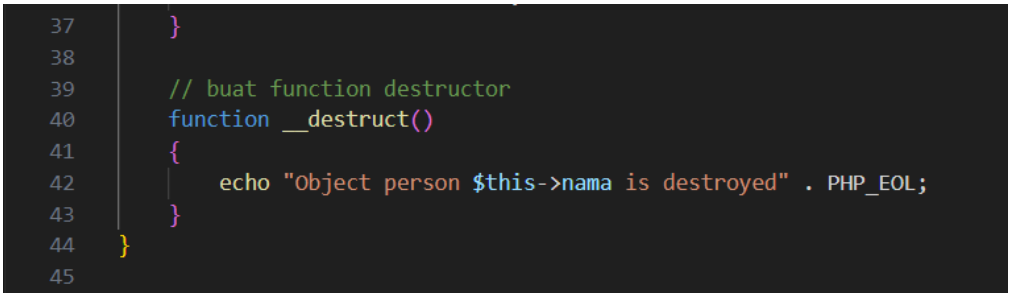
Penjelsanya:

Pada tampilan Kode di atas ini membuat dua kelas dalam paradigma pemrograman berorientasi objek. Pertama, ada kelas `Manager` dengan properti nama (`$nama`) dan fungsi `sayHello` yang mencetak pesan sapaan dengan nama manajer. Kemudian, kelas `VicePresident` dibuat yang mewarisi semua properti dan metode dari kelas `Manager`. Ini menciptakan struktur hirarki di mana `VicePresident` dapat menggunakan metode `sayHello` dari kelas `Manager` tanpa mendefinisikan kembali. Dengan cara ini, konsep pewarisan digunakan untuk mengorganisir dan mewariskan fungsionalitas antar kelas.

- person.php

data > person.php > Person

```php
<?php

// membuat kelas person
class Person{
    // membuat properti
    var string $nama;
    // gunakan nullable properti
    var ?string $alamat = null;
    // gunakan default value untuk properti
    var string $negara = "Indonesia";
    //  function sayHello
    function sayHello(string $nama){
        echo "Hei $nama" . PHP_EOL;
    }
    // function sayHello nullable dengan percabangan
    function sayHelloNull(?string $nama)
    {
        if (is_null($nama)) {
            echo "Hi, my nama is $this->nama" . PHP_EOL;
        } else {
            echo "Hi $nama, my nama is $this->nama" . PHP_EOL;
        }
    }

    // buat const author
    const AUTHOR = "Ahmad Dwi Cahyadi";
    // buat function info untuk self keyword
    function info()
    {
        echo "Author : " . self::AUTHOR . PHP_EOL;
    }
    // buat function constructor
    function __construct(string $nama, ?string $alamat)
    {
        $this->nama = $nama;
        $this->alamat = $alamat;
```



```
37      }
38
39      // buat function destructor
40      function __destruct()
41      {
42          echo "Object person $this->nama is destroyed" . PHP_EOL;
43      }
44  }
45
```

Penjelsanya :

Kode ini mendefinisikan kelas `Person` dalam PHP dengan beberapa fitur. Properti kelas mencakup `$nama` sebagai string, `$alamat` sebagai nullable string dengan nilai default null, dan `$negara` sebagai string dengan nilai default "Indonesia". Terdapat dua fungsi `sayHello`, satu untuk menerima parameter nama dan yang lainnya menerima nullable parameter nama dengan menggunakan percabangan. Konstanta `AUTHOR` didefinisikan sebagai konstanta kelas, dan fungsi `info` menggunakan kata kunci `self` untuk mengakses konstanta tersebut. Konstruktor menerima parameter `$nama` dan `$alamat` untuk menginisialisasi properti. Terakhir, ada fungsi destruktor yang mencetak pesan saat objek `Person` dihancurkan. Ini adalah kelas yang cukup lengkap dengan berbagai fitur, termasuk konstruktor, destruktor, dan penggunaan nullable properti.

- product.php

data > product.php > ...

```php
<?php

class Product
{
    protected string $name;
    protected int $price;

    public function __construct(string $name, int $price)
    {
        $this->name = $name;
        $this->price = $price;
    }

    public function getName(): string
    {
        return $this->name;
    }

    public function getPrice(): int
    {
        return $this->price;
    }
}

class ProductDummy extends Product
{

    public function info()
    {
        echo "Name $this->name" . PHP_EOL;
        echo "Price $this->price" . PHP_EOL;
    }

}
```

Penjelasanya :

Kode ini mendefinisikan dua kelas dalam PHP, yaitu `Product` dan `ProductDummy`. Kelas `Product` memiliki properti proteksi `$name` dan `$price`, serta konstruktor untuk menginisialisasi nilai properti tersebut. Metode `getName` dan `getPrice` digunakan untuk mengakses nilai properti tersebut. Kelas `ProductDummy` mewarisi kelas `Product` dan menambahkan metode `info` yang mencetak informasi nama dan harga produk. Dengan ini, `ProductDummy` dapat mengakses properti proteksi dari kelas induknya. Keseluruhan, ini adalah contoh penerapan pewarisan dan enkapsulasi dalam pemrograman berorientasi objek, di mana `ProductDummy` memperluas fungsionalitas dari kelas `Product`.

- programer.php

data > programmer.php > ...

```php
<?php

class Programmer
{

    public string $name;

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

}

class BackendProgrammer extends Programmer
{
}

class FrontendProgrammer extends Programmer
{
}

class Company
{
    public Programmer $programmer;
}


function sayHelloProgrammer(Programmer $programmer)
{
    if ($programmer instanceof BackendProgrammer) {
        echo "Hello Backend Programmer $programmer->name" . PHP_EOL;
    } else if ($programmer instanceof FrontendProgrammer) {
        echo "Hello Frontend Programmer $programmer->name" . PHP_EOL;
    } else if ($programmer instanceof Programmer) {
        echo "Hello Programmer $programmer->name" . PHP_EOL;
    }
```

Penjelasanya :

Kode ini menggambarkan konsep pewarisan dan polimorfisme dalam pemrograman berorientasi objek dengan menggunakan tiga kelas: `Programmer`, `BackendProgrammer`, dan `FrontendProgrammer`. Semua kelas programmer memiliki properti publik `$name` dan konstruktor untuk menginisialisasi nama programmer. Selain itu, ada fungsi `sayHelloProgrammer` yang menerima objek `Programmer` sebagai parameter dan memberikan salam berdasarkan jenis programmer, baik itu `BackendProgrammer`, `FrontendProgrammer`, atau `Programmer` umum. Ini memanfaatkan operator `instanceof` untuk menentukan jenis objek dan memberikan pesan salam yang sesuai. Keseluruhan, ini menciptakan hierarki kelas dengan kemampuan polimorfisme untuk

menangani berbagai jenis programmer dengan satu fungsi yang dapat digunakan secara fleksibel.

- shape.php

Penjelasanya :

Kode ini mendemonstrasikan konsep overriding dan penggunaan `parent::` dalam pemrograman berorientasi objek dengan menggunakan dua kelas, yaitu `Shape` dan `Rectangle`, yang berada dalam namespace `Data`. Kelas `Shape` memiliki metode `getCorner` yang mengembalikan nilai -1. Kelas `Rectangle` mewarisi dari `Shape` dan mengoverride metode `getCorner` untuk mengembalikan nilai 4. Selain itu,

`Rectangle` memiliki metode tambahan, `getParentCorner`, yang menggunakan `parent::getCorner()` untuk memanggil metode `getCorner` dari kelas induknya (`Shape`). Ini memberikan kemampuan untuk mengakses metode dari kelas induk dalam konteks kelas turunan. Keseluruhan, kode ini menunjukkan cara mengganti perilaku metode dalam kelas turunan dan menggunakan `parent::` untuk berinteraksi dengan metode dari kelas induk.

- constant.php

Penjelasanya :

Kode ini mencakup beberapa aspek dasar dalam PHP. Pertama, ia mengimpor kelas `Person` dari file `person.php`. Selanjutnya, kode ini mendefinisikan konstanta dengan `define()` untuk judul (`TITLE`) dan menggunakan `const` untuk versi aplikasi (`APP_VERSION`). Terakhir, kode ini mencetak hasil dari konstanta judul, versi aplikasi, dan konstanta kelas `Person` (`AUTHOR`). Ini menunjukkan penggunaan define, const, dan cara mengakses konstanta dari kelas yang diimpor.

Keseluruhan, kode ini adalah contoh sederhana penggunaan konstanta dalam PHP dan cara mengimpor dan mengakses kelas dari file eksternal.

- constraktor.php

Penjelasanya :

Kode ini mengilustrasikan proses pembuatan objek dari kelas `Person` setelah mengimpornya dari file eksternal `person.php`. Objek baru, `$ahmadl`, dibuat dengan menggunakan konstruktor kelas `Person` yang memerlukan dua parameter, yaitu nama ("Ahmad") dan alamat ("seluma"). Selanjutnya, kode menggunakan `var_dump` untuk menampilkan informasi lengkap tentang objek tersebut, termasuk propertinya. Ini memberikan gambaran rinci tentang struktur dan nilai dari objek yang baru dibuat. Keseluruhan, ini adalah contoh penerapan konsep objek dan penggunaan konstruktor dalam PHP.

- destructor.php

desturctor.php > ...

```php
<?php

// import data/person.php
require_once "data/Person.php";

// buat 2 object new person dengan parameter yang berbeda
$ahmadl = new Person("Ahmadl", "seluma");
$bunga = new Person("bunga", "bandung");

// tambahkan echo "Program Selesai" . PHP_EOL;
echo "Program Selesai" . PHP_EOL;

```

Penjelasanya :

Kode ini memanfaatkan kelas `Person` yang diimpor dari file eksternal `person.php` untuk membuat dua objek, `$ahmadl` dan `$bunga`, dengan parameter yang berbeda pada konstruktor. Objek-objek ini mewakili dua individu dengan nama dan alamat yang berbeda. Setelah pembuatan objek, kode menampilkan pesan "Program Selesai" menggunakan `echo`. Keseluruhan, ini adalah contoh penggunaan kelas dan objek dalam PHP untuk menciptakan dan mengelola instance yang berbeda dengan data yang berbeda pula.

- funcition.php

function.php > ...

```php
<?php

// import data/person.php
require_once "data/person.php";

// buat object baru dari kelas person
$person1 = new Person("Ahmadl","Seluma");

// panggil function
$person1->sayHello("Ahmadl");

```

Penjelasanya :

Kode ini menggunakan kelas `Person` yang diimpor dari file eksternal `person.php` untuk membuat objek baru dengan nama `$person1` dan menginisialisasinya dengan nama "Ahmadl" dan alamat "Seluma". Selanjutnya, kode memanggil metode `sayHello` pada objek `$person1` dengan memberikan parameter "Ahmadl". Metode tersebut mencetak pesan sapaan dengan menggunakan nama yang diberikan. Ini merupakan contoh sederhana dari bagaimana objek dan metode digunakan dalam PHP untuk berinteraksi dengan data yang dimiliki oleh suatu kelas.

- import.php

Penjelasanya :

Kode ini melakukan beberapa hal terkait namespace dan penggunaan fungsi dan konstanta. Pertama, mengimpor kelas `Conflict` dari file `conflict.php` dan fungsi serta konstanta dari file `helper.php`. Selanjutnya, menggunakan namespace dengan mendeklarasikan objek `$conflict1` dari kelas `Conflict` dan `$conflict2` dengan memberikan namespace lengkap. Menggunakan fungsi `helpMe()` dan mencetak nilai konstanta `APPLICATION`. Penggunaan namespace membuat pengelompokan dan pemisahan antara kelas dan fungsi yang berasal dari berbagai file, sementara kata kunci `use` digunakan untuk menyederhanakan pemanggilan kelas, fungsi, dan konstanta. Keseluruhan,

ini adalah contoh implementasi namespace dan penggunaan elemen-elemen dari file eksternal dalam PHP.

- importalias.php

importAlias.php > ...

```php
<?php

require_once "data/Conflict.php";
require_once "data/Helper.php";

use Data\satu\Conflict as Conflict1;
use Data\dua\Conflict as Conflict2;
use function Helper\helpMe as help;
use const Helper\APPLICATION as APP;

$conflict1 = new Conflict1();
$conflict2 = new Conflict2();

help();

echo APP . PHP_EOL;
```

Penjelasanya :

Kode ini menggunakan beberapa konsep PHP, seperti penggunaan namespace, aliasing, dan kata kunci `use`. Pertama, mengimpor kelas `Conflict` dari file `Conflict.php` dan kelas `Helper` dari file `Helper.php`. Selanjutnya, menggunakan kata kunci `use` untuk memberikan alias ke namespace dan elemen-elemen yang diimpor, misalnya, `Conflict` diubah menjadi `Conflict1` dan `Conflict2`. Fungsi `helpMe` dan konstanta `APPLICATION` juga diakses melalui alias `help` dan `APP`. Setelah itu, membuat objek dari `Conflict1` dan `Conflict2`, memanggil fungsi `help()`, dan mencetak nilai konstanta `APP`. Keseluruhan, ini adalah contoh penggunaan aliasing dan namespace untuk mengelola konflik nama dalam PHP dan membuat kode lebih bersih dan mudah dipahami.

- inhertiance.php

```php
<?php

// import data/person.php
require_once "data/Manager.php";
// buat object new manager dan tambahakan value nama kemudian panggil function
$manager1 = new Manager();
$manager1->nama = "Ahmadl";
$manager1->sayHello("Ahmadlboy");

// buat object new vicepresident dan tambahakan value nama kemudian panggil function
$vicePresident1 = new VicePresident();
$vicePresident1->nama = "Bunga";
$vicePresident1->sayHello("Bunga safitri");
```

Penjelasanya :

Kode ini mengimport kelas `Manager` dan `VicePresident` dari file `Manager.php` dan menciptakan objek `Manager` dan `VicePresident`. Setelah itu, nilai properti `nama` untuk masing-masing objek diatur, yaitu "Ahmadl" untuk `manager1` dan "Bunga" untuk `vicePresident1`. Kemudian, memanggil metode `sayHello` pada keduanya dengan memberikan parameter nama yang berbeda. Ini menunjukkan penggunaan konsep pemrograman berorientasi objek, di mana objek `Manager` dan `VicePresident` memiliki properti dan metode yang dapat diakses dan dimanipulasi. Keseluruhannya, ini adalah contoh sederhana dari pembuatan objek dan pemanggilan metode dalam konteks pemrograman berorientasi objek.

- namespace.php

```php
<?php

// buat namespace
// import data dari conflict
require "conflict.php";
// buat obeject dari namespace yang di buat
$conflict1 = new data\satu\conflict();
$conflict2 = new data\dua\conflict();
// import data helper
require "helper.php";
// tampilkan helper menggunakan echo
echo Helper\APPLICATION .PHP_EOL;
// masukan Helper\helpMe();
Helper\helpMe();
```

Penjelasanya :

Kode ini menciptakan dua namespace, `data\satu` dan `data\dua`, dengan mengimport kelas `conflict` dari file `conflict.php` ke dalam keduanya. Selanjutnya, objek `$conflict1` dan `$conflict2` dibuat dari masing-masing namespace. Kemudian, file `helper.php` diimpor dan konstanta `APPLICATION` dan fungsi `helpMe` dari namespace `Helper` digunakan dan ditampilkan menggunakan `echo`. Ini adalah contoh penggunaan namespace untuk mengorganisir kelas dan fungsi dalam PHP, memastikan pemisahan antara elemen-elemen yang mungkin memiliki nama yang sama, dan mempermudah pengelolaan dan pemeliharaan kode.

- object.php

object.php > ...

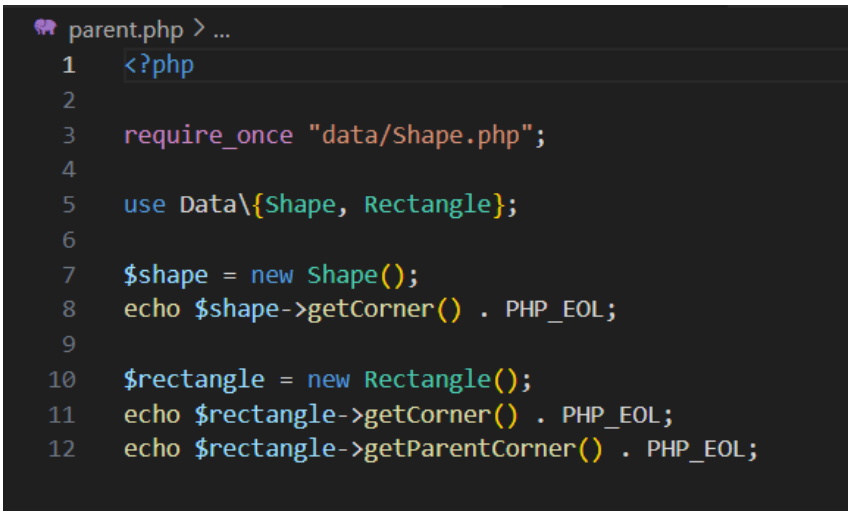
```php
<?php

// import data/person.php
require_once "data/person.php";

// buat object baru dari kelas person
$person = new Person("Ahmadl","Seluma");

// manipulasi properti nama, alamat, negara
$person->nama = "Ahmadl";
$person->alamat = "Seluma";
$person->negara = "Indonesia";

// menampilkan hasil
echo "nama = {$person->nama}" . PHP_EOL;
echo "alamat = {$person->alamat}" . PHP_EOL;
echo "negara = {$person->negara}" . PHP_EOL;

```

Penjelasanya :

Kode ini mengimport kelas `Person` dari file `person.php` dan membuat objek baru dari kelas tersebut dengan nama `$person`. Selanjutnya, properti objek, seperti `nama`, `alamat`, dan `negara`,

dimanipulasi dengan memberikan nilai baru. Akhirnya, nilai-nilai properti tersebut ditampilkan menggunakan pernyataan `echo`. Ini merupakan contoh penerapan konsep pemrograman berorientasi objek dalam PHP, di mana objek dibuat dari suatu kelas, propertinya dapat dimanipulasi, dan nilai-nilainya dapat diakses dan ditampilkan.

- parent.php



Penjelasanya :

Kode ini menggunakan kelas `Shape` dan `Rectangle` yang diimpor dari file `Shape.php`. Pertama, objek `$shape` dibuat dari kelas `Shape`, dan metode `getCorner()` dipanggil untuk menampilkan hasilnya. Selanjutnya, objek `$rectangle` dibuat dari kelas `Rectangle`, yang merupakan turunan dari `Shape`, dan metode `getCorner()` dipanggil kembali untuk menampilkan hasil spesifik dari kelas `Rectangle`. Terakhir, metode `getParentCorner()` dipanggil pada objek `$rectangle` untuk mengakses metode `getCorner()` dari kelas induknya (`Shape`). Keseluruhannya, ini adalah contoh implementasi pewarisan dan penggunaan namespace dalam PHP, di mana kelas turunan dapat mengakses metode dari kelas induknya.

- polymorphism.php

polymorphism.php > ...

```php
<?php

require_once "data/programmer.php";

$company = new Company();
$company->programmer = new Programmer("Ahmad");
var_dump($company);

$company->programmer = new BackendProgrammer("Ahmad");
var_dump($company);

$company->programmer = new FrontendProgrammer("Ahmad");
var_dump($company);

sayHelloProgrammer(new Programmer("Ahmad Dwi Cahyadi"));
sayHelloProgrammer(new BackendProgrammer("Ahmad Dwi Cahyadi"));
sayHelloProgrammer(new FrontendProgrammer("Ahmad Dwi Cahydi"));
```

Penjelasanya :

Kode ini menggunakan kelas-kelas `Programmer`, `BackendProgrammer`, dan `FrontendProgrammer` yang diimpor dari file `programmer.php`. Pertama, objek `$company` dibuat dari kelas `Company`, dan properti `programmer` diatur sebagai objek `Programmer` dengan nama "Ahmad". Setelah itu, properti `programmer` diubah menjadi objek `BackendProgrammer` dan `FrontendProgrammer` secara berturut-turut, dengan setiap kali menampilkan struktur objek menggunakan `var_dump`. Selanjutnya, fungsi `sayHelloProgrammer` dipanggil untuk menampilkan salam berdasarkan jenis programmer yang diberikan sebagai parameter. Keseluruhan, ini adalah contoh penerapan polimorfisme dalam pemrograman berorientasi objek di mana objek dari kelas-kelas turunan dapat digunakan secara fleksibel sesuai dengan konteksnya.

- properti.php

properti.php > ...

```php
<?php

// import data/person.php
require_once "data/person.php";

// buat object baru dari kelas person
$person1 = new Person("Ahmadl","Seluma");

// manipulasi properti nama person
$person1->nama = "Ahmad";
$person1->alamat = "Seluma";
$person1->negara = "Indonesia";
// menampilkan hasil
echo "nama = {$person1->nama}" . PHP_EOL;
echo "alamat = {$person1->alamat}" . PHP_EOL;
echo "negara = {$person1->negara}" . PHP_EOL;

```

Penjelasanya :

Kode ini mengimport kelas `Person` dari file `person.php` dan membuat objek baru dengan nama `$person1`. Properti dari objek ini, seperti `nama`, `alamat`, dan `negara`, dimanipulasi dengan memberikan nilai baru. Setelah itu, nilai-nilai properti tersebut ditampilkan menggunakan pernyataan `echo`. Ini adalah contoh penerapan konsep pemrograman berorientasi objek dalam PHP, di mana objek dibuat dari suatu kelas, dan propertinya dapat dimanipulasi dan ditampilkan sesuai kebutuhan program.

- selfkeyword.php

selfKeyword.php > ...

```php
<?php

// import data/person.php
require_once "data/person.php";

// buat object baru dari kelas person
$person1 = new Person("Ahmadl","Seluma");

// panggil function
$person1->sayHello("Ahmadl");

// panggil self keyword
$person1->info();

```

Penjelasanya :

Kode ini mengimport kelas `Person` dari file `person.php` dan membuat objek baru dengan nama `$person1`. Setelah itu, memanggil metode `sayHello` pada objek `$person1` dengan memberikan parameter "Ahmadl", yang mencetak pesan sapaan. Selanjutnya, memanggil metode `info` pada objek `$person1`, yang menggunakan kata kunci `self` untuk mengakses konstanta kelas (`AUTHOR`). Ini menunjukkan cara menggunakan metode dalam konteks objek dan mengakses elemen kelas menggunakan kata kunci `self`. Keseluruhannya, ini adalah contoh penerapan metode dan penggunaan `self` dalam pemrograman berorientasi objek dengan PHP.

- thiskeyword.php

thisKeyword.php > ...

```php
<?php

// import data/person.php
require_once "data/person.php";

// buat object dari kelas person
$ahmadl = new Person("Ahmadl", "Seluma");

// tambahkan value nama di object
$ahmadl->nama = "ahmadl";

// panggil function sayHelloNull dengan parameter
$ahmadl->sayHelloNull("Bunga Safitri");

// buat object dari kelas person
$bunga = new Person("Bunga", "Bandung");

// tambahkan value nama di object
$bunga->nama = "Cookie Pratama";

// panggil function sayHelloNull dengan parameter null
$bunga->sayHelloNull(null);

```

Penjelasanya :

Kode ini mengimport kelas `Person` dari file `person.php` dan menciptakan dua objek, `$ahmadl` dan `$bunga`, dengan menggunakan konstruktor untuk menginisialisasi properti nama dan alamat. Setelah itu, nilai properti `nama` diobjek `$ahmadl` diubah menjadi "ahmadl" dan metode `sayHelloNull` dipanggil dengan memberikan parameter "Bunga Safitri". Kemudian, objek `$bunga` dibuat dengan nilai properti `nama` diubah menjadi "Cookie Pratama", dan metode `sayHelloNull` dipanggil lagi, kali ini dengan memberikan parameter null. Keseluruhannya, ini adalah contoh penerapan konstruktor, pengubahan nilai properti, dan penggunaan metode dalam konteks pemrograman berorientasi objek di PHP.

- visability.php

visability.php > ...

```php
<?php

require_once "data/Product.php";

$product = new Product("Apple", 20000);

// tampilkan product get name
echo $product->getName(). PHP_EOL;
// tampilkan product get price
echo $product->getPrice(). PHP_EOL;

$dummy = new ProductDummy("Dummy", 1000);
$dummy->info();
```

Penjelasanya :

Kode ini mengimport kelas `Product` dan `ProductDummy` dari file `Product.php`. Pertama, objek `$product` dibuat dari kelas `Product` dengan memberikan nilai nama "Apple" dan harga 20000 melalui konstruktor. Selanjutnya, menggunakan metode `getName()` dan `getPrice()` untuk menampilkan nama dan harga produk tersebut. Kemudian, objek `$dummy` dibuat dari kelas `ProductDummy` dengan

memberikan nilai nama "Dummy" dan harga 1000 melalui konstruktor. Metode `info()` pada objek `$dummy` dipanggil, yang mencetak informasi nama dan harga produk dummy. Keseluruhannya, ini adalah contoh penggunaan kelas dan metode dalam pemrograman berorientasi objek di PHP, dengan objek yang dibuat dari kelas yang berbeda memiliki fungsionalitas yang sesuai dengan jenisnya.