

# **RESPONSI PROYEK PEMROGRAMAN BERBASIS OBJEK**



## **Disusun Oleh :**

Nama : Meisy Dianita

NPM : G1F022008

## **Asisten Dosen :**

- |                       |             |
|-----------------------|-------------|
| 1. Fadia Nur Shafitri | (G1F021010) |
| 2. Alvin Indrawan     | (G1F021020) |

## **Dosen Pengampu :**

- |                                    |
|------------------------------------|
| 1. Ferzha Putra Utama, S.T., M.Eng |
| 2. Arie Vatesia, S.T., M.TI, Ph.D  |

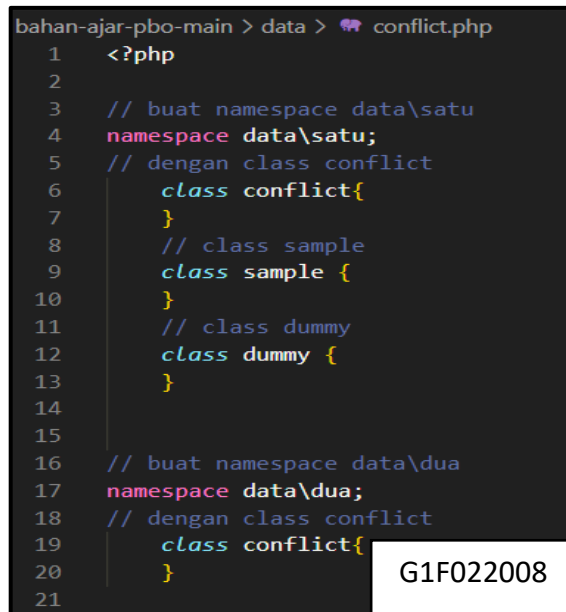
**PROGRAM STUDI SISTEM INFORMASI  
FAKULTAS TEKNIK  
UNIVERSITAS BENGKULU  
T.A 2023/2024**

## 1. Data

Data merupakan suatu *folder* yang terdapat pada *folder* bahan-ajar-pbo-main. Pada *folder* data terdapat beberapa *file* yang berisi *conflict*, *helper*, *manager*, *person*, *product*, *programmer*, dan *shape*. Pada *folder* data ini, inti dari pengkodean akan dibuat yang nantinya akan dipanggil pada *file-file* lain. Dengan adanya *folder* data maka kita tidak perlu melakukan perulangan pengkodean sesuai dengan prinsip DRY (*Don't Repeat Yourself*) yang terdapat pada pemrograman berorientasi objek. Hal ini akan mempermudah kita dalam melakukan pengkodean selanjutnya. Pembuatan *folder* data akan menjadi efektif dikarenakan dengan adanya *folder* ini maka kode menjadi lebih ringkas dan mudah untuk dimengerti.

Setiap *file* yang terdapat pada *folder* bahan-ajar-pbo-main disimpan dalam bentuk *.php*. Hal ini juga berlaku dengan *file* yang terdapat pada *folder* data. *Folder* data menjadi *folder* utama tempat pengembangan pengkodean yang dilakukan pada *folder* bahan-ajar-pbo-main. Untuk mengembangkan *file* lain maka kita harus memperhatikan *file* yang terdapat pada *folder* ini yaitu *folder* data. Berikut ini adalah penjelasan dari masing-masing *file* yang terdapat pada *folder* data di dalam *folder* bahan-ajar-pbo-main.

### 1) Conflict



```
bahan-ajar-pbo-main > data > conflict.php
1  <?php
2
3  // buat namespace data\satu
4  namespace data\satu;
5  // dengan class conflict
6  class conflict{
7  }
8  // class sample
9  class sample {
10 }
11 // class dummy
12 class dummy {
13 }
14
15
16 // buat namespace data\dua
17 namespace data\dua;
18 // dengan class conflict
19 class conflict{
20 }
21
```

Gambar 1 Perintah Kode Conflict

Source Code :

```
<?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{
    }

```

Penjelasan :

Gambar 1 menampilkan perintah kode *file conflict.php*. Pada awalnya, *file conflict.php* hanya berisi komentar yang memerintahkan mahasiswa untuk mengerjakan perintah yang sesuai dengan komentar yang diperintahkan. Mahasiswa diminta untuk membuat *namespace data\satu* dimana pada *namespace* tersebut terdiri dari *class conflict*, *class sample*, dan *class dummy*. Selanjutnya, mahasiswa juga diminta untuk membuat *namespace data\dua* dengan *class conflict*.

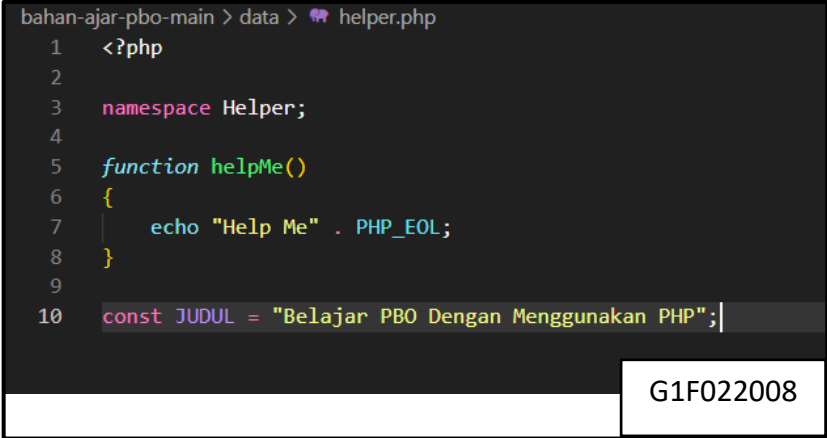
Untuk membuat *namespace* kita dapat langsung menuliskan kode `namespace data\satu;`. *Namespace* merupakan suatu fitur yang digunakan untuk mengelompokkan kelas, objek, ataupun fungsi yang berada di bawahnya. Pada *namespace data\satu*, kita telah mengelompokkan *class conflict*, *class sample*, dan *class dummy* berada pada *namespace data\satu*. Dengan demikian, ketika digunakan maka kelas, fungsi ataupun objek yang terdapat pada *namespace data\satu* akan lebih mudah untuk diklasifikasikan. Selanjutnya, kita diperintahkan untuk membuat *class conflict*, *class sample*, dan *class dummy* yang berada pada *namespace\data\satu*. Hal ini dapat dengan mudah dibuat dengan menggunakan kode `class nama_class ( ) {}`. Setelah class berhasil dibuat maka kita dapat melanjutkan pada perintah selanjutnya.

Perintah selanjutnya adalah pembuatan *namespace data\dua*. Kita dapat langsung melakukan pengkodean dengan menuliskan kode `namespace data\dua;` dimana di dalam *namespace* tersebut terdapat *class conflict*. Dengan demikian, seluruh perintah yang diperlukan pada *file conflict.php* telah berhasil dibuat.

Penggunaan *namespace* pada *file conflict.php* penting untuk dilakukan karena ketika kita ingin mengembangkan aplikasi dengan Pemrograman Berorientasi Objek yang menggunakan bahasa pemrograman PHP dalam cakupan yang luas, kita memerlukan *library* eksternal. Ketika hal ini dilakukan maka terdapat kemungkinan nama *class* dan fungsi yang kita lakukan sama. Hal ini dibuktikan dengan *namespace data\satu* dan *namespace data\dua* yang

memiliki class conflict dengan nama sama. Jika tidak diklasifikasikan dengan namespace maka akan terjadinya error karena terdapat kesamaan nama class dalam satu *file*.

## 2) Helper



```
bahan-ajar-pbo-main > data > helper.php
1  <?php
2
3  namespace Helper;
4
5  function helpMe()
6  {
7      echo "Help Me" . PHP_EOL;
8  }
9
10 const JUDUL = "Belajar PBO Dengan Menggunakan PHP";
```

Gambar 2 Perintah Kode Helper

Source Code :

```
<?php

namespace Helper;

function helpMe()
{
    echo "Help Me" . PHP_EOL;
}

const JUDUL = "Belajar PBO Dengan Menggunakan PHP";
```

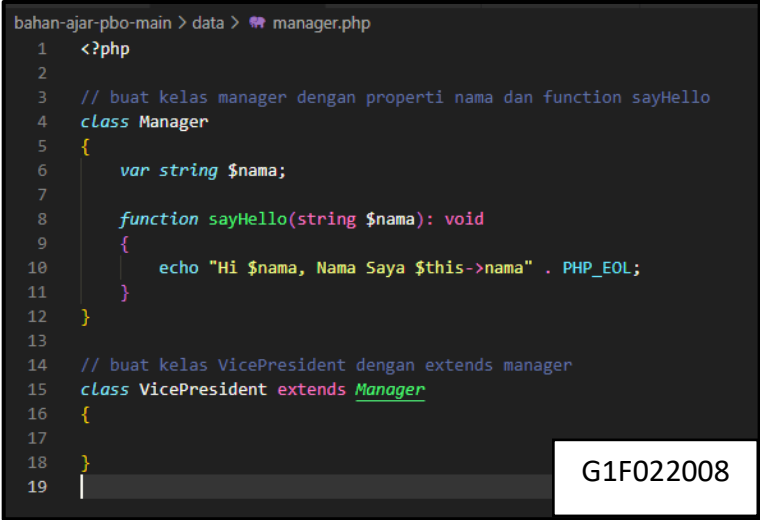
Penjelasan :

Gambar 2 menampilkan perintah kode yang terdapat pada *file* helper.php. Pada perintah di atas terdapat fitur *namespace* yang digunakan untuk mengklasifikasikan fungsi helpMe yang terdapat pada *file* helper.php. Pada *file* ini terdapat function helpMe(){} yang merupakan *user-defined function*. Hal ini dikarenakan *function* helpMe merupakan *function* yang dibuat oleh pengguna. *Function* helpMe digunakan untuk menampilkan tulisan “Help Me” yang dipanggil dengan menggunakan perintah `echo`. Perintah `echo "Help Me"` disertai dengan kode `. PHP_EOL` yang digunakan untuk *endline*.

Kita juga menggunakan fungsi `const` yang digunakan untuk mendeklarasikan *variable* dengan nilai yang tidak dapat diubah setelah nilai diberikan. Pada gambar di atas dapat dilihat bahwa kita menggunakan fungsi `const` dengan nama konstanta yaitu JUDUL. Adapun isi dari konstanta JUDUL adalah “Belajar PBO Dengan Menggunakan PHP”. *File* helper.php menjadi kelas induk dimana dalam pemanggilan konstanta judul pada *file-file* selanjutnya harus disesuaikan dengan penggunaan fungsi dan *namespace* yang terdapat pada *file* helper.php. Apabila pemanggilan yang dilakukan tidak sesuai dengan *file* helper.php dan kita hendak

menggunakannya maka hal ini dapat memunculkan suatu perintah kode *error* yang berarti terjadi kesalahan dalam melakukan pengkodean.

### 3) Manager



```
bahan-ajar-pbo-main > data > manager.php
1  <?php
2
3  // buat kelas manager dengan properti nama dan function sayHello
4  class Manager
5  {
6      var string $nama;
7
8      function sayHello(string $nama): void
9      {
10         echo "Hi $nama, Nama Saya $this->nama" . PHP_EOL;
11     }
12 }
13
14 // buat kelas VicePresident dengan extends manager
15 class VicePresident extends Manager
16 {
17
18 }
19
```

Gambar 3 Perintah Kode Manager

Source Code :

```
<?php

// buat kelas manager dengan properti nama dan function sayHello
class Manager
{
    var string $nama;
    function sayHello(string $nama): void
    {
        echo "Hi $nama, Nama Saya $this->nama" . PHP_EOL;
    }
}

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

Penjelasan ;

Gambar 3 menampilkan kode yang terdapat pada *file* manager.php yang terdapat pada *folder* data di dalam folder bahan-ajar-pbo-main. Pada *file* manager.php, kita diperintahkan untuk membuat kelas manager dengan properti nama dan function sayHello. Selanjutnya, kita diperintahkan ntuk membuat kelas VicePresident dengan perintah *extend* manager.

Untuk membuat class manager kita dapat menggunakan perintah `class Manager () { }`. Kemudian, kita dapat membuat properti nama dengan menggunakan perintah `var` yang disertai dengan tipe data dari properti nama. Untuk properti nama kita dapat menggunakan tipe data string yang dapat berisi karakter-karakter yang dibutuhkan dalam menuliskan nama.

Selanjutnya kita perlu membuat function sayHello. Function sayHello merupakan contoh dari user-defined function dimana fungsi ini dibuat sendiri oleh pengguna. Dalam function sayHello kita dapat meletakkan suatu parameter yang akan diterima fungsi ketika *file* ini dijalankan. Adapun parameter yang diletakkan adalah variable nama dengan tipe data string. Selanjutnya, kita dapat menampilkan tulisan dengan menggunakan perintah `echo` dimana dalam pemanggilan itu kita dapat langsung menggunakan variable nama yang disertai dengan return nama yang dipanggil dengan perintah `$this->nama`. Selanjutnya, kita dapat menggunakan kode `PHP_EOL` yang digunakan untuk *endline*.

Setelah membuat kelas manager kita diperintahkan untuk membuat kelas VicePresident dengan menggunakan prinsip *extends* pada kelas manager. Hal ini berarti bahwa kelas VicePresident dapat menggunakan perintah dan *function* yang terdapat pada *class manager* tanpa mendeklarasikan kembali. Dengan demikian, class VicePresident juga memiliki *variable* nama dengan *function* sayHello.

#### 4) Person

```

bahan-ajar-pbo-main > data > person.php
1  <?php
2
3  // membuat kelas person
4  class Person{
5      // membuat properti
6      var string $nama;
7
8      // gunakan nullable properti
9      var ?string $alamat = null;
10
11     // gunakan default value untuk properti
12     var string $negara = "Indonesia";
13
14     // buat function sayHello
15     function sayHello(string $nama){
16         echo "Hello $nama" . PHP_EOL;
17         //echo "<br>";
18     }
19
20     // buat function sayHello nullable dengan percabangan
21     function sayHelloNull(?string $nama)
22     {
23         if (is_null($nama)) {
24             echo "Hi, Nama Saya $this->nama" . PHP_EOL;
25         } else {
26             echo "Hi $nama, Nama Saya $this->nama" . PHP_EOL;
27         }
28     }
29
30     // buat const author
31     const PENGAJAR = "Meisy Dianita";
32
33     // buat function info untuk self keyword
34     function info()
35     {
36         echo "Nama Pengajar : " . self::PENGAJAR . PHP_EOL;
37     }
38
39     // buat function constructor
40     function __construct(string $nama, ?string $alamat)
41     {
42         $this->nama = $nama;
43         $this->alamat = $alamat;
44     }
45
46     // buat function destructor
47     function __destruct()
48     {
49         echo "<br>";
50         echo "Object person $this->nama is destroyed" . PHP_EOL;
51     }
52 }
53

```

G1F022008

Gambar 4 Kode Perintah Person

Source Code :

<?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";

// buat function sayHello

```

function sayHello(string $nama){
    echo "Hello $nama" . PHP_EOL;
}

// buat function sayHello nullable dengan percabangan
function sayHelloNull(?string $nama)
{
    if (is_null($nama)) {
        echo "Hi, Nama Saya $this->nama" . PHP_EOL;
    } else {
        echo "Hi $nama, Nama Saya $this->nama" . PHP_EOL;
    }
}

// buat const author
const PENGAJAR = "Meisy Dianita";

// buat function info untuk self keyword
function info()
{
    echo "Nama Pengajar : " . self::PENGAJAR . PHP_EOL;
}

// buat function constructor
function __construct(string $nama, ?string $alamat)
{
    $this->nama = $nama;
    $this->alamat = $alamat;
}

// buat function destructor
function __destruct()
{
    echo "<br>";
    echo "Object person $this->nama is destroyed" . PHP_EOL;
}
}

```

Penjelasan :

Gambar 4 menampilkan perintah kode yang terdapat pada *file* person.php. Pada *file* person.php kita diperintahkan untuk membuat kelas person dengan beberapa properti, yaitu properti nama, properti yang bersifat *nullable*, *function sayHello*, *function sayHello* yang bersifat *nullable*, *const author*, *function info* untuk self keyword, *function* untuk *constructor*, dan juga *function* untuk *destructor*.



Untuk membuat *class person* maka kita dapat menggunakan perintah `class Person {}` dimana kita dapat membuat berbagai properti pada kelas tersebut. Kita dapat membuat properti nama dengan menggunakan perintah `var`. Selanjutnya, untuk *variable* alamat kita biarkan bernilai *null*. Untuk *variable* negara kita dapat memasukkan *default value* dengan cara mendeklarasikan nilai yang terdapat pada *variable* negara. *Default value* negara berisi Indonesia. Hal ini berarti bahwa, ketika pengguna tidak memasukkan negara ketika *file* `person.php` dipanggil maka secara otomatis negara yang ditampilkan adalah Indonesia.

Selanjutnya, kita dapat membuat *function* `sayHello` dan `sayHelloNull` dimana kedua *function* ini merupakan *user-defined function* yaitu *function* yang dibuat dari pengguna. Pada *function* `sayHello` diperintahkan untuk menampilkan *hello* disertai dengan nilai dari *variable* *nama* yang diberikan pengguna. Adapun *function* `sayHelloNull` merupakan fungsi yang digunakan ketika pengguna memberikan nilai *null* pada *nama* maka terdapat percabangan yang akan terjadi. Masing-masing perintah `echo` dituliskan beserta dengan `.PHP_EOL` (*end of line*) yang berarti *endl* dari kalimat tersebut.

Hal selanjutnya adalah membuat konstanta dengan nama konstanta yaitu `PENGAJAR`. Untuk nilai dari konstanta `PENGAJAR` ini adalah “Meisy Dianita”. Konstanta `PENGAJAR` ini dibuat agar nilai dari `PENGAJAR` tidak berubah-ubah. Kemudian, terdapat *function* `info` dimana ketika *function* `info` dipanggil maka akan menampilkan nama pengajar.

*Function* `__construct` dan *function* `__destruct` merupakan *method* khusus yang dapat digunakan pada saat pembelajaran berorientasi objek. *Function* `__construct` sendiri memiliki parameter *nama* dan *alamat* yang akan digunakan pada *file* lain nantinya. Adapun *function* `__destruct` digunakan untuk menampilkan tulisan bahwa *Object person* sudah *didestroyed*. Setiap pemanggilan *file* `person.php` maka tulisan ini akan muncul.

## 5) Product

```
bahan-ajar-pbo-main > data > product.php
1  <?php
2
3  class Product
4  {
5      protected string $name;
6      protected int $price;
7
8      public function __construct(string $name, int $price)
9      {
10         $this->name = $name;
11         $this->price = $price;
12     }
13
14     public function getName(): string
15     {
16         echo "Nama Produk : "; return $this->name;
17     }
18
19     public function getPrice(): int
20     {
21         echo "Harga Produk : "; return $this->price;
22     }
23 }
24
25 class ProductDummy extends Product
26 {
27
28     public function info()
29     {
30         echo "Nama Produk : $this->name" . PHP_EOL;
31         echo "<br>";
32         echo "Harga Produk : $this->price" . PHP_EOL;
33     }
34
35 }
```

G1F022008

Gambar 5 Kode Perintah pada Product

Source Code :

```
<?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
    {
        echo "Nama Produk : "; return $this->name;
    }

    public function getPrice(): int
```

```

    {
        echo "Harga Produk : "; return $this->price;
    }
}

class ProductDummy extends Product
{

    public function info()
    {
        echo "Nama Produk : $this->name" . PHP_EOL;
        echo "<br>";
        echo "Harga Produk : $this->price" . PHP_EOL;
    }

}

```

Penjelasan :

Gambar 5 menampilkan perintah yang digunakan dalam membangun *file* product.php. Pada *file* product.php terdapat kelas *Product* yang di dalamnya terdapat beberapa fungsi seperti *function \_\_construct*, *function getName*, dan *function getPrice*. Pada *class Product* terdapat perintah `protected string $name;` dan `protected int $price;` yang berarti bahwa *variables* nama dan price hanya dapat digunakan pada *class Product* atau kelas turunannya. Pada *class Product*, saya menambahkan perintah `echo "Nama Produk"` dan `echo "Harga Produk"` yang dilakukan agar luaran yang dihasilkan nantinya menjadi lebih bagus dipandang oleh pengguna. Dengan demikian, kita dapat mengetahui mana yang nama produk dan mana yang harga produk.

Pada *class Product* juga terdapat kelas *ProductDumbby* yang melakukan *extends* terhadap kelas Produk. Hal ini berarti bahwa kelas *ProductDummy* dapat mengakses semua kelas, *function*, ataupun *method* yang dimiliki oleh kelas *Product*. Hal ini dapat membuat pengguna melakukan pengkodean menjadi lebih ringkas dan mudah. Selain itu, kode menjadi lebih efektif dikarenakan kelas, *method*, dan fungsi yang terdapat pada satu kelas dapat tetap diakses tanpa harus melakukan pendeklarasian ulang.

Pada *class ProductDummy* terdapat *function info* dimana ketika produk dituliskan dapat ditampilkan juga Nama Produk dan Harga Produk. Pada *function info* terdapat perintah `PHP_EOL` yang berarti *endline*. Perintah "`<br>`" digunakan untuk memberikan baris baru pada penulisan nama produk dan harga produk agar keduanya tidak berada pada baris yang sama. Hal ini sudah dilakukan pemodifikasian oleh penulis.

## 6) Programmer

```

bahan-ajar-pbo-main > data > programmer.php
1  <?php
2
3  class Programmer
4  {
5
6      public string $name;
7
8      public function __construct(string $name)
9      {
10         $this->name = $name;
11     }
12
13 }
14
15 class BackendProgrammer extends Programmer
16 {
17 }
18
19 class FrontendProgrammer extends Programmer
20 {
21 }
22
23 class Company
24 {
25     public Programmer $programmer;
26 }
27
28
29 function sayHelloProgrammer(Programmer $programmer)
30 {
31     if ($programmer instanceof BackendProgrammer) {
32         echo "Hello Backend Programmer $programmer->name" . PHP_EOL;
33     } else if ($programmer instanceof FrontendProgrammer) {
34         echo "Hello Frontend Programmer $programmer->name" . PHP_EOL;
35     } else if ($programmer instanceof Programmer) {
36         echo "Hello Programmer $programmer->name" . PHP_EOL;
37     }
38 }

```

G1F022008

Gambar 6 Kode Perintah pada File programmer.php

Source Code :

```

<?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;
        }
    }
}

```

Penjelasan :

Gambar 6 menampilkan kode yang terdapat pada *file* programmer.php. Pada *file* programmer.php terdapat *class Programmer*, *class BackendProgrammer*, *class FrondendProgrammer*, dan *class Company*. Pada *class programmer.php* juga berisi *function sayHelloProgrammer* dimana menampilkan beberapa kondisi yang terdapat pada kondisi tertentu.

*Class Programmer* berisi *function \_\_construct* dengan parameter nama dengan tipe data string. Kemudian, untuk kelas *BackendProgrammer* dan kelas *FrondendProgrammer* dikenakan aksi *extends* terhadap *class Programmer*. Hal ini berarti kedua kelas tersebut dapat mengambil berbagai *class*, *function*, dan *method* yang terdapat pada *class Programmer*.

Pada *function info* terdapat keyword *instance of* yang digunakan untuk mengecek apakah suatu objek merupakan turunan dari suatu kelas. Keyword ini akan mengembalikan nilai benar jika objek adalah *instance* dari kelas tertentu dan akan mengembalikan nilai salah jika objek bukan merupakan *instance* dari kelas tertentu. Pada masing-masing perintah *echo* terdapat perintah *.PHP\_EOL* yang berarti *endline* yang akan memberikan baris baru untuk kata selanjutnya yang disatukan dengan kalimat tersebut jika hal ini diperlukan. Dengan demikian, *class Programmer* dapat digunakan pada *file* yang memerlukannya.

7) Shape

```

bahan-ajar-pbo-main > data > shape.php
1  <?php
2
3  namespace Data;
4
5  class Shape
6  {
7
8      public function getCorner()
9      {
10         return -1;
11     }
12
13 }
14
15 class Rectangle extends Shape
16 {
17
18     public function getCorner()
19     {
20         return 4;
21     }
22
23     public function getParentCorner()
24     {
25         return parent::getCorner();
26     }
27
28 }

```

G1F022008

Gambar 7 Kode Perintah pada File data/shape.php

Source Code :

```
<?php
```

```
namespace Data;
```

```
class Shape
```

```
{
```

```
    public function getCorner()
```

```
    {
```

```
        return -1;
```

```
    }
```

```
}
```

```
class Rectangle extends Shape
```

```
{
```

```
    public function getCorner()
```

```
    {
```

```
        return 4;
```

```
    }
```

```
    public function getParentCorner()
```

```
    {
```

```
        return parent::getCorner();
```

```
    }
```

```
}
```

Penjelasan :

Gambar 7 menampilkan perintah yang terdapat pada *file* data/shape.php. Pada kode di atas terdapat *namespace* yang bernama Data dimana *class Shape* dan *class Rectangle* terdapat pada *namespace* tersebut. Pada *class Shape* terdapat sebuah *function* dengan nama *getCorner* yang memiliki *return* -1. Pada *file* data/shape.php juga terdapat *class Rectangle* yang memiliki *extends* dengan *class Shape*. Pada *class Rectangle* juga terdapat *function getCorner* yang memiliki *return* 4. Selain itu, pada *class Rectangle* juga terdapat *function getParentCorner* yang memiliki *return parent*.

## 2. Constant



```
bahan-ajar-pbo-main > constant.php
1  <?php
2
3  // import data/person.php
4  require_once "data/Person.php";
5
6  // buat define
7  define("JUDUL", "Belajar PBO");
8
9  // buat const app version
10 const APP_VERSION = "1.0.0";
11
12 // tampilkan hasil
13 echo JUDUL . PHP_EOL;
14 echo APP_VERSION . PHP_EOL;
15 echo Person::PENGAJAR . PHP_EOL;
16
```

Gambar 8 Kode Perintah pada File constant.php

Source Code :

```
<?php
// import data/person.php
require_once "data/Person.php";

// buat define
define("JUDUL", "Belajar Pemrograman Berorientasi Objek");
// buat const app version
const APP_VERSION = "1.0.0";
// tampilkan hasil
echo JUDUL . PHP_EOL;
echo APP_VERSION . PHP_EOL;
echo Person::PENGAJAR . PHP_EOL;
```

Penjelasan :

Gambar 8 menampilkan kode perintah pada *file* constant.php. Pada *file* constant.php, kita diminta untuk mengimport data/person.php, membuat *define*, membuat *const* dengan nama APP\_Version, dan menampilkan hasil dari kode yang kita lakukan. Untuk melakukan suatu *import* data kita dapat menggunakan perintah `require_once`. Perintah `require_once` digunakan untuk memasukkan *file* php data/person.php pada *file* constant.php. Dengan adanya perintah ini maka *file* constant.php dapat mengakses *file* data/person.php dengan mudah.

Selanjutnya terdapat perintah `define` yang digunakan untuk mendefinisikan sesuatu. Pada perintah `define` yang terdapat pada *file* constant.php berisi “JUDUL” yang disertai dengan isi dari judul yaitu “Belajar Pemrograman Berorientasi Objek”. Untuk nama “JUDUL” akan terus digunakan ketika kita membutuhkannya. Dengan demikian, kita harus melakukan definisi ini dengan nama yang sesuai yaitu JUDUL ketika hendak menampilkan isi dari JUDUL.

Kita juga diperintahkan untuk membuat konstanta dengan nama APP\_VERSION. Untuk pembuatan konstanta kita dapat menggunakan perintah `const` yang digunakan untuk membuat konstanta APP\_VERSION. Selanjutnya, kita perlu mengisi isi dari konstanta tersebut. Dalam hal ini diisi dengan 1.0.0.

Untuk menampilkan hasil kita dapat menggunakan perintah `echo` yang digunakan untuk menampilkan JUDUL, APP\_VERSION, dan PENGAJAR yang terdapat pada class Person pada *file* person.php. Untuk setiap pemanggilan perintah `echo` kita dapat menggunakan perintah `. PHP_EOL` yang berarti *endline*.



Gambar 9 Luaran File constant.php

Penjelasan :

Gambar 9 menampilkan luaran *file* constant.php. Luaran yang dihasilkan dari *file* tersebut adalah “Belajar Pemrograman Berorientasi Objek 1.0.0 Meisy Dianita”. Hal ini berarti kode yang kita susun sebelumnya sudah benar hingga menghasilkan luaran yang sesuai dengan yang diinginkan. Sesuai dengan perintah untuk menampilkan hasil tadi, hal pertama yang akan ditampilkan adalah judul yang berisi “Belajar Pemrograman Berorientasi Objek” kemudian APP\_VERSION yang berisi “1.0.0” serta nama PENGAJAR yang telah dibuat pada *file* data/person yaitu “Meisy Dianita”.



### 3. Constractor



```
bahan-ajar-pbo-main > constractor.php
1  <?php
2
3  // import data/person.php
4  require_once "data/Person.php";
5
6  // buat object new person dengan 2 parameter
7  $meisy = new Person ("Meisy Dianita", "Bengkulu");
8
9  // vardump object
10 var_dump($meisy);
11
```

Gambar 10 Kode Perintah constractor.php

Source Code :

```
<?php
```

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

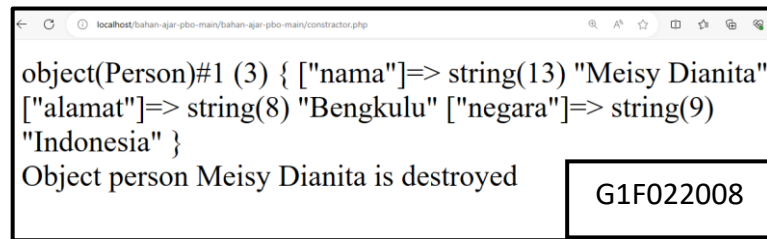
// buat object new person dengan 2 parameter
$meisy = new Person ("Meisy Dianita", "Bengkulu");

// vardump object
var_dump($meisy);
```

Penjelasan :

Gambar 10 menampilkan perintah `constractor.php`. Pada *file* `constractor.php`, kita diminta untuk *mengimport* `data/person.php`, membuat objek baru dengan 2 parameter, dan menggunakan *function* `var_dump`. Untuk melakukan suatu *import* data kita dapat menggunakan perintah `require_once`. Perintah `require_once` digunakan untuk memasukkan *file* `php` `data/person.php` pada *file* `constractor.php`. Dengan adanya perintah ini maka *file* `constractor.php` dapat mengakses *file* `data/person.php` dengan mudah.

Untuk membuat objek baru kita dapat menyesuaikan dengan perintah `new Person` pada *file* `data/person.php`. Dikarenakan yang diminta terdiri dari dua parameter maka kita dapat mengisi parameter nama dan alamat. Perintah terakhir adalah menggunakan fungsi `var_dump` pada objek yang baru saja kita buat. Perintah `var_dump` digunakan untuk proses *debugging* dimana ketika kita perlu mengetahui struktur informasi dari objek yang kita gunakan. Perintah `var_dump` akan menampilkan informasi berupa nilai dan tipe data yang terdapat pada suatu *variable*. Dalam hal ini kita melakukan perintah `var_dump` pada objek `meisy`. Dengan demikian, luaran nantinya akan menghasilkan struktur informasi apa saja yang terdapat pada objek yang `meisy`. Dengan demikian, apabila terjadinya *error* pada suatu *file* yang begitu kompleks kita dapat membedah kode yang telah kita susun sedemikian rupa hingga mendapatkan solusi yang lebih mudah untuk mengatasi *error*.



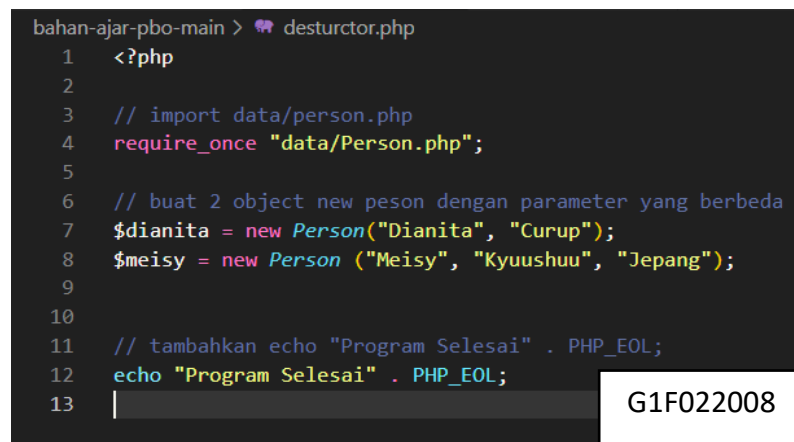
```
object(Person)#1 (3) { ["nama"]=> string(13) "Meisy Dianita"
["alamat"]=> string(8) "Bengkulu" ["negara"]=> string(9)
"Indonesia" }
Object person Meisy Dianita is destroyed
```

Gambar 11 Luaran File constructor.php

Penjelasan :

Gambar 11 menampilkan luaran *file* constructor.php yang kodenya telah kita rancang sebelumnya. Pada gambar di atas terdapat struktur informasi yang terdapat pada objek yang baru saja kita buat. Pada *variable* nama memiliki tipe data String dengan panjang 13 yang bernilai “Meisy Dianita”. Dalam hal ini spasi juga dihitung sebagai karakter. Alamat pada gambar di atas juga memiliki tipe data String dengan panjang 8 yang bernilai “Bengkulu”. Untuk Negara, walaupun kita tidak mendefinisikannya sebelumnya, tetap menampilkan Negara Indonesia. Hal ini dikarenakan pada *file* data/person.php kita telah mendefinisikan default value untuk negara. Dengan demikian, walaupun kita tidak mengisi negara Indonesia akan tetap menampilkan default value pada objek yang baru dibuat.

#### 4. Desturctor



```
bahan-ajar-pbo-main > 🐞 destructor.php
1  <?php
2
3  // import data/person.php
4  require_once "data/Person.php";
5
6  // buat 2 object new peson dengan parameter yang berbeda
7  $dianita = new Person("Dianita", "Curup");
8  $meisy = new Person ("Meisy", "Kyuushuu", "Jepang");
9
10
11 // tambahkan echo "Program Selesai" . PHP_EOL;
12 echo "Program Selesai" . PHP_EOL;
13
```

Gambar 12 Kode Perintah pada File destructor.php

Source Code :

```
<?php
// import data/person.php
require_once "data/Person.php";

// buat 2 object new peson dengan parameter yang berbeda
$dianita = new Person("Dianita", "Curup");
$meisy = new Person ("Meisy", "Kyuunshuu", "Jepang");

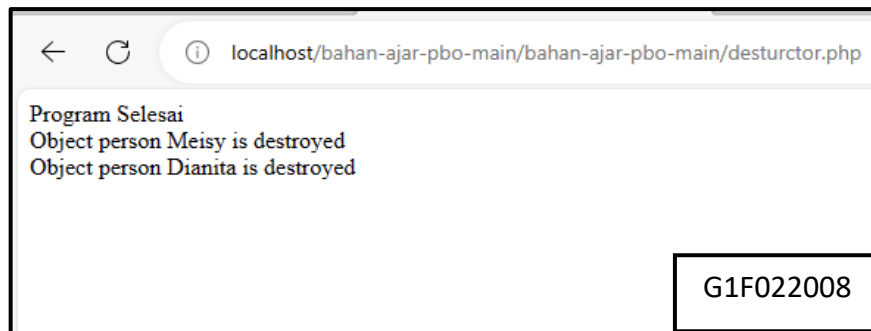
// tambahkan echo "Program Selesai" . PHP_EOL;
```

```
echo "Program Selesai" . PHP_EOL;
```

Penjelasan :

Gambar 12 menampilkan kode perintah destructor.php. Pada *file* destrurctor.php, kita diminta untuk meng-*import* data/person.php, membuat 2 objek baru dengan parameter yang berbeda, dan menggunakan *function* var\_dump. Untuk melakukan suatu *import* data kita dapat menggunakan perintah `require_once`. Perintah `require_once` digunakan untuk memasukkan *file* php data/person.php pada *file* destructor.php. Dengan adanya perintah ini maka *file* destructor.php dapat mengakses *file* data/person.php dengan mudah.

Untuk membuat objek baru kita dapat menyesuaikan dengan perintah `new Person` pada *file* data/person.php. Dikarenakan yang diminta terdiri dari dua parameter yang berbeda maka kita dapat mengisi parameter nama dan alamat untuk objek dianita dan parameter nama, alamat, negara untuk objek meisy. Selanjutnya, kita diperintahkan untuk menampilkan program selesai dengan diakhiri dengan *endline*.



Gambar 13 Luaran pada File destructor.php

Penjelasan :

Gambar 13 menampilkan luaran pada *file* destructor.php. Luaran yang dihasilkan pada gambar di atas menyatakan bahwa program selesai dan objek Meisy dan Dianita yang telah kita buat sebagai objek baru sebelumnya sudah di-*destroyed*. Dengan demikian, luaran yang dihasilkan telah sesuai dengan yang dibutuhkan oleh pengguna.

## 5. Function

```
bahan-ajar-pbo-main > function.php
1  <?php
2
3  // import data/person.php
4  require_once "data/person.php";
5
6  // buat object baru dari kelas person
7  $meisy = new Person ("Meisy", "Bengkulu");
8
9  // panggil function
10 $meisy->sayHello("Meisy");
11
```

G1F022008

Gambar 14 Kode Perintah pada File function.php

Source Code :

```
<?php

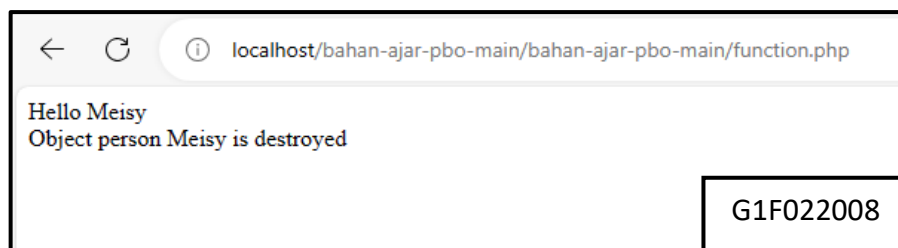
// import data/person.php
require_once "data/person.php";

// buat object baru dari kelas person
$meisy = new Person ("Meisy", "Bengkulu");

// panggil function
$meisy->sayHello("Meisy");
```

Penjelasan :

Gambar 14 menampilkan kode perintah pada *file* function.php. Pada *file* function.php, kita diminta untuk mengimport data/person.php yang disertai dengan perintah untuk membuat objek baru dari kelas Person yang terdapat pada data/person.php. Untuk melakukan suatu *import* data kita dapat menggunakan perintah `require_once`. Perintah `require_once` digunakan untuk memasukkan *file* php data/person.php pada *file* function.php. Dengan adanya perintah ini maka *file* function.php dapat mengakses *file* data/person.php dengan mudah. Selanjutnya, kita diperintahkan untuk memanggil *function* sayHello yang telah dibuat pada file data/person.php sebelumnya. Dengan demikian, perintah yang terdapat pada file function.php telah berhasil dibuat.



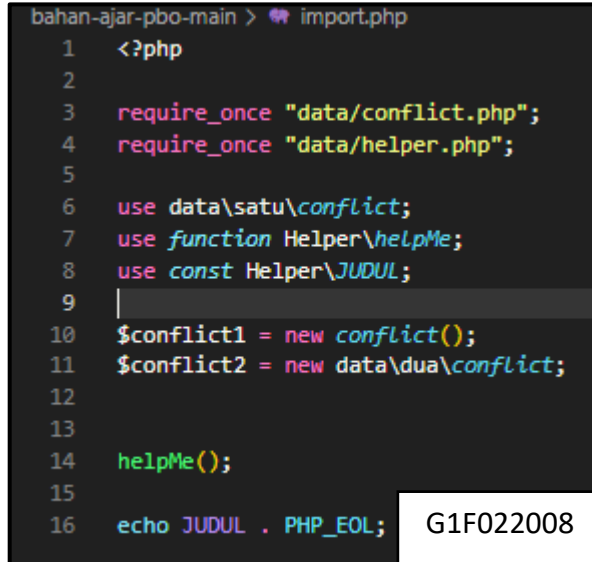
Gambar 15 Luaran pada File function.php

Penjelasan :

Gambar 15 menampilkan luaran pada *file* function.php. Pada luaran tersebut terdapat luaran Hello Meisy dimana luaran ini dihasilkan dari *function* sayHello yang telah kita panggil sebelumnya pada *file* function.php yang berasal dari *import* data/person.php. Hal ini juga berlaku pada luaran yang mengatakan “Object person Meisy is destroyed”. Perintah untuk menampilkan hal ini juga didapatkan dari hasil *import* yang dilakukan pada *file* data/person.php

sebelumnya. Dengan demikian, luaran yang dihasilkan telah memenuhi tiga aspek yang harus dipenuhi pada *file* function.php, yaitu meng-*import* data/person.php, membuat objek baru dari kelas Person yang terdapat pada data/person.php, serta memanggil function sayHello yang telah dideklarasikan sebelumnya.

## 6. Import



```
bahan-ajar-pbo-main > import.php
1  <?php
2
3  require_once "data/conflict.php";
4  require_once "data/helper.php";
5
6  use data\satu\conflict;
7  use function Helper\helpMe;
8  use const Helper\JUDUL;
9
10 $conflict1 = new conflict();
11 $conflict2 = new data\dua\conflict;
12
13
14 helpMe();
15
16 echo JUDUL . PHP_EOL;
```

Gambar 16 Kode Perintah pada File import.php

Source Code :

```
<?php
require_once "data/conflict.php";
require_once "data/helper.php";
use data\satu\conflict;
use function Helper\helpMe;
use const Helper\JUDUL;
$conflict1 = new conflict();
$conflict2 = new data\dua\conflict;
helpMe();
echo JUDUL . PHP_EOL;
```

Penjelasan :

Gambar 16 menampilkan kode perintah pada *file* import.php. Pada *file* ini terdapat perintah *import* data/conflict.php dan data/conflict/helper.php. Untuk melakukan suatu *import* data kita dapat menggunakan perintah `require_once`. Perintah `require_once` digunakan untuk memasukkan file data/conflict.php dan data/conflict/helper.php pada *file* import.php. Dengan adanya perintah ini maka file import.php dapat mengakses *file* data/conflict.php dan data/conflict/helper.php dengan mudah.

Pada *file* import.php, kita menggunakan data\satu\conflict yang digunakan untuk mengakses data\satu\conflict pada *file* conflict.php. Kita juga menggunakan *function* Helper yang berada pada class helpMe serta konstanta Helper yang berisi JUDUL.

Untuk membuat *conflict* yang baru maka kita dapat menggunakan perintah *new*. Selanjutnya, kita dapat menggunakan *method* helpMe dan menampilkan JUDUL agar judul dapat ditampilkan pada luaran yang kita inginkan. Dengan demikian, kode yang terdapat pada *file* import.php telah berhasil dibangun.



Gambar 17 Luaran pada File import.php

Penjelasan :

Gambar 17 menampilkan luaran pada *file* import.php. Luaran yang dihasilkan pada gambar di atas adalah Help Me yang berasal dari *method* helpMe yang telah kita import pada data/helper.php serta JUDUL yang berisi “Belajar PBO Dengan Menggunakan PHP” yang telah di-import pada data/helper.php. Dengan demikian, luaran yang dihasilkan sudah sesuai dengan yang dibutuhkan oleh pengguna.

## 7. ImportAlias

```
bahan-ajar-pbo-main > importAlias.php
1  <?php
2
3  require_once "data/Conflict.php";
4  require_once "data/Helper.php";
5
6  use data\satu\conflict as conflict1;
7  use data\dua\conflict as conflict2;
8  use function Helper\helpMe as help;
9  use const Helper\JUDUL as APP;
10 $conflict1 = new Conflict1();
11 $conflict2 = new Conflict2();
12
13 help();
14
15 echo APP . PHP_EOL;
```

Gambar 18 Kode Perintah pada File importAlias.php

Source Code :

```
<?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\JUDUL as APP;

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

help();
echo APP . PHP_EOL;

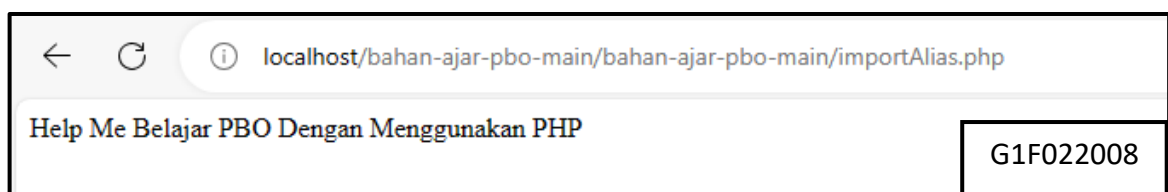
```

Penjelasan :

Gambar 18 menampilkan perintah pada *file* importAlias.php. Pada *file* ini terdapat perintah *import* data/conflict.php dan data/conflict/helper.php. Untuk melakukan suatu *import* data kita dapat menggunakan perintah `require_once`. Perintah `require_once` digunakan untuk memasukkan *file* data/conflict.php dan data/conflict/helper.php pada *file* import.php. Dengan adanya perintah ini maka *file* import.php dapat mengakses *file* data/conflict.php dan data/conflict/helper.php dengan mudah.

Pada *file* import.php, kita menggunakan data\satu\conflict yang digunakan untuk mengakses data\satu\conflict pada *file* conflict.php. `as` conflict1 dan data\dua\conflict `as` conflict2. Kita juga menggunakan alias untuk *function* Helper\helpMe dengan mengaliaskannya menjadi help serta const Helper\JUDUL dialiaskan menjadi APP.

Untuk membuat *conflict* yang baru maka kita dapat menggunakan perintah `new`. Selanjutnya, kita dapat menggunakan *method* help yang sudah dialiaskan dari method helpMe dan menampilkan APP yang merupakan alias dari *const* JUDUL agar judul dapat ditampilkan pada luaran yang kita inginkan. Dengan demikian, kode yang terdapat pada *file* import.php telah berhasil dibangun.



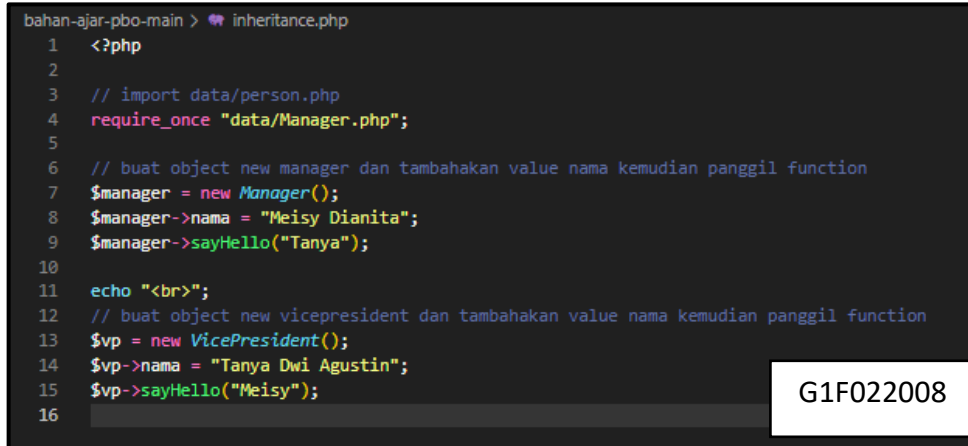
Gambar 19 Luaran pada File importAlias.php

Penjelasan :

Gambar 19 menampilkan luaran pada *file* import.php. Luaran yang dihasilkan pada gambar di atas adalah Help Me yang berasal dari *method* help yang merupakan alias dari *method* helpMe yang terdapat pada data/Helper.php yang telah kita import pada data/Helper.php sebelumnya. Selanjutnya APP yang merupakan alias dari *const* JUDUL yang berisi “Belajar

PBO Dengan Menggunakan PHP” yang telah diimport pada data/helper.php. Dengan demikian, luaran yang dihasilkan sudah sesuai dengan yang dibutuhkan oleh pengguna. Oleh karena itu, penggunaan perintah `as` pada kode yang dirancang sebelumnya telah berhasil dilakukan oleh penulis.

## 8. Inheritance



```
bahan-ajar-pbo-main > inheritance.php
1  <?php
2
3  // import data/person.php
4  require_once "data/Manager.php";
5
6  // buat object new manager dan tambahkan value nama kemudian panggil function
7  $manager = new Manager();
8  $manager->nama = "Meisy Dianita";
9  $manager->sayHello("Tanya");
10
11 echo "<br>";
12 // buat object new vicepresident dan tambahkan value nama kemudian panggil function
13 $vp = new VicePresident();
14 $vp->nama = "Tanya Dwi Agustin";
15 $vp->sayHello("Meisy");
16
```

Gambar 20 Kode Perintah pada File inheritance.php

Source Code :

```
<?php

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

// buat object new manager dan tambahkan value nama kemudian panggil function
$manager = new Manager();
$manager->nama = "Meisy Dianita";
$manager->sayHello("Tanya");

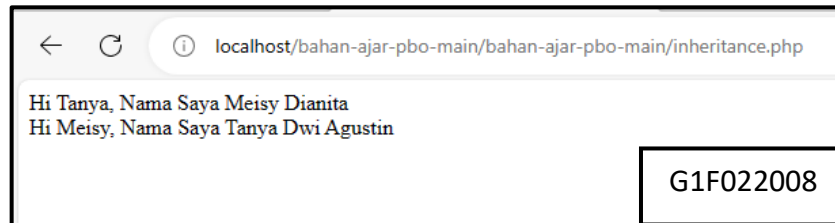
echo "<br>";
// buat object new vicepresident dan tambahkan value nama kemudian panggil
function
$vp = new VicePresident();
$vp->nama = "Tanya Dwi Agustin";
$vp->sayHello("Meisy");
```

Penjelasan :

Gambar 20 menampilkan kode perintah pada *file* inheritance.php. Pada *file* ini terdapat perintah `import` data/Manager.php. Untuk melakukan suatu `import` data kita dapat menggunakan perintah `require_once`. Perintah `require_once` digunakan untuk memasukkan *file* data/Manager.php pada *file* inheritance.php. Dengan adanya perintah ini maka *file* inheritance.php dapat mengakses *file* data/Manager.php.



Selanjutnya, kita diperintahkan untuk membuat objek baru pada *Class manager* dengan menambahkan isi dari nama yang disertai perintah untuk memanggil *function* sayHello. Hal ini juga berlaku pada *class VicePresident*. Kita diperintahkan untuk membuat objek baru pada *Class manager* dengan menambahkan isi dari nama yang disertai perintah untuk memanggil *function* sayHello.



Gambar 21 Luaran pada File inheritance.php

Penjelasan :

Gambar 21 menampilkan luaran pada *file* inheritance.php. Pada luaran tersebut dihasilkan terdapat *function* sayHello yang sudah di-import pada *class Manager* dan *class VicePresident*. Dengan demikian, luaran yang dihasilkan telah sesuai dengan yang dibutuhkan oleh kode.

## 9. Name Space

```
bahan-ajar-pbo-main > nameSpace.php
1  <?php
2
3  // buat namespace
4  // import data dari conflict
5  require_once "data/conflict.php";
6
7  // buat oboject dari namespace yang di buat
8  $conflict1 = new data\satu\conflict;
9  $conflict2 = new data\dua\conflict;
10
11 // import data helper
12 require_once "data/helper.php";
13
14 // tampilkan helper menggunakan echo
15 echo Helper\JUDUL . PHP_EOL;
16
17 // masukan Helper\helpMe();
18 echo "<br>";
19 Helper\helpMe();
```

Gambar 22 Kode Perintah pada File nameSpace.php

Source Code :

```
<?php

// buat namespace
// import data dari conflict
Require_once "data/conflict.php";
// buat oboject dari namespace yang di buat
$conflict1 = new data\satu\conflict;
$conflict2 = new data\dua\conflict;
```

```
// import data helper
Require_once "data/helper.php";
// tampilkan helper menggunakan echo
echo Helper\JUDUL . PHP_EOL;
// masukan Helper\helpMe();
echo "<br>";
Helper\helpMe();
```

Penjelasan :

Gambar 22 menampilkan kode perintah pada *file* nameSpace.php. Pada *file* ini terdapat perintah *import* data/conflict.php dan data/helper.php. Untuk melakukan suatu *import* data kita dapat menggunakan perintah `require_once`. Perintah `require_once` digunakan untuk memasukkan *file* data/conflict.php dan data/helper.php pada *file* nameSpace.php. Dengan adanya perintah ini maka *file* nameSpace.php dapat mengakses file data/conflict.php dan data/helper.php

Untuk *import* yang dilakukan pada data/conflict.php dilakukan pembuatan objek dari *namespace* yang dibuat yaitu data\satu\conflict dan data\dua\conflict. Adapun untuk *import* data/helper.php maka diperintahkan untuk menampilkan JUDUL yang disertai dengan *method* helpMe().



Gambar 23 Luaran pada File nameSpace.php

Penjelasan :

Gambar 23 menampilkan luaran pada *file* nameSpace.php. Pada luaran tersebut dihasilkan terdapat JUDUL serta *method* helpMe yang di-*import* dari *file* data/helper.php. Dengan demikian, luaran yang dihasilkan telah sesuai dengan syarat yang diperlukan oleh kode.

## 10. Object

```

bahan-ajar-pbo-main > object.php
1  <?php
2
3  // import data/person.php
4  require_once "data/person.php";
5
6  // buat object baru dari kelas person
7  $meisy = new Person("Meisy","Bengkulu");
8
9  // manipulasi properti nama, alamat, negara
10
11  $meisy->nama = "Meisy";
12  $meisy->alamat = "Bengkulu";
13  $meisy->negara = "Indonesia";
14
15  // menampilkan hasil
16  echo "Nama : {$meisy->nama}" . PHP_EOL;
17  echo "<br>";
18  echo "Alamat : {$meisy->alamat}" . PHP_EOL;
19  echo "<br>";
20  echo "Negara : {$meisy->negara}" . PHP_EOL;

```

G1F022008

Gambar 24 Kode Perintah pada File object.php

<?php

// import data/person.php  
require\_once "data/person.php";

// buat object baru dari kelas person  
\$meisy = new Person("Meisy","Bengkulu");

// manipulasi properti nama, alamat, negara

\$meisy->nama = "Meisy";  
\$meisy->alamat = "Bengkulu";  
\$meisy->negara = "Indonesia";

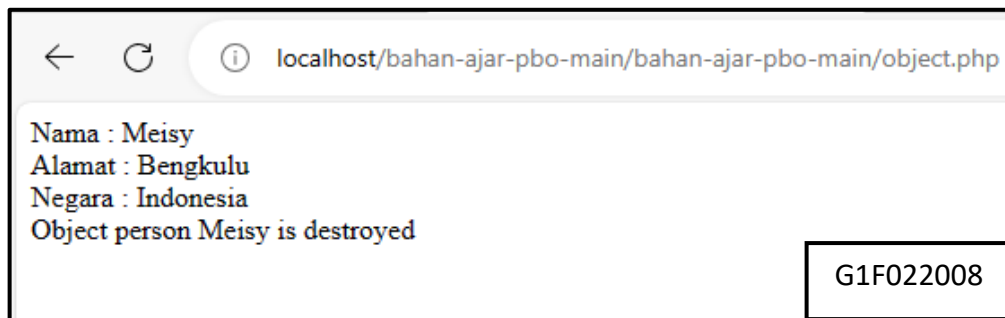
// menampilkan hasil  
echo "Nama : {\$meisy->nama}" . PHP\_EOL;  
echo "<br>";  
echo "Alamat : {\$meisy->alamat}" . PHP\_EOL;  
echo "<br>";  
echo "Negara : {\$meisy->negara}" . PHP\_EOL;

Penjelasan :

Gambar 24 menampilkan perintah yang terdapat pada *file* object.php. Pada *file* object.php, kita diminta untuk meng-*import* data/person.php, membuat objek baru dari kelas Person, memanipulasi properti nama, alamat, dan negeri, serta menampilkan hasil. Untuk melakukan suatu import data kita dapat menggunakan perintah `require_once`. Perintah

`require_once` digunakan untuk memasukkan *file* php `data/person.php` pada *file* `object.php`. Dengan adanya perintah ini maka *file* `object.php` dapat mengakses *file* `data/person.php` dengan mudah.

Untuk membuat objek baru kita dapat menggunakan perintah `new`. Dalam melakukan manipulasi properti dapat dideklarasikan dengan memanggil nama objek baru yang disertai dengan nama, alamat, dan negara. Untuk menampilkan hasil kita dapat menggunakan perintah `echo`.



Gambar 25 Luaran pada File `object.php`

Penjelasan :

Gambar 25 menampilkan luaran pada *file* `object.php`. Pada luaran tersebut ditampilkan manipulasi properti, seperti nama, alamat, dan negara yang telah dilakukan sebelumnya. Hasil yang ditampilkan juga sesuai dengan perintah yang terdapat pada *file* `object.php`. Dengan demikian, kita dapat melanjutkan pada pengkodean selanjutnya karena *file* `object.php` telah selesai dan benar.

#### 11. Parent



Gambar 26 Kode Perintah pada File `parent.php`

Source Code :

```
<?php
```

```
require_once "data/Shape.php";
```

```

use Data\{Shape, Rectangle};

$shape = new Shape();
echo $shape->getCorner() . PHP_EOL;

$rectangle = new Rectangle();
echo $rectangle->getCorner() . PHP_EOL;
echo $rectangle->getParentCorner() . PHP_EOL;

```

Penjelasan :

Gambar 26 menampilkan perintah yang terdapat pada *file* parent.php. Pada *file* parent.php, kita diminta untuk meng-*import* data/shape.php. Untuk melakukan suatu *import* data kita dapat menggunakan perintah `require_once`. Perintah `require_once` digunakan untuk memasukkan *file* php data/parent.php pada *file* parent.php. Dengan adanya perintah ini maka *file* parent.php dapat mengakses *file* data/shape.php dengan mudah.

Selanjutnya kita dapat membuat objek baru pada *Class Shape* dengan menggunakan perintah `new`. Hal ini juga berlaku pada objek baru pada *class Rectangle*. Kita juga dapat menggunakan perintah `new`. Setelah membuat kedua objek baru maka kita dapat menampilkan objek yang telah kita buat dengan menggunakan perintah `echo`.



Gambar 27 Luaran pada File parent.php

Penjelasan :

Gambar 27 menampilkan luaran pada *file* parent.php. Luaran yang dihasilkan dari *file* parent.php tersebut adalah -1 4 -1. Hal ini didapatkan dari perintah yang telah kita *import* dari *file* data/Shape.php sebelumnya. Dengan demikian, hasil yang dikeluarkan dari *file* parent.php adalah -1 4 -1 dan sesuai dengan *import* pada data/Shape.php.

## 12. Polymorphism

```

bahan-ajar-pbo-main > polymorphism.php
1  <?php
2
3  require_once "data/Programmer.php";
4
5  $company = new Company();
6  $company->programmer = new Programmer("Meisy");
7  var_dump($company);
8
9  $company->programmer = new BackendProgrammer("Dianita");
10 var_dump($company);
11
12 $company->programmer = new FrontendProgrammer("Meme");
13 var_dump($company);
14 echo "<br>";
15 sayHelloProgrammer(new Programmer("Meisy"));
16 echo "<br>";
17 sayHelloProgrammer(new BackendProgrammer("Dianita"));
18 echo "<br>";
19 sayHelloProgrammer(new FrontendProgrammer("Meme"));
20

```

G1F022008

Gambar 28 Kode Perintah pada File polymorphism.php

Source Code :

```

<?php

require_once "data/Programmer.php";

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

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

$company->programmer = new FrontendProgrammer("Meme");
var_dump($company);
echo "<br>";
sayHelloProgrammer(new Programmer("Meisy"));
echo "<br>";
sayHelloProgrammer(new BackendProgrammer("Dianita"));
echo "<br>";
sayHelloProgrammer(new FrontendProgrammer("Meme"));

```

Penjelasan :

Gambar 28 menampilkan perintah yang terdapat pada *file* polymorphism.php. Pada *file* polymorphism.php, kita diminta untuk meng-*import* data/Programmer.php. Untuk melakukan suatu *import* data kita dapat menggunakan perintah `require_once`. Perintah `require_once` digunakan untuk memasukkan *file* php data/Programmer.php pada *file*

polymorphism.php. Dengan adanya perintah ini maka *file* polymorphism.php dapat mengakses *file* data/Programmer dengan mudah.

Selanjutnya kita dapat membuat objek baru pada *Class Programmer* dengan menggunakan perintah **new**. Hal ini juga berlaku pada objek baru pada class *BackendProgrammer* dan *FrontendProgrammer* kita juga dapat menggunakan perintah **new**. Pada masing-masing objek baru terdapat perintah **var\_dump**. Perintah **var\_dump** digunakan untuk proses *debugging* dimana ketika kita perlu mengetahui struktur informasi dari objek yang kita gunakan. Perintah **var\_dump** akan menampilkan informasi berupa nilai dan tipe data yang terdapat pada suatu *variable*. Dengan demikian, luaran nantinya akan menghasilkan struktur informasi apa saja yang terdapat pada objek yang meisy. Dengan demikian, apabila terjadinya error pada suatu *file* yang begitu kompleks kita dapat membedah kode yang telah kita susun sedemikian rupa hingga mendapatkan solusi yang lebih mudah untuk mengatasi error.

```
object(Company)#1 (1) { ["programmer"]=> object(Programmer)#2 (1) { ["name"]=> string(5)
"Meisy" } } object(Company)#1 (1) { ["programmer"]=> object(BackendProgrammer)#3 (1) {
["name"]=> string(7) "Dianita" } } object(Company)#1 (1) { ["programmer"]=>
object(FrontendProgrammer)#2 (1) { ["name"]=> string(4) "Meme" } }
Hello Programmer Meisy
Hello Backend Programmer Dianita
Hello Frontend Programmer Meme
```

G1F022008

Gambar 29 Luaran File polymorphism.php

Penjelasan :

Gambar 29 menampilkan luaran *file* polymorphism.php. Pada gambar di atas terdapat struktur informasi yang terdapat pada objek yang baru saja kita buat. Pada gambar di atas terdapat *object Programmer*, *object BackendProgrammer*, dan *object FrontendProgrammer* yang disertai dengan struktur informasinya. Untuk menampilkan perintah sayHelloProgrammer kita dapat menggunakan *function* yang telah didefinisikan sebelumnya pada *Class Programmer*. Dengan demikian, kode tersebut telah menampilkan “hello programmer”.

### 13. Properti

```

bahan-ajar-pbo-main > properti.php
1  <?php
2
3  // import data/person.php
4  require_once "data/person.php";
5
6  // buat object baru dari kelas person
7  $meisy = new Person("Meisy","Bengkulu");
8
9  // manipulasi properti nama person
10 $meisy->nama = "Meisy";
11
12 // menampilkan hasil
13 echo "Nama : {$meisy->nama}" . PHP_EOL;
14 echo "<br>";
15 echo "Alamat : {$meisy->alamat}" . PHP_EOL;
16 echo "<br>";
17 echo "Negara : {$meisy->negara}" . PHP_EOL;

```

G1F022008

Gambar 30 Kode Perintah pada File properti.php

Source Code :

```

<?php

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

// buat object baru dari kelas person
$meisy = new Person("Meisy","Bengkulu");

// manipulasi properti nama person
$meisy->nama = "Meisy";

// menampilkan hasil
echo "Nama : {$meisy->nama}" . PHP_EOL;
echo "<br>";
echo "Alamat : {$meisy->alamat}" . PHP_EOL;
echo "<br>";
echo "Negara : {$meisy->negara}" . PHP_EOL;

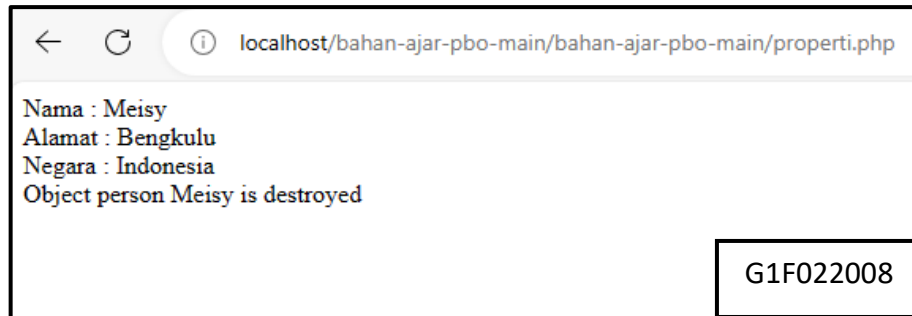
```

Penjelasan :

Gambar 30 menampilkan perintah yang terdapat pada *file* properti.php. Pada *file* properti.php, kita diminta untuk meng-*import* data/person.php, membuat objek baru dari kelas Person, memanipulasi properti nama, serta menampilkan hasil. Untuk melakukan suatu *import* data kita dapat menggunakan perintah `require_once`. Perintah `require_once` digunakan untuk memasukkan *file* php data/person.php pada *file* properti.php. Dengan adanya perintah ini maka *file* properti.php dapat mengakses file data/person.php dengan mudah.



Untuk membuat objek baru kita dapat menggunakan perintah `new`. Dalam melakukan manipulasi properti dapat dideklarasikan dengan memanggil nama objek baru yang disertai dengan nama, alamat, dan negara. Untuk menampilkan hasil kita dapat menggunakan perintah `echo`.



Gambar 31 Luaran pada File properti.php

Penjelasan :

Gambar 31 menampilkan luaran pada *file* properti.php. Pada luaran tersebut ditampilkan manipulasi properti nama. Hasil yang ditampilkan juga sesuai dengan perintah yang terdapat pada *file* properti.php. Dengan demikian, kita dapat melanjutkan pada pengkodean selanjutnya karena *file* properti.php telah selesai dan benar.

#### 14. SelfKeyword

```
bahan-ajar-pbo-main > selfKeyword.php
1  <?php
2
3  // import data/person.php
4  require_once "data/person.php";
5
6  // buat object baru dari kelas person
7  $meisy = new Person("Meisy","Bengkulu");
8
9  // panggil function
10 $meisy->sayHello("Meisy");
11
12 // panggil self keyword
13 echo "<br>";
14 $meisy->info();
```

Gambar 32 Kode Perintah pada File selfKeyword.php

Source Code :

```
<?php
```

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

```
// buat object baru dari kelas person
```

```
$meisy = new Person("Meisy","Bengkulu");
```

```
// panggil function
```

```
$meisy->sayHello("Meisy");
```

```
// panggil self keyword
```

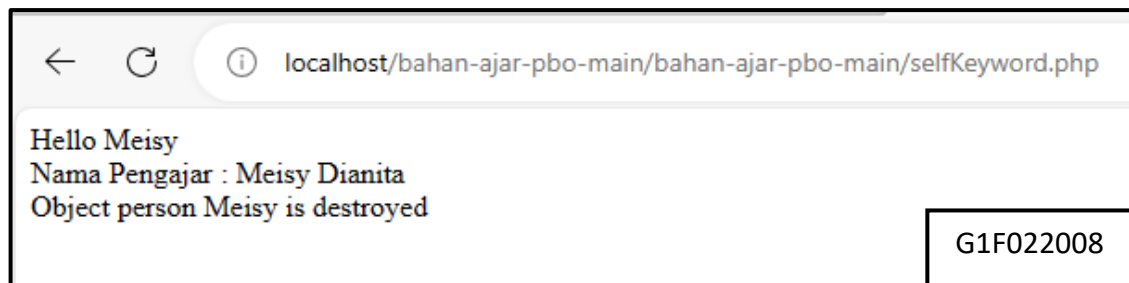
```
echo "<br>";
```

```
$meisy->info();
```

Penjelasan :

Gambar 32 menampilkan perintah yang terdapat pada *file* selfKeyword.php. Pada *file* selfKeyword.php, kita diminta untuk mengimport data/person.php, membuat objek baru dari kelas Person, memanggil function, serta memanggil self keyword. Untuk melakukan suatu import data kita dapat menggunakan perintah `require_once`. Perintah `require_once` digunakan untuk memasukkan *file* php data/person.php pada *file* selfKeyword.php. Dengan adanya perintah ini maka *file* selfKeyword.php dapat mengakses *file* data/person.php dengan mudah.

Untuk membuat objek baru kita dapat menggunakan perintah `new`. Kita dapat memanggil *function* yang sesuai yaitu sayHello serta memberikan informasi mengenai objek baru yang telah kita buat. Kita juga dapat menggunakan perintah `echo "<br>"` yang digunakan untuk membuat *new line* pada output selanjutnya.



Gambar 33 Luaran pada File selfKeyword.php

Penjelasan :

Gambar 33 menampilkan luaran pada *file* selfKeyword.php. Pada *file* selfKeyword.php terdapat luaran yang mengeluarkan sapaan berupa “Hello Meisy”. Selanjutnya, terdapat “Nama Pengajar : Meisy Dianita” yang disertai dengan *function* `__destruct` yang telah dibuat pada *file* data/person.php. Oleh karena itu, setiap kita memanggil atau meng-*import file* data/person.php maka akan secara otomatis terdapat “Object person \$nama is destroyed”. Dengan demikian, luaran yang dihasilkan telah sesuai dengan perintah dan kode yang dibutuhkan pada *file* selfKeyword.php.

15. ThisKeyword

```

bahan-ajar-pbo-main > thisKeyword.php
1  <?php
2
3  // import data/person.php
4  require_once "data/person.php";
5
6  // buat object dari kelas person
7  $meisy = new Person("Meisy","Bengkulu");
8
9  // tambahkan value nama di object
10 $meisy->nama = "Meisy";
11
12 // panggil function sayHelloNull dengan parameter
13 $meisy->sayHello("Dianita");
14 echo "<br>";
15
16 // buat object dari kelas person
17 $dianita = new Person ("Dianita", "Curup");
18
19 // tambahkan value nama di object
20 $dianita->sayHello("Meisy");
21
22 // panggil function sayHelloNull dengan parameter null
23 echo "<br>";
24 $dianita->sayHelloNull("Meisy");

```

G1F022008

Gambar 34 Kode Perintah pada File thisKeyword.php

Source Code :

```

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

// buat object dari kelas person
$meisy = new Person("Meisy","Bengkulu");

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

// panggil function sayHelloNull dengan parameter
$meisy->sayHello("Dianita");
echo "<br>";

// buat object dari kelas person
$dianita = new Person ("Dianita", "Curup");

// tambahkan value nama di object
$dianita->sayHello("Meisy");

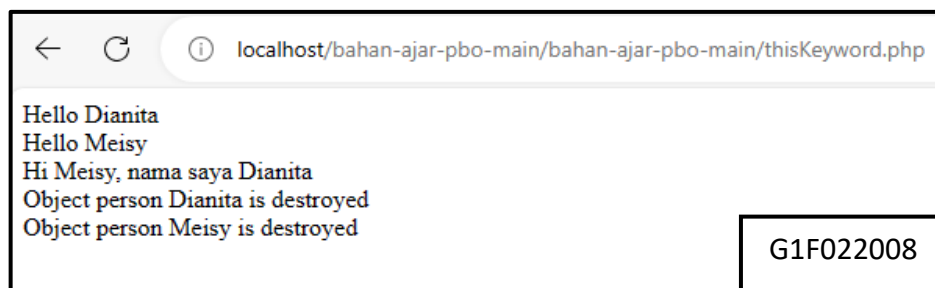
// panggil function sayHelloNull dengan parameter null
echo "<br>";
$dianita->sayHelloNull("Meisy");

```

Penjelasan :

Gambar 34 menampilkan perintah yang terdapat pada *file* thisKeyword.php. Pada *file* thisKeyword.php, kita diminta untuk meng-*import* data/person.php, membuat objek baru dari kelas Person, menambahkan nilai nama di objek, memanggil *function* sayHelloNull dengan parameter sebanyak dua buah. Untuk melakukan suatu *import* data kita dapat menggunakan perintah `require_once`. Perintah `require_once` digunakan untuk memasukkan *file* php data/person.php pada *file* thisKeyword.php. Dengan adanya perintah ini maka *file* thisKeyword.php dapat mengakses *file* data/person.php dengan mudah.

Untuk membuat objek baru kita dapat menggunakan perintah `new`. Kita dapat memanggil *function* yang sesuai yaitu sayHello dan sayHelloNull serta memberikan informasi mengenai objek baru yang telah kita buat yang sesuai dengan perintah komentar yang terdapat pada *file* thisKeyword.php. Kita juga dapat menggunakan perintah `echo "<br>"` yang digunakan untuk membuat *new line* pada output selanjutnya.



Gambar 35 Luaran pada File thisKeyword.php

Penjelasan :

Gambar 35 menampilkan luaran pada *file* thisKeyword.php. Pada luaran tersebut terdapat sapaan pada kedua objek yang sudah dibuat pada *file* thisKeyword.php yang disertai dengan *destroyed* yang juga dilakukan pada kedua objek.

## 16. Visibility

```
bahan-ajar-pbo-main > visibility.php
1  <?php
2
3  require_once "data/Product.php";
4
5  $product = new Product("Mouse", 85000);
6
7  // tampilkan product get name
8  echo $product->getName() . PHP_EOL;
9  echo "<br>";
10
11 // tampilkan product get price
12 echo $product->getPrice() . PHP_EOL;
13 echo "<br>";
14 $dummy = new ProductDummy("Charger Laptop Lenovo", 350000);
15
16 $dummy->info();
```

Gambar 36 Kode Perintah pada File visibility.php

Source Code :

```

<?php

require_once "data/Product.php";

$product = new Product("Mouse", 85000);

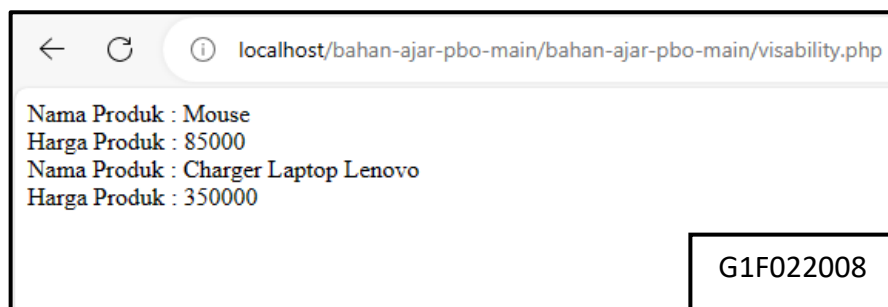
// tampilkan product get name
echo $product->getName() . PHP_EOL;
echo "<br>";

// tampilkan product get price
echo $product->getPrice() . PHP_EOL;
echo "<br>";
$dummy = new ProductDummy("Charger Laptop Lenovo", 350000);
$dummy->info();

```

Penjelasan :

Gambar 34 menampilkan perintah yang terdapat pada *file* visibility.php. Pada *file* visibility.php, kita diminta untuk meng-*import* data/person.php, menampilkan *product get name* dan menampilkan *product get price*. Untuk melakukan suatu *import* data kita dapat menggunakan perintah `require_once`. Perintah `require_once` digunakan untuk memasukkan *file* php data/person.php pada *file* visibility.php. Dengan adanya perintah ini maka *file* visibility.php dapat mengakses *file* data/person.php dengan mudah. Kita juga dapat menggunakan perintah `echo "<br>"` yang digunakan untuk membuat *new line* pada output selanjutnya.



Gambar 37 Luaran File visibility.php

Penjelasan :

Gambar 37 menampilkan luaran *file* visibility.php. Pada luaran tersebut terdapat nama produk, harga produk dari kedua objek yang telah dibuat sebelumnya. Dapat dilihat pada gambar 37 bahwa Mouse memiliki harga 85000 dan Charger Laptop Lenovo memiliki harga 350000. Kedua objek ditampilkan dengan menggunakan perintah *getName* dan *getPrice* yang sesuai dengan perintah yang terdapat pada komentar yang ada pada *source code*. Dengan demikian, luaran yang dihasilkan telah sesuai dengan yang dibutuhkan.