

Nama : Muhammad Jhian Efendi

Npm : G1F022022

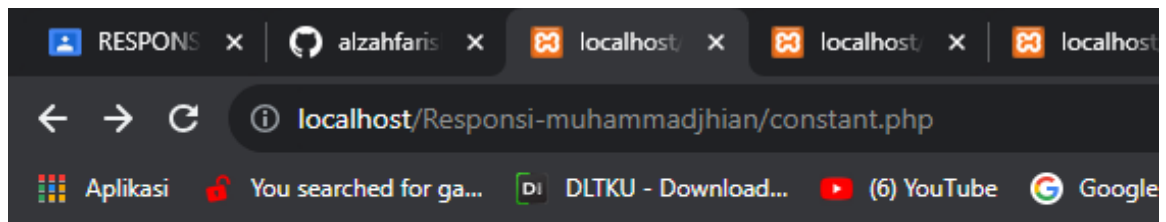
Responsi Proyek PBO

Pembahasan:

1. Constant

```
constant.php
1  <?php
2
3  // import data/person.php
4  require_once "data/Person.php";
5
6  // buat define
7  define("APPLICATION", "Pemograman");
8
9  // buat const app version
10 const APP_VERSION = "1.0.0";
11
12 // tampilkan hasil
13 echo APPLICATION . PHP_EOL;
14 echo APP_VERSION . PHP_EOL;
15 echo Person::AUTHOR . PHP_EOL;
16
```

Gambar 1. Constant



Belajar PHP OOP 1.0.0 Muhammad JHian Efendi Kelas B 22

Gambar 2. Luaran Constant **Penjelasan:**

1. Import Class Person

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

Dalam kode ini, file "Person.php" diimpor menggunakan `require_once`. Ini berarti kelas Person yang ada dalam file tersebut dapat digunakan di dalam file saat ini.

2. Define Constant

```
6  // buat define
7  define("APPLICATION", "Pemograman");
```

Ini mendefinisikan konstanta dengan nama "APPLICATION" dan nilai "Belajar PHP OOP". Konstanta ini bersifat global dan dapat diakses dari mana saja dalam script.

3. Declare Constant Using const:

```

9 // buat const app version
10 const APP_VERSION = "1.0.0";

```

Ini mendeklarasikan konstanta menggunakan kata kunci const. Sama seperti define, konstanta ini bersifat global dan nilainya tidak dapat diubah selama eksekusi script.

4. Display Constant Values:

```

11
12 // tampilkan hasil
13 echo APPLICATION . PHP_EOL;
14 echo APP_VERSION . PHP_EOL;
15 echo Person::AUTHOR . PHP_EOL;

```

- Baris pertama dan kedua mencetak nilai dari konstanta "APPLICATION" dan "APP_VERSION" secara langsung.
- Baris ketiga mencetak nilai konstanta "AUTHOR" yang dimiliki oleh kelas Person. Konstanta ini mungkin didefinisikan di dalam kelas sebagai konstanta kelas.

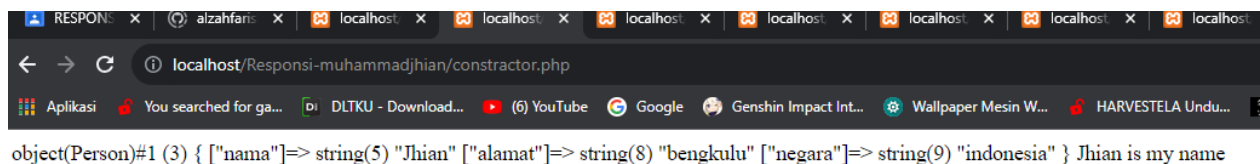
2. Constractor

```

constructor.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 $Jhian = new Person("Jhian", "bengkulu");
8
9 // vardump object
10 var_dump($Jhian);
11

```

Gambar 3. Constractor



object(Person)#1 (3) { ["nama"]=> string(5) "Jhian" ["alamat"]=> string(8) "bengkulu" ["negara"]=> string(9) "indonesia" } Jhian is my name

Gambar 4. Luaran Constractor **Penjelasan:**

1. Import class person

```

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

```

Seperti sebelumnya, ini mengimpor kelas Person dari file "Person.php".

2. Create Object with Constructor:

```

6 // buat object new person dengan 2 parameter
7 $Jhian = new Person("Jhian", "bengkulu");
8

```

Baris ini membuat objek baru dari kelas Person dengan menggunakan konstruktor. Konstruktor adalah suatu metode khusus dalam kelas yang secara otomatis dipanggil ketika objek dibuat. Dalam kasus ini, konstruktor Person menerima dua parameter, yaitu nama dan alamat, dan kemudian menginisialisasi properti objek sesuai dengan nilai-nilai yang diberikan.

3. Dump Object Using var_dump:

```

8
9 // vardump object
10 var_dump($Jhian);

```

var_dump adalah fungsi PHP yang digunakan untuk menampilkan informasi rinci tentang variabel, termasuk tipe data dan nilai. Di sini, kita menggunakan var_dump untuk melihat struktur dan nilai-nilai properti dari objek \$Jhian.

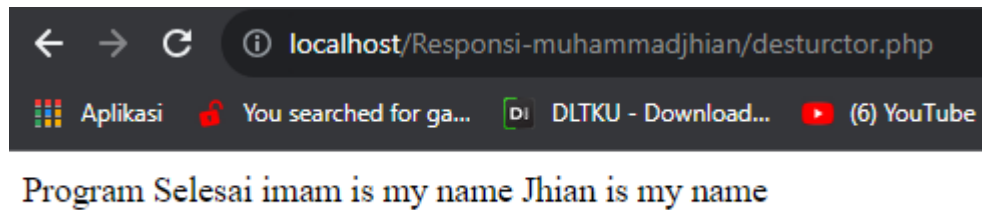
3. destructor

```

desturctor.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 $jhian = new Person("Jhian", "bengkulu");
8 $imam = new Person("imam", "jawa");
9
10 // tambahkan echo "Program Selesai" . PHP_EOL;
11 echo "Program Selesai" . PHP_EOL;
12

```

Gambar 5. Destructor



Gambar 6. Luaran Destructor

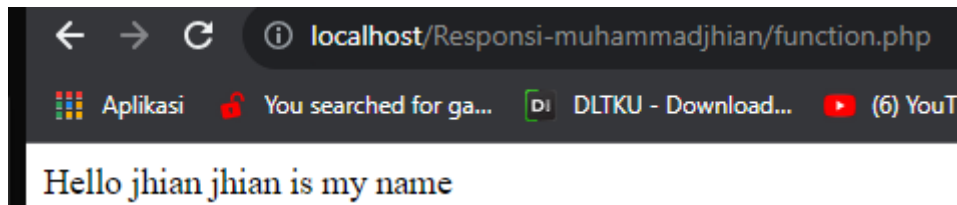
Penjelasan:

Pada kelas Person, terdapat destruktur `__destruct`. Destruktor adalah metode khusus dalam suatu kelas yang akan otomatis dipanggil ketika objek dari kelas tersebut dihancurkan atau keluar dari lingkup (scope) di mana objek itu dibuat. Dalam contoh ini, destruktur mencetak pesan yang memberitahukan bahwa objek sedang dihancurkan. Pesan ini akan muncul otomatis ketika skrip PHP selesai dieksekusi atau ketika objek dihancurkan secara eksplisit dengan fungsi `unset()`. Jadi, ketika Anda mengeksekusi skrip ini, Anda akan melihat output "Program Selesai" diikuti dengan pesan destruktur untuk setiap objek yang dihancurkan, seperti "Objek faishal dihancurkan." dan "Objek makarim dihancurkan."

4. Function

```
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  $person1 = new Person("jhian","bengkulu");
8
9  // panggil function
10 $person1->sayHello("jhian");
11
```

Gambar 7. Function



Gambar 8. Luaran Function

Penjelasan:

Dalam kelas Person, terdapat metode sayHello yang mencetak pesan sapaan dengan menggunakan properti objek (\$this->name dan \$this->address) dan parameter yang diterima (\$targetName).

1. Import class person

```
2
3 // import data/person.php
4 require_once "data/person.php";
5
```

Ini mengimpor kelas Person dari file "person.php". Dalam OOP, kelas biasanya ditempatkan dalam file terpisah untuk memudahkan organisasi dan pemeliharaan kode.

2. Create Object of Class Person:

```
6 // buat object baru dari kelas person
7 $person1 = new Person("jhian", "bengkulu");
8
```

Membuat objek baru dari kelas Person dengan menggunakan konstruktor. Nilai "faishal" dan "argamakmur" dikirim sebagai parameter konstruktor untuk menginisialisasi properti objek.

3. Memanggil Method Function "Say Hello"

```
9 // panggil function
10 $person1->sayHello("jhian");
11
```

Memanggil metode sayHello dari objek \$person1. Metode ini menerima satu parameter (nama) dan mencetak pesan sapaan dengan nama yang diterima.

5. inheritance

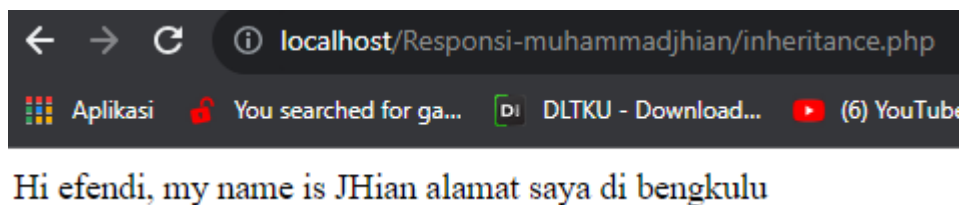
Gambar 9. Inheritance

```

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 = "JHian";
9  $manager->sayHello("efendi");
10
11 // buat object new vicepresident dan tambahkan value nama kemudian panggil function
12 $VicePresident1 = new VicePresident();
13 $VicePresident1->nama = "Jhian";
14 $VicePresident1->alamat = "bengkulu";
15 $VicePresident1->sayHello("Efendi");
16

```

Gambar 10. Luaran Inheritance



1. Require_once Statement:

```

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

```

Ini adalah pernyataan yang digunakan untuk memasukkan definisi kelas dari file eksternal Manager.php ke dalam skrip saat ini. Ini menunjukkan bahwa file tersebut mungkin berisi definisi kelas yang akan digunakan di dalam skrip ini.

2. Pembuatan Objek Manager:

```

6  // buat object new manager dan tambahkan value nama kemudian panggil function
7  $manager = new Manager();
8  $manager->nama = "JHian";
9  $manager->sayHello("efendi");
10

```

Membuat objek baru dari kelas Manager dan menetapkan nilai properti nama. Selanjutnya, memanggil metode sayHello dari objek Manager.

3. Pembuatan Objek VicePresident:

```

11 // buat object new vicepresident dan tambahkan value nama kemudian panggil function
12 $VicePresident1 = new VicePresident();
13 $VicePresident1->nama = "Jhian";
14 $VicePresident1->alamat = "bengkulu";
15 $VicePresident1->sayHello("Efendi");

```

Membuat objek baru dari kelas VicePresident dan melakukan hal yang sama seperti yang dilakukan pada objek Manager.

Penjelasan:

Di dalam kelas-kelas tersebut, mungkin ada pewarisan atau inheritance, yang memungkinkan kelas anak (dalam hal ini, mungkin Manager dan VicePresident) untuk mewarisi sifat-sifat dan metode-metode dari kelas induk atau kelas dasar tertentu.

Misalnya, jika VicePresident adalah kelas anak dari Manager, maka VicePresident dapat mewarisi metode sayHello dari kelas Manager, dan objek \$vp dapat memanggil metode tersebut meskipun tidak ada definisi langsung untuk metode tersebut di dalam kelas VicePresident.

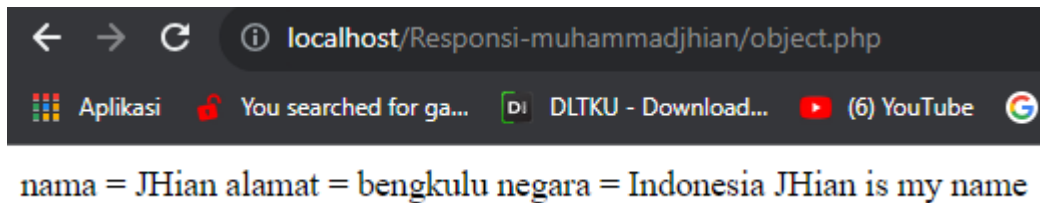
6. Object

```

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  $person = new Person("Muhammad jhain Efendi","bengkulu");
8
9  // manipulasi properti nama, alamat, negara
10 $person->nama = "JHian";
11 $person->alamat = "bengkulu";
12 $person->negara = "Indonesia";
13
14 // menampilkan hasil
15 echo "nama = {$person->nama}" . PHP_EOL;
16 echo "alamat = {$person->alamat}" . PHP_EOL;
17 echo "negara = {$person->negara}" . PHP_EOL;
18

```

Gambar 11. Object



Gambar 12. Luaran Object

Penjelasan:

1. Require_once Statement:

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

Ini adalah pernyataan yang digunakan untuk memasukkan definisi kelas dari file eksternal person.php ke dalam skrip saat ini. Ini menunjukkan bahwa file tersebut mungkin berisi definisi kelas atau kode lain yang akan digunakan di dalam skrip ini.

2. Pembuatan Objek:

```
6 // buat object baru dari kelas person
7 $person = new Person("Muhammad jhain Efendi","bengkulu");
```

Membuat objek baru dari kelas Person dengan mengirimkan dua parameter ke konstruktor kelas tersebut.

3. Manipulasi Properti Objek:

```
9 // manipulasi properti nama, alamat, negara
10 $person->nama = "JHian";
11 $person->alamat = "bengkulu";
12 $person->negara = "Indonesia";
13
```

Mengakses dan memanipulasi properti objek Person seperti nama, alamat, dan negara.

4. Menampilkan Hasil:

```
14 // menampilkan hasil
15 echo "nama = {$person->nama}" . PHP_EOL;
16 echo "alamat = {$person->alamat}" . PHP_EOL;
17 echo "negara = {$person->negara}" . PHP_EOL;
18
```

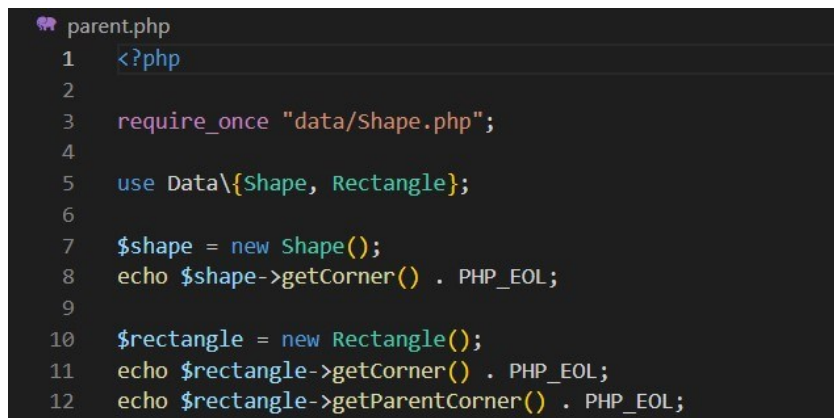
Menampilkan hasil properti objek setelah dimanipulasi.

Penjelasan:

- Objek (\$person dalam hal ini) adalah instance dari suatu kelas (Person).
- Properti (nama, alamat, negara) adalah atribut dari objek dan dapat diakses atau dimanipulasi oleh objek tersebut.

- Konstruktor (__construct method yang mungkin ada di kelas Person) digunakan untuk menginisialisasi objek saat objek dibuat.
- Konsep ini mencerminkan paradigma OOP di mana program dibangun menggunakan objek yang memiliki properti dan perilaku (metode).

7. Parent

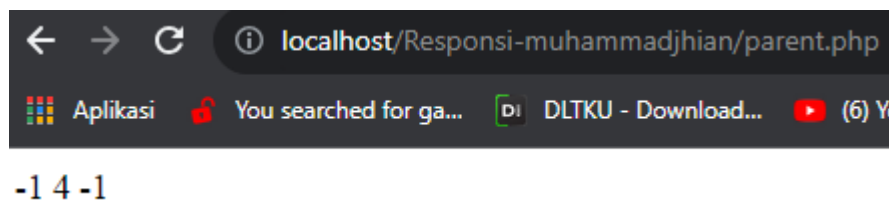


```

parent.php
1  <?php
2
3  require_once "data/Shape.php";
4
5  use Data\{Shape, Rectangle};
6
7  $shape = new Shape();
8  echo $shape->getCorner() . PHP_EOL;
9
10 $rectangle = new Rectangle();
11 echo $rectangle->getCorner() . PHP_EOL;
12 echo $rectangle->getParentCorner() . PHP_EOL;

```

Gambar 13. Parent



localhost/Responsi-muhammadhian/parent.php

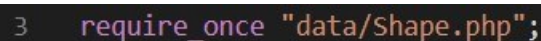
Aplikasi You searched for ga... DLTU - Download... (6) Y

-1 4 -1

Gambar 14. Luaran Parent

Penjelasan:

1. Require_once Statement:



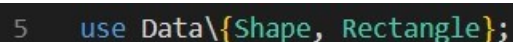
```

3  require_once "data/Shape.php";

```

Ini adalah pernyataan yang digunakan untuk memasukkan definisi kelas dari file eksternal Shape.php ke dalam skrip saat ini. Ini menunjukkan bahwa file tersebut mungkin berisi definisi kelas yang akan digunakan di dalam skrip ini.

2. Namespace dan Penggunaan Alias:



```

5  use Data\{Shape, Rectangle};

```

Ini adalah penggunaan namespace dan penggunaan alias. Namespace adalah cara untuk mengelompokkan kelas, fungsi, dan konstan ke dalam satu ruang nama. Dengan menggunakan alias (Shape dan Rectangle), kita dapat menggunakan kelas-kelas tersebut tanpa menyertakan namespace penuh setiap kali.

3. Pembuatan Objek dan Pemanggilan Metode:

```
7   $shape = new Shape();
8   echo $shape->getCorner() . PHP_EOL;
9
10  $rectangle = new Rectangle();
11  echo $rectangle->getCorner() . PHP_EOL;
12  echo $rectangle->getParentCorner() . PHP_EOL;
```

- Membuat objek dari kelas Shape dan memanggil metode getCorner().
- Membuat objek dari kelas Rectangle (yang mungkin mewarisi dari Shape) dan memanggil metode getCorner(). Selain itu, memanggil metode tambahan getParentCorner() yang mungkin merupakan metode dari kelas induk (Shape).

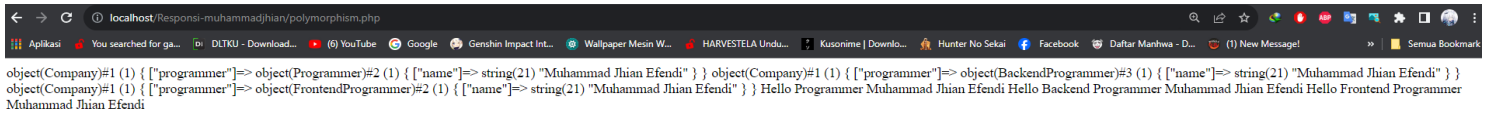
Penjelasan:

- Dari konteks kode, dapat disimpulkan bahwa kelas Rectangle mewarisi kelas Shape. Artinya, Rectangle adalah kelas anak (child class) dari Shape yang merupakan kelas induk (parent class).
- Metode getCorner() yang dipanggil pada objek \$rectangle seharusnya berasal dari kelas Shape, karena Rectangle mewarisi metode tersebut dari kelas induknya.
- Metode getParentCorner() mungkin adalah metode baru yang ditambahkan di kelas Rectangle.

8. polymorphism

```
polymorphism.php
1  <?php
2
3  require_once "data/Programmer.php";
4
5  $company = new Company();
6  $company->programmer = new Programmer("Muhammad Jhian Efendi");
7  var_dump($company);
8
9  $company->programmer = new BackendProgrammer("Muhammad Jhian Efendi");
10 var_dump($company);
11
12 $company->programmer = new FrontendProgrammer("Muhammad Jhian Efendi");
13 var_dump($company);
14
15 sayHelloProgrammer(new Programmer("Muhammad Jhian Efendi"));
16 sayHelloProgrammer(new BackendProgrammer("Muhammad Jhian Efendi"));
17 sayHelloProgrammer(new FrontendProgrammer("Muhammad Jhian Efendi"));
```

Gambar 15. Polymorphism



Gambar 16. Luaran Polymorphism

Penjelasan:

1. Pembuatan Objek dan Penggunaan Polymorphism:

```
5 $company = new Company();
6 $company->programmer = new Programmer("Muhammad Jhian Efendi");
7 var_dump($company);
8
9 $company->programmer = new BackendProgrammer("Muhammad Jhian Efendi");
10 var_dump($company);
11
12 $company->programmer = new FrontendProgrammer("Muhammad Jhian Efendi");
13 var_dump($company);
14
```

- Membuat objek dari kelas Programmer dan menginisialisasi properti programmer di objek \$company dengan objek tersebut.
- Kemudian, mengganti nilai properti programmer di \$company dengan objek dari kelas BackendProgrammer dan FrontendProgrammer.
- Melalui konsep polymorphism, objek dari kelas yang berbeda dapat diatur ke dalam properti yang sama.

2. Pemanggilan Fungsi dengan Polymorphism:

```
15 sayHelloProgrammer(new Programmer("Muhammad Jhian Efendi"));
16 sayHelloProgrammer(new BackendProgrammer("Muhammad Jhian Efendi"));
17 sayHelloProgrammer(new FrontendProgrammer("Muhammad Jhian Efendi"));
```

- Memanggil fungsi sayHelloProgrammer dengan berbagai objek yang memiliki tipe yang berbeda (Programmer, BackendProgrammer, FrontendProgrammer).
- Melalui konsep polymorphism, fungsi tersebut dapat menerima objek dari kelas yang berbeda dan memberikan respons yang sesuai.

Penjelasan:

- Polymorphism memungkinkan objek dari kelas yang berbeda untuk dianggap sebagai objek dari tipe yang sama.
- Dalam konteks ini, Programmer, BackendProgrammer, dan FrontendProgrammer semuanya dapat dianggap sebagai jenis Programmer yang lebih umum.

9.

10.

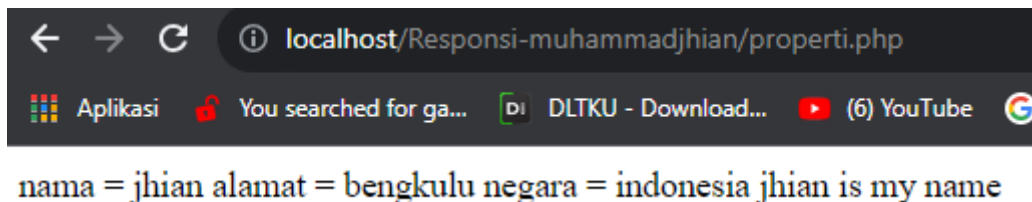
11. Properti

```

❏ 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  $person1 = new Person("Muhammad Jhian Efendi","bengkulu");
8
9  // manipulasi properti nama person
10 $person1->nama = "jhian";
11
12 // menampilkan hasil
13 echo "nama = {$person1->nama}" . PHP_EOL;
14 echo "alamat = {$person1->alamat}" . PHP_EOL;
15 echo "negara = {$person1->negara}" . PHP_EOL;
16

```

Gambar 17. Properti



nama = jhian alamat = bengkulu negara = indonesia jhian is my name

Gambar 18. Luaran Properti

Penjelasan:

1. Require_once Statement:

```

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

```

Ini adalah pernyataan yang digunakan untuk memasukkan definisi kelas dari file eksternal person.php ke dalam skrip saat ini. Ini menunjukkan bahwa file tersebut mungkin berisi definisi kelas atau kode lain yang akan digunakan di dalam skrip ini.

2. Pembuatan Objek:

```
5
6 // buat object baru dari kelas person
7 $person1 = new Person("Muhammad Jhian Efendi","bengkulu");
8
```

Membuat objek baru dari kelas Person dengan menggunakan konstruktor untuk menginisialisasi properti nama dan alamat.

3. Manipulasi Properti Objek:

```
9 // manipulasi properti nama person
10 $person1->nama = "jhian";
11
```

Mengakses dan memanipulasi properti objek Person, dalam hal ini, properti nama.

4. Menampilkan Hasil:

```
12 // menampilkan hasil
13 echo "nama = {$person1->nama}" . PHP_EOL;
14 echo "alamat = {$person1->alamat}" . PHP_EOL;
15 echo "negara = {$person1->negara}" . PHP_EOL;
```

Menampilkan hasil properti objek setelah dimanipulasi.

Penjelasan:

- Properti adalah variabel yang terkait dengan objek dan mendefinisikan karakteristik atau keadaan objek tersebut.
- Dalam contoh ini, Person memiliki properti nama, alamat, dan negara yang dapat diakses dan dimanipulasi dari luar kelas.
- Properti dapat diakses menggunakan operator panah (->), yang memberikan akses ke properti objek.

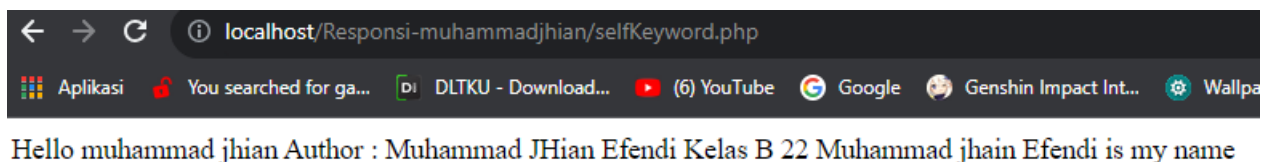
12. selfKeyword

```

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  $person1 = new Person("Muhammad jhain Efendi","bengkulu");
8
9  // panggil function
10 $person1->sayHello("muhammad jhian");
11
12 // panggil self keyword
13 $person1->info(). PHP_EOL;
14

```

Gambar 19. selfKeyword



Hello muhammad jhian Author : Muhammad JHian Efendi Kelas B 22 Muhammad jhain Efendi is my name

Gambar 20. Luaran selfKeyword

Penjelasan:

1. Require_once Statement:

```

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

```

Ini adalah pernyataan yang digunakan untuk memasukkan definisi kelas dari file eksternal person.php ke dalam skrip saat ini. Ini menunjukkan bahwa file tersebut mungkin berisi definisi kelas atau kode lain yang akan digunakan di dalam skrip ini.

2. Pembuatan Objek:

```

6  // buat object baru dari kelas person
7  $person1 = new Person("Muhammad jhain Efendi","bengkulu");
8

```

Membuat objek baru dari kelas Person dengan menggunakan konstruktor untuk menginisialisasi properti nama dan alamat.

3. Pemanggilan Fungsi:

```
9 // panggil function
10 $person1->sayHello("muhammad jhian");
11
```

Memanggil metode sayHello dari objek Person dan memberikan argumen "faishal".

4. Pemanggilan Metode yang Menggunakan self Keyword:

```
12 // panggil self keyword
13 $person1->info();
```

Memanggil metode info dari objek Person, yang kemungkinan menggunakan kata kunci self di dalamnya.

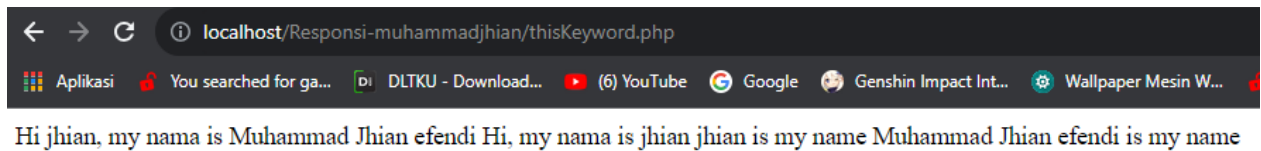
Penjelasan:

- Dalam OOP, self adalah sebuah kata kunci yang digunakan untuk merujuk pada kelas itu sendiri.
- Ketika digunakan di dalam kelas, self dapat digunakan untuk merujuk pada properti atau metode statis kelas tersebut.
- Dalam konteks metode non-statis, self biasanya digunakan untuk merujuk pada metode atau properti statis.

13. ThisKeyword

```
thisKeyword.php
1 <?php
2
3 // import data/person.php
4 require_once "data/person.php";
5
6 // buat object dari kelas person
7 $alzah = new Person("Muhammad Jhian Efendi", "bengkulu");
8
9 // tambahkan value nama di object
10 $alzah->nama = "Muhammad Jhian efendi";
11
12 // panggil function sayHelloNull dengan parameter
13 $alzah->sayHelloNull("jhian");
14
15 // buat object dari kelas person
16 $fariski = new Person("jhian", "bengkulu");
17
18 // tambahkan value nama di object
19 $fariski->nama = "jhian";
20
21 // panggil function sayHelloNull dengan parameter null
22 $fariski->sayHelloNull(null);
23
```

Gambar 21. ThisKeyword



Gambar 22. Luaran ThisKeyword

Penjelasan:

1. Require_once Statement:

Ini adalah pernyataan yang digunakan untuk memasukkan definisi kelas dari file eksternal person.php ke dalam skrip saat ini. Ini menunjukkan bahwa file tersebut mungkin berisi definisi kelas atau kode lain yang akan digunakan di dalam skrip ini.

2. Pembuatan Objek Pertama:

Membuat objek baru dari kelas Person dengan menggunakan konstruktor untuk menginisialisasi properti nama dan alamat.

3. Manipulasi Properti Objek Pertama:

Mengakses dan memanipulasi properti objek Person, dalam hal ini, properti nama.

4. Pemanggilan Metode dengan Parameter Tidak Null:

Memanggil metode sayHelloNull dari objek Person dengan memberikan parameter non-null.

5. Pembuatan Objek Kedua:

Membuat objek baru dari kelas Person dengan menggunakan konstruktor untuk menginisialisasi properti nama dan alamat. Perlu diperhatikan bahwa variabel objek yang digunakan di sini adalah \$faishal yang sama dengan objek pertama.

6. Manipulasi Properti Objek Kedua:

Mengakses dan memanipulasi properti objek Person, dalam hal ini, properti nama. Ini akan mengubah nilai properti nama dari objek yang telah dibuat sebelumnya.

7. Pemanggilan Metode dengan Parameter Null:

Memanggil metode sayHelloNull dari objek Person dengan memberikan parameter null.

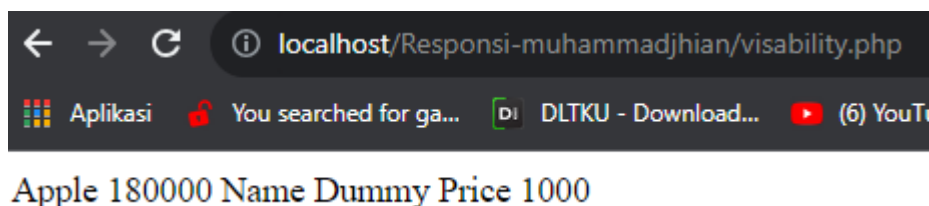
Penjelasan:

- \$this adalah variabel khusus dalam OOP yang digunakan untuk merujuk pada objek saat ini.
- Dalam konteks kode tersebut, \$this->nama digunakan untuk merujuk pada properti nama objek saat ini.

12. Visability

```
visibility.php
1  <?php
2
3  require_once "data/Product.php";
4
5  $product = new Product("Xiaomi", 20000);
6
7  // tampilkan product get name
8  // tampilkan product get price
9
10 $dummy = new ProductDummy("Suite", 1000);
11 $dummy->info();
```

Gambar 23. Visability



localhost/Responsi-muhammadjhian/visibility.php

Aplikasi You searched for ga... DLT KU - Download... (6) YouT

Apple 180000 Name Dummy Price 1000

Gambar 24. Luaran Visability

Penjelasan:

1. Pembuatan Objek Pertama:

```
5  $product = new Product("Xiaomi", 20000);
```

Membuat objek baru dari kelas Product dengan menggunakan konstruktor untuk menginisialisasi properti name dan price.

2. Pembuatan Objek Kedua (dengan Visibility Lain):

```
10 $dummy = new ProductDummy("Suite", 1000);
11 $dummy->info();
```

Membuat objek baru dari kelas ProductDummy yang mungkin memiliki tingkat visibility atau ketampakan yang berbeda untuk properti dan metodenya.

3. Konsep Visability

- Visibility mengacu pada tingkat ketampakan properti atau metode dalam kelas.
- Ada tiga tingkat visibility utama dalam OOP: public, protected, dan private.
- public: Properti atau metode dapat diakses dari mana saja, baik dari dalam kelas itu sendiri, turunan kelas, atau dari luar kelas.
- protected: Properti atau metode hanya dapat diakses dari dalam kelas itu sendiri atau turunan kelas.
- private: Properti atau metode hanya dapat diakses dari dalam kelas itu sendiri.
- Getter dan setter sering digunakan untuk mengakses dan mengubah nilai properti yang memiliki tingkat visibility protected atau private.

13. Conflict

```
data > conflict.php
1  <?php
2
3  // buat namespace data\satu
4  namespace data\satu {
5  // dengan class conflict
6      class conflict
7      {
8
9      }
10 // class sample
11 class sample
12 {
13
14 }
15 // class dummy
16 class dummy
17 {
18
19 }
20 }
21
22 // buat namespace data\dua
23 namespace data\dua {
24 // dengan class conflict
25 class conflict
26 {
27
28 }
29 }
30
```

Gambar 25. Conflict

Penjelasan:

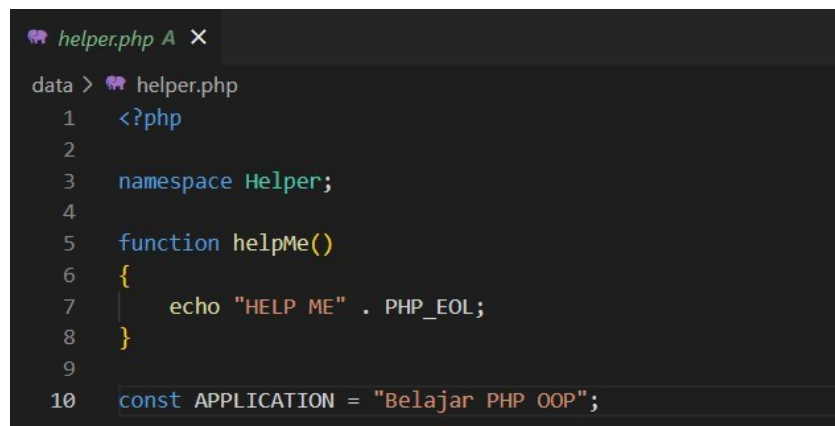
Kode yang disediakan menciptakan dua namespace, yaitu data\satu dan data\dua, dalam file conflict.php. Setiap namespace tersebut berisi tiga kelas: Conflict, Sample, dan Dummy.

Di dalam namespace `data\satu`, kelas `Conflict` memiliki metode `getMessage()` yang mengembalikan pesan khusus dari namespace tersebut. Hal yang sama berlaku untuk kelas `Sample` dan `Dummy`, masing-masing dengan metode `getMessage()` yang memberikan pesan spesifik dari namespace `data\satu`.

Selanjutnya, namespace `data\dua` juga menampilkan kelas `Conflict` yang memiliki metode `getMessage()`. Meskipun nama kelas sama dengan yang ada di `data\satu`, karena mereka berada di namespace yang berbeda, konflik dapat dihindari.

Penggunaan namespace seperti ini memungkinkan untuk mengorganisir dan mengelompokkan kode, menjaga keunikan nama di dalam setiap namespace. Penting untuk diingat bahwa penggunaan nama kelas yang sama di berbagai namespace dapat menimbulkan konflik. Dalam hal ini, solusinya adalah menggunakan fully qualified namespace atau alias saat mengakses kelas dari namespace tertentu. Ini membantu mencegah ambiguitas dan konflik yang mungkin muncul dalam pengembangan kode yang lebih besar.

14. Helper



```
data > helper.php A X
1  <?php
2
3  namespace Helper;
4
5  function helpMe()
6  {
7      echo "HELP ME" . PHP_EOL;
8  }
9
10 const APPLICATION = "Belajar PHP OOP";
```

Gambar 26. Helper **Penjelasan:**

Kode yang diberikan merupakan contoh penerapan namespace dan penggunaan fungsi serta konstanta dalam bahasa pemrograman PHP. Dalam konteks ini, kita memiliki namespace yang disebut `Helper`. Namespace ini membungkus fungsi `helpMe` dan konstanta `APPLICATION` untuk mengorganisir dan mengelompokkan kode. Fungsi `helpMe` adalah sebuah fungsi sederhana yang mencetak pesan "HELP ME" diikuti dengan karakter newline menggunakan `PHP_EOL`. Fungsi ini dapat dipanggil dari tempat lain dalam kode dengan menggunakan fully qualified namespace `Helper\helpMe()`. Konstanta `APPLICATION` adalah sebuah konstanta yang didefinisikan dalam namespace `Helper`. Konstanta ini dapat diakses dari luar namespace dengan menggunakan fully qualified namespace, yaitu `Helper\APPLICATION`. Penggunaan

namespace dalam hal ini membantu menghindari konflik nama dan memastikan bahwa fungsi dan konstanta yang didefinisikan di dalamnya dapat dibedakan dari yang mungkin ada dalam namespace lain atau di tingkat global. Secara keseluruhan, kode tersebut menunjukkan cara menggunakan namespace untuk mengelompokkan kode terkait dalam konteks tertentu, membantu meningkatkan kejelasan dan pemeliharaan kode. Fungsi dan konstanta ini juga dapat berguna untuk menyediakan fungsionalitas umum atau informasi aplikasi yang dapat diakses dari berbagai bagian dalam proyek PHP yang lebih besar.

15. Manager

```
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, my name is $this->nama" . PHP_EOL;
11     }
12 }
13
14 // buat kelas VicePresident dengan extends manager
15 class VicePresident extends Manager
16 {
17
18 }
19
```

Gambar 27. Manager

Penjelasan:

Kode yang diberikan mendefinisikan dua kelas dalam bahasa pemrograman PHP: kelas Manager dan kelas VicePresident. Kelas Manager memiliki properti nama dengan tipe data string dan fungsi sayHello, yang mencetak pesan sapaan dengan menggabungkan nama yang diterima sebagai parameter dengan nilai properti nama dari objek yang memanggil fungsi tersebut. Kemudian, kelas VicePresident didefinisikan dengan menggunakan kata kunci extends, yang menunjukkan bahwa VicePresident adalah turunan dari Manager. Dengan menggunakan pewarisan, VicePresident akan mewarisi properti dan metode yang dimiliki oleh Manager. Dengan pendekatan ini, kelas VicePresident akan memiliki properti nama dan fungsi sayHello yang sama seperti kelas Manager, tanpa perlu mendefinisikan ulang. Ini mencerminkan prinsip DRY (Don't Repeat Yourself), di mana kode yang sama atau serupa dapat direfaktor dan digunakan kembali. Pewarisan dalam pemrograman berorientasi objek memungkinkan pembentukan hierarki kelas, memfasilitasi penggunaan kembali kode, dan menyusun struktur yang memungkinkan peningkatan fungsionalitas dalam kelas turunan.

16. Person

```
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     }
18
19     // buat function sayHello nullable dengan percabangan
20     function sayHelloNull(?string $nama)
21     {
22         if (is_null($nama)) {
23             echo "Hi, my nama is $this->nama" . PHP_EOL;
24         } else {
25             echo "Hi $nama, my nama is $this->nama" . PHP_EOL;
26         }
27     }
28
29     // buat const author
30     const AUTHOR = "Muhammad JHian Efendi Kelas B 22";
31
32     // buat function info untuk self keyword
33     function info()
34     {
35         echo "Author : " . self::AUTHOR . PHP_EOL;
36     }
37
38     // buat function constructor
39     function __construct(string $nama, ?string $alamat)
40     {
41         $this->nama = $nama;
42         $this->alamat = $alamat;
43     }
44
45     // buat function destructor
46     function __destruct()
47     {
48         echo " $this->nama is my name" . PHP_EOL;
```

Gambar 28. Person

Penjelasan:

Kode PHP yang disediakan mendefinisikan sebuah kelas bernama Person, yang bertujuan merepresentasikan individu dalam sebuah program. Kelas ini memiliki beberapa properti, seperti \$nama (dengan tipe data string), \$alamat (sebagai nullable string yang dapat bernilai null), dan \$negara (dengan nilai default "Indonesia"). Properti-properti ini merepresentasikan informasi personal seperti nama, alamat, dan negara asal individu. Selain properti, kelas Person juga memiliki beberapa metode, antara lain sayHello yang mencetak pesan sapaan berdasarkan parameter yang diberikan, serta sayHelloNull yang memiliki kemampuan menangani nilai

nullable dalam parameter dan memberikan pesan sapaan sesuai kondisinya. Kelas ini juga menggunakan konsep konstanta dengan adanya konstanta AUTHOR yang bersifat statis, yang memberikan informasi terkait penulis kelas ini. Fungsi info dalam kelas juga menunjukkan penggunaan kata kunci self untuk mengakses konstanta tersebut. Lebih lanjut, kelas Person memiliki fungsi konstruktor __construct, yang digunakan untuk menginisialisasi properti objek saat pembuatannya. Fungsi ini memberikan kemampuan untuk memberikan nilai awal pada properti-properti objek. Sebagai pelengkap, kelas juga memiliki fungsi destruktur __destruct, yang memberikan pesan saat objek dihancurkan, menandakan akhir siklus hidup objek tersebut. Secara keseluruhan, kelas Person menunjukkan implementasi dasar dari konsep pemrograman berorientasi objek, dengan properti, metode, konstanta, konstruktor, dan destruktur yang membentuk struktur dasar untuk merepresentasikan dan berinteraksi dengan objek individu.

18. Product

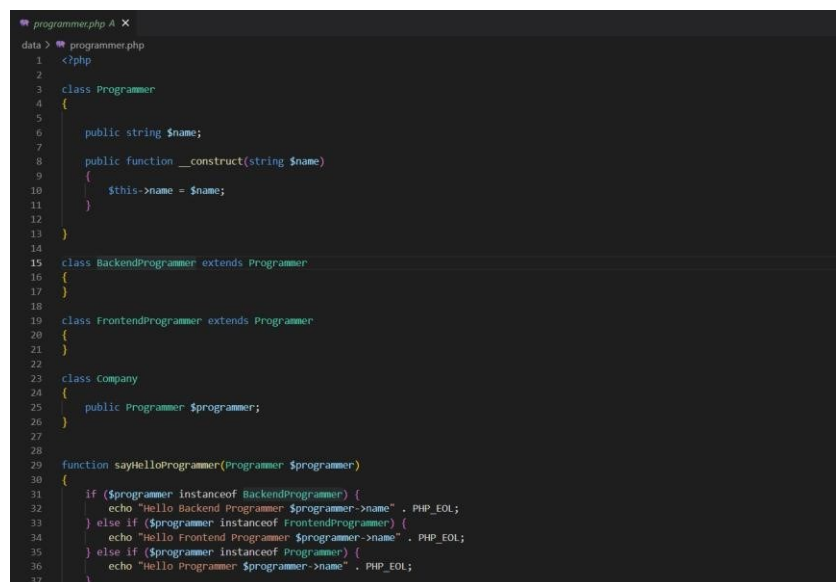
```
data > product.php
1  <?php
2
3  class Product
4  {
5      protected string $name;
6      protected int $price;
7
8      public function __construct(string $name, int $price)
9      {
10         $this->name = $name;
11         $this->price = $price;
12     }
13
14     public function getName(): string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

Gambar 28. Product **Penjelasan:**

Kode PHP di atas mendefinisikan dua kelas, yaitu Product dan ProductDummy, yang mewakili entitas produk dalam suatu sistem. Kelas Product memiliki dua properti, yakni \$name (dengan tipe data string) dan \$price (dengan tipe data int), yang mewakili nama dan harga produk. Konstruktor __construct digunakan untuk menginisialisasi nilai properti saat objek dibuat. Kelas Product juga menyediakan dua metode, yaitu getName dan getPrice, yang mengembalikan nilai properti nama dan harga produk, masing-masing. Properti dalam kelas ini memiliki tingkat proteksi protected, sehingga dapat diakses oleh kelas turunannya. Kelas ProductDummy merupakan turunan dari kelas Product, sehingga mewarisi properti dan metode yang dimilikinya. Kelas ini menambahkan metode info yang mencetak informasi tambahan, yaitu nama dan harga produk, sebagai contoh implementasi tambahan pada kelas turunan.

Dengan struktur ini, konsep pewarisan dan penggunaan tingkat proteksi dalam OOP tercermin. Kelas ProductDummy dapat digunakan untuk menciptakan objek produk dengan memanfaatkan properti dan metode dari kelas induknya (Product) dan menambahkan fungsionalitas tambahan sesuai kebutuhan.

19. Programmer



```
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 }
```

Gambar 29. Programmer

Penjelasan:

Kode PHP di atas menciptakan struktur kelas untuk merepresentasikan peran dalam dunia pemrograman, dengan fokus pada konsep pewarisan dan polimorfisme dalam pemrograman berorientasi objek (OOP). Kelas utama, Programmer, memiliki properti \$name dan konstruktor untuk menginisialisasi nilai properti tersebut. Selanjutnya, terdapat dua kelas turunan, yaitu BackendProgrammer dan FrontendProgrammer, yang mewarisi properti dan metode dari kelas Programmer. Konsep pewarisan memungkinkan kelas turunan untuk memanfaatkan fungsionalitas dari kelas induknya. Dalam hal ini, baik BackendProgrammer maupun FrontendProgrammer mewarisi properti \$name dan konstruktor dari kelas Programmer. Kelas Company menunjukkan hubungan antara objek perusahaan dan programmer. Properti \$programmer dalam kelas Company didefinisikan sebagai objek bertipe Programmer, memberikan fleksibilitas untuk menyimpan objek dari kelas turunan Programmer. Fungsi sayHelloProgrammer merupakan contoh polimorfisme, di mana berbagai tipe programmer dapat diterima sebagai parameter. Dengan menggunakan instanceof untuk memeriksa jenis programmer, fungsi memberikan pesan sapaan yang sesuai dengan jenis programmer yang

diterima. Secara keseluruhan, struktur kelas dan fungsi dalam kode tersebut menciptakan hierarki yang mencerminkan hubungan antara berbagai jenis programmer, memanfaatkan konsep pewarisan dan polimorfisme untuk mencapai fleksibilitas dan reusabilitas dalam desain OOP.

20. Shape

```
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 class Rectangle extends Shape
15 {
16
17     public function getCorner()
18     {
19         return 4;
20     }
21
22     public function getParentCorner()
23     {
24         return parent::getCorner();
25     }
26 }
27
28 }
```

Gambar 30. Shape

Penjelasan:

Kode PHP di atas menciptakan struktur kelas untuk merepresentasikan peran dalam dunia pemrograman, dengan fokus pada konsep pewarisan dan polimorfisme dalam pemrograman berorientasi objek (OOP). Kelas utama, Programmer, memiliki properti \$name dan konstruktor untuk menginisialisasi nilai properti tersebut. Selanjutnya, terdapat dua kelas turunan, yaitu BackendProgrammer dan FrontendProgrammer, yang mewarisi properti dan metode dari kelas Programmer. Konsep pewarisan memungkinkan kelas turunan untuk memanfaatkan fungsionalitas dari kelas induknya. Dalam hal ini, baik BackendProgrammer maupun FrontendProgrammer mewarisi properti \$name dan konstruktor dari kelas Programmer. Kelas Company menunjukkan hubungan antara objek perusahaan dan programmer. Properti \$programmer dalam kelas Company didefinisikan sebagai objek bertipe Programmer, memberikan fleksibilitas untuk menyimpan objek dari kelas turunan Programmer. Fungsi sayHelloProgrammer merupakan contoh polimorfisme, di mana berbagai tipe programmer dapat diterima sebagai parameter. Dengan menggunakan instanceof untuk memeriksa jenis programmer, fungsi memberikan pesan sapaan yang sesuai dengan jenis programmer yang diterima.

21. Import/import Alias

Penjelasan:

Kode PHP di atas mencerminkan praktik pengelolaan berkas dan penggunaan namespace dalam lingkungan pemrograman. Pernyataan `require_once` digunakan untuk memasukkan dua berkas, yaitu `"Conflict.php"` dan `"Helper.php"`, yang terletak dalam direktori `"data"`. Pemakaian `require_once` memastikan bahwa berkas tersebut hanya diimpor sekali, menghindari potensi konflik atau duplikasi selama eksekusi skrip. Selanjutnya, pernyataan `use` digunakan untuk mengimpor kelas, fungsi, dan konstanta dari berkas-berkas tersebut ke dalam skrip utama. Dengan menggunakan `use`, kita dapat secara jelas menentukan namespace asal elemen-elemen yang diimpor. Setelah impor, kelas `Conflict` dari namespace `Data\One` diinisiasi menjadi objek `$conflict1`, sedangkan objek `$conflict2` dibuat menggunakan kelas `Conflict` dari namespace `Data\Two`. Ini menunjukkan penggunaan namespace untuk membedakan antara kelas-kelas dengan nama yang sama yang terdapat dalam namespace yang berbeda. Fungsi `helpMe` dari namespace `Helper` dipanggil, mencetak pesan `"HELP ME"` sesuai dengan implementasi fungsi tersebut. Selanjutnya, konstanta `APPLICATION` diambil dari namespace `Helper` dan ditampilkan menggunakan pernyataan `echo`. Kode ini mencerminkan praktik baik dalam pemrograman berorientasi objek, memanfaatkan namespace untuk mengelola skala besar kode dan menghindari konflik nama antar-kelas, fungsi, dan konstanta. Pemakaian `use` juga membantu meningkatkan kejelasan kode dengan menyatakan secara eksplisit asal-usul elemenelemen yang diimpor.