

Nama : Ahmad Sobari  
NPM : G1F022058  
PRODI : Sistem Infromasi  
MATKUL : Proyek Pemrograman Berorientasi Object

## Responsi

### Proyek Pemrograman Berorientasi Object

#### Soal:

1. Silahkan lakukan git clone repositori dari <https://github.com/alzahfariski/bahan-ajar-pbo> (silahkan liat di youtube caranya).
2. lengkapi code php yang belum lengkap sehingga setiap file dapat di run dan tidak memunculkan error.
3. upload atau lakukan git push ke akun git kalian masing-masing.
4. salin url lalu kumpulkan dengan berikan penjelasan mengenai pemahaman kalian secara descriptive (contoh penjelasan mengenai file object.php menggunakan code apa saja dan berfungsi untuk apa) penjelelasan kalian akan mempengaruhi penilaian.

#### Pembahasan:

Link github yang menjadi lokasi upload file yang dapat di akses pengguna lain

[https://github.com/Luxon514/G1F022058\\_Ahmad\\_sobari.git](https://github.com/Luxon514/G1F022058_Ahmad_sobari.git)

#### 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", "PHP Pemrograman Berorientasi Object");
8
9  // buat const app version
10 const APP_VERSION = "2.0";
11
12 // tampilkan hasil
13 echo APPLICATION . PHP_EOL;
14 echo APP_VERSION . PHP_EOL;
15 echo Person::AUTHOR . PHP_EOL;
16
```

Gambar 1. Susunan Program constant

```
PHP Pemrograman Berorientasi Object 2.0 Kelas B 22
```

Gambar 2 output program constant

#### Penjelasan:

### 1) Import class person

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

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", "PHP Pemrograman Berorientasi Object");
8
```

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 constant

```
9 // buat const app version
10 const APP_VERSION = "2.0";
11
```

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

```
12 // tampilkan hasil
13 echo APPLICATION . PHP_EOL;
14 echo APP_VERSION . PHP_EOL;
15 echo Person::AUTHOR . PHP_EOL;
16
```

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

```
1 <?php
2
3 // import data/person.php
4 require_once "data/Person.php";
5
6 // buat object new person dengan 2 parameter
7 $ahmad = new Person("Ahmad Sobari", "Curup");
8
9 // vardump object
10 var_dump($ahmad);
11
```

Gambar 2. susunan program constractor

```
object(Person)#1 (3) { ["nama"]=> string(12) "Ahmad Sobari" ["alamat"]=> string(5) "Curup" ["negara"]=> string(9) "Indonesia" } Object person Ahmad Sobari is destroyed
```

Gambar 3. Output program 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 $ahmad = new Person("Ahmad Sobari", "Curup");
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

```

9 // vardump object
10 var_dump($ahmad);
11

```

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 \$ahmad.

## 3. Dectructor

```

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 $ahmad = new Person("Ahmad", "Curup");
8 $sobari = new Person("Sobari", "Bengko");
9
10 // tambahkan echo "Program Selesai" . PHP_EOL;
11 echo "Program Selesai" . PHP_EOL;
12

```

Gambar 4. Susunan program destructor

```

Program Selesai Object person Sobari is destroyed Object person Ahmad is destroyed

```

Gambar 5. Output program 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 sobari dihancurkan." dan "Objek ahmad dihancurkan."

## 4. Function

```

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("Sobari", "Bengko");
8
9 // panggil function
10 $person1->sayHello("Ahmad");
11

```

Gambar 6. Susunan program function

```
Hello Ahmad Object person Sobari is destroyed
```

Gambar 7. Output program 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

```
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("Sobari", "Bengko");
```

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

#### 3) Memanggil method function "say hello"

```
8
9 // panggil function
10 $person1->sayHello("Ahmad");
11
```

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

## 5. 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 = "Ahmad";
9 $manager->sayHello("yo");
10
11 // buat object new vicepresident dan tambahkan value nama kemudian panggil function
12 $vp = new VicePresident();
13 $vp->nama = "Sobari";
14 $vp->sayHello("hello");
15
```

Gambar 8. Susunan program inheritance

```
Hi yo, my name is Ahmad Hi hello, my name is Sobari
```

Gambar 9. Output program inheritance

### 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.

### 1) Require\_once Statement

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

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 = "Ahmad";
9 $manager->sayHello("yo");
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 $vp = new VicePresident();
13 $vp->nama = "Sobari";
14 $vp->sayHello("hello");
15
```

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

## 6. Object

```
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("Ahmad","Curup");
8
9 // manipulasi properti nama, alamat, negara
10 $person->nama = "Ahmad";
11 $person->alamat = "Curup";
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 10. Susunan program Object

```
nama = Ahmad alamat = Curup negara = Indonesia Object person Ahmad is destroyed
```

Gambar 11. Output program object

### 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).

#### 1) Require\_once Statement

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

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 object

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

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

#### 3) Manipulasi property object

```
9 // manipulasi properti nama, alamat, negara
10 $person->nama = "Ahmad";
11 $person->alamat = "Curup";
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.

### 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 12. Susunan program parent

```
-5.5 -5
```

Gambar 13. Output program parent

#### 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.

### 1) Require\_once Statement

```
1 <?php
2
3 require_once "data/Shape.php";
4
```

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

```
4
5 use Data\{Shape, Rectangle};
6
```

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

```
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;
```

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).

## 8. Polymorphism

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

Gambar 14. Susunan program polymorphism

```
object(Company)#1 (1) { ["programmer"]=> object(Programmer)#2 (1) { ["name"]=> string(5) "Luxon" } }
object(Company)#1 (1) { ["programmer"]=> object(BackendProgrammer)#3 (1) { ["name"]=> string(5) "Squzi" } }
object(Company)#1 (1) { ["programmer"]=> object(FrontendProgrammer)#2 (1) { ["name"]=> string(3) "Ziu" } }
Hello Programmer Luxon Hello Backend Programmer Squzi Hello Frontend Programmer Ziu
```

Gambar 15. Output program polymorphism

### 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.

#### 1) Pembuatan Objek dan Penggunaan Polymorphism

```
5 $company = new Company();
6 $company->programmer = new Programmer("Luxon");
7 var_dump($company);
8
9 $company->programmer = new BackendProgrammer("Squzi");
10 var_dump($company);
11
12 $company->programmer = new FrontendProgrammer("Ziu");
13 var_dump($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 yang berasal nya 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("Luxon"));
16 sayHelloProgrammer(new BackendProgrammer("Squzi"));
17 sayHelloProgrammer(new FrontendProgrammer("Ziu"));
```

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.

## 9. Properti

```
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("Ahmad", "Curup");
8
9 // manipulasi properti nama person
10 $person1->nama = "Ahmad";
11
12 // menampilkan hasil
13 echo "nama = {$person1->nama}" . PHP_EOL;
14 echo "alamat = {$person1->alamat}" . PHP_EOL;
15 echo "negara = {$person1->negara}" . PHP_EOL;
16
```

Gambar 16. Susunan program Properti



```
nama = Ahmad alamat = Curup negara = indonesia Object person Ahmad is destroyed
```

Gambar 17. Output program property

### 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.

#### 1) Require\_once Statement

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

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 Object

```
5
6 // buat object baru dari kelas person
7 $person1 = new Person("Ahmad", "Curup");
8
```

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

#### 3) Manipulasi Properti Objek

```
8
9 // manipulasi properti nama person
10 $person1->nama = "Ahmad";
11
```

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

#### 4) Menampilkan Hasil

```
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
```

Menampilkan hasil properti objek setelah dimanipulasi.

## 10. 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("Ahmad", "Curup");
8
9 // panggil function
10 $person1->sayHello("Ahmad");
11
12 // panggil self keyword
13 $person1->info();
14
```

Gambar 18. Susunan program Selfkeyword

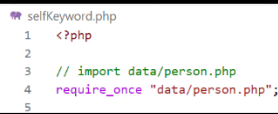


Gambar 19. Output program Selfkeyword

Penjelasan:

Dalam OOP, self adalah sebuah kata kunci yang digunakan untuk merujuk pada kelas tersebut 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.

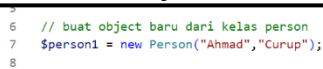
#### 1) Require\_once Statement



```
selfKeyword.php
1 <?php
2
3 // import data/person.php
4 require_once "data/person.php";
5
```

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

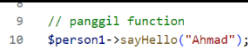


```

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

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

#### 3) Pemanggilan Fungsi

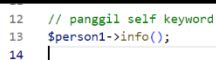


```

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

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

#### 4) Pemanggilan Metode yang Menggunakan self Keyword



```

12 // panggil self keyword
13 $person1->info();
14 |
```

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

## 11. 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  $ahmad = new Person("Ahmad", "Curup");
8
9  // tambahkan value nama di object
10 $ahmad->nama = "Ahmad";
11
12 // panggil function sayHelloNull dengan parameter
13 $ahmad->sayHelloNull("All");
14
15 // buat object dari kelas person
16 $sobari = new Person("Sobari", "Bengko");
17
18 // tambahkan value nama di object
19 $sobari->nama = "Sobari";
20
21 // panggil function sayHelloNull dengan parameter null
22 $sobari->sayHelloNull(null);
23

```

Gambar 20. Susunan program Thiskeyword

```

Hi All, my nama is Ahmad Hi, my nama is Sobari Object person Sobari is destroyed Object person Ahmad is destroyed

```

Gambar 21. Output program Thiskeyword

### 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.

#### 1) Require\_once Statement

```

thisKeyword.php
1  <?php
2
3  // import data/person.php
4  require_once "data/person.php";
5

```

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

```

6  // buat object dari kelas person
7  $ahmad = new Person("Ahmad", "Curup");
8

```

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

#### 3) Manipulasi Properti Objek Pertama

```

9  // tambahkan value nama di object
10 $ahmad->nama = "Ahmad";
11

```

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

#### 4) Pemanggilan Metode dengan Parameter Tidak Null

```

12 // panggil function sayHelloNull dengan parameter
13 $ahmad->sayHelloNull("All");
14

```

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

#### 5) Pembuatan Objek Kedua

```

14
15 // buat object dari kelas person
16 $sobari = new Person("Sobari", "Bengko");
17

```

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

```

18 // tambahkan value nama di object
19 $sobari->nama = "Sobari";
20

```

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 kedua

```

20
21 // panggil function sayHelloNull dengan parameter null
22 $sobari->sayHelloNull(null);
23 |

```

Memanggil metode sayHelloNull dari objek Person dengan memberikan parameter null.

## 12. Visability

```

visibility.php
1 <?php
2
3 require_once "data/Product.php";
4
5 $product = new Product("Adamantium", 10000000);
6
7 // tampilkan product get name
8 // tampilkan product get price
9
10 $graphine = new ProductDummy("Graphine", 5000000);
11 $graphine->info();

```

Gambar 22. Susunan program visibility

```

Name Graphine Price 5000000

```

Gambar 23. Output program visibility

### Penjelasan:

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.

#### 1) Pembuatan Objek Pertama

```

visibility.php
1 <?php
2
3 require_once "data/Product.php";
4
5 $product = new Product("Adamantium", 10000000);
6

```

Membuat sebuah atau beberapa objek baru dari kelas Product dengan menggunakan konstruktor untuk menginisialisasi properti name dan price agar sesuai dengan apa yang diinginkan pengguna.

## 2) Pembuatan Objek Kedua (dengan Visibility Lain)

```
10 $graphine = new ProductDummy("Graphine", 5000000);  
11 $graphine->info();
```

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

## 13. Conflict

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

Gambar 24. Susunan program 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.

## 14. Helper

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

Gambar 25. Susunan program helper

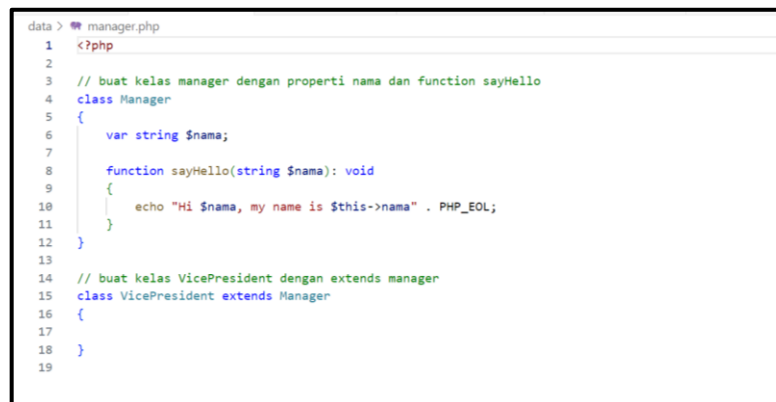
### Penjelasan:

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 fullyqualified 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 fungsidan 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 26. Susunan program manager

### Penejelasan:

Kode yang diberikan mendefinisikan dua kelas dalam bahasa pemrograman PHP: kelasManager 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 fungsinya 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 direfaktorkan 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 name is $this->nama" . PHP_EOL;
24         } else {
25             echo "Hi $nama, my name is $this->nama" . PHP_EOL;
26         }
27     }
28
29     // buat const author
30     const AUTHOR = "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 "Object person $this->nama is destroyed" . PHP_EOL;
49     }
50 }
51
```

Gambar 27. Susunan program 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.

## 17. 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 27. Susunan program 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.

## 18. Programmer

The image shows a code editor window with a file named 'programmer.php'. The code defines a 'Programmer' class with a public string property '\$name' and a constructor '\_\_construct' that sets '\$this->name' to the passed '\$name'. It also defines two subclasses: 'BackendProgrammer' and 'FrontendProgrammer', both extending 'Programmer'. A 'Company' class is defined with a public 'Programmer' property '\$programmer'. Finally, a function 'sayHelloProgrammer' is defined, which takes a 'Programmer' object and uses 'instanceof' to check if it's a 'BackendProgrammer', 'FrontendProgrammer', or a general 'Programmer', then echoes a corresponding message.

```
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     }
```

Gambar 28. Susunan program 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.

## 19. Shape

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

Gambar 29. Susunan program 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.