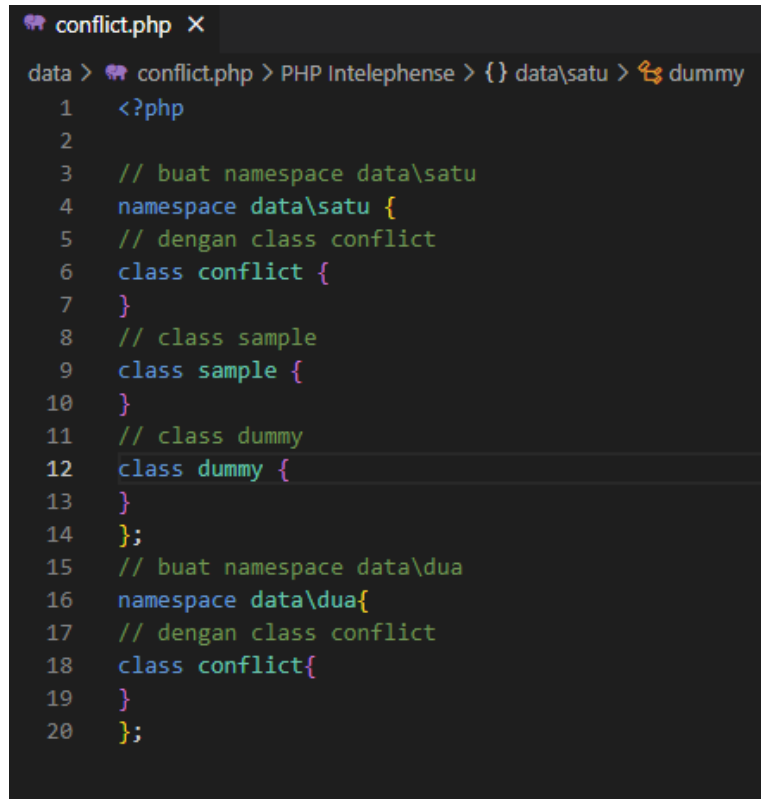


Nama : Ahmad Fadhila
NPM : G1F022005

Responsi

- 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 // buat namespace data\dua
16 namespace data\dua{
17 // dengan class conflict
18 class conflict{
19 }
20 };
```

Penjelasan

Kode PHP yang diberikan mencakup penggunaan namespace untuk mengorganisir kelas-kelas dalam dua namespace yang berbeda.

Class sample: Ini adalah contoh kelas sample yang berada dalam namespace data\satu.

Class dummy: Ini adalah contoh kelas dummy yang berada dalam namespace data\satu.

Class conflict: Ini adalah kelas dengan nama conflict yang berada dalam namespace data\dua. terdapat nama yang sama dengan kelas di namespace data\satu, tetapi karena mereka berada dalam namespace yang berbeda, tidak ada konflik

Dengan menggunakan namespace, kita dapat menghindari konflik nama kelas yang sama ketika kita memiliki beberapa kelas dengan nama yang sama dalam proyek PHP yang lebih besar

- helper.php

```
conflict.php  helper.php X
data > helper.php > ...
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 Fadhil";
```

Penjelasan

Kode ini menciptakan namespace Helper yang mengandung sebuah fungsi (helpMe()) dan sebuah konstanta (APPLICATION). Dengan menempatkan kode di dalam namespace Helper, kita dapat memanggil fungsi dan mengakses konstanta menggunakan nama namespace tersebut. Dengan menggunakan namespace, kita dapat menghindari konflik nama dan memastikan bahwa fungsi dan konstanta yang kita deklarasikan tidak bentrok dengan kode dari namespace lain dalam proyek PHP yang lebih besar. Kode ini menciptakan namespace Helper yang mengandung sebuah fungsi (helpMe()) dan sebuah konstanta (APPLICATION). Dengan menempatkan kode di dalam namespace Helper, kita dapat memanggil fungsi dan mengakses konstanta menggunakan nama namespace tersebut.

- manager.php

```
conflict.php  manager.php X
data > manager.php > ...
1  <?php
2
3  // buat kelas manager dengan properti nama dan function say
4  class Manager
5  {
6      var string $nama;
7
8      function sayHello(string $nama)
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
```

Penjelasan

Kode PHP yang diberikan mendefinisikan dua kelas, yaitu Manager dan VicePresident, serta menggunakan konsep pewarisan (inheritance). Properti nama: Kelas Manager memiliki properti nama yang bertipe string. Properti ini dapat digunakan untuk menyimpan nama manajer.

Metode sayHello: Kelas Manager memiliki metode sayHello yang menerima parameter \$nama. Metode ini mencetak pesan sapaan yang mengandung nama yang diberikan dan nama manajer dari properti nama. extends Manager: Kelas VicePresident meng-extends (mewarisi) kelas Manager. Artinya, VicePresident akan mewarisi semua properti dan metode dari kelas Manager. Dengan kata lain, VicePresident akan memiliki properti nama dan metode sayHello seperti yang didefinisikan di kelas Manager. Dengan pewarisan, kelas VicePresident dapat menggunakan fungsionalitas dari kelas Manager tanpa perlu mendefinisikannya ulang.

- person.php

```
conflict.php  person.php x
data > person.php > ...
1  <?php
2
3  // membuat kelas person
4  class Person{
5      // membuat properti
6      var string $nama;
7      // gunakan nullable properti
8      var ?string $alamat = null;
9      // gunakan default value untuk properti
10     var string $negara = "Indonesia";
11     // buat function sayHello
12     function sayHello(string $nama){
13         echo "Hei $nama" . PHP_EOL;
14     }
15     // buat function sayHello nullable dengan percabangan
16     function sayHelloNull(?string $nama)
17     {
18         if (is_null($nama)) {
19             echo "Hi, my nama is $this->nama" . PHP_EOL;
20         } else {
21             echo "Hi $nama, my nama is $this->nama" . PHP_EOL;
22         }
23     }
24
25     // buat const author
26     const AUTHOR = "Ahmad Fadhila";
27     // buat function info untuk self keyword
28     function info()
29     {
30         echo "Author : " . self::AUTHOR . PHP_EOL;
31     }
32     // buat function constructor
33     function __construct(string $nama, ?string $alamat)
34     {
35         $this->nama = $nama;
36         $this->alamat = $alamat;
37     }
38
39     // buat function destructor
40     function __destruct()
41     {
42         echo "Object person $this->nama is destroyed" . PHP_EOL;
43     }
44 }
45
```

Penjelasan

Kelas Person merupakan representasi dari entitas individu dengan beberapa atribut dan perilaku. Dalam kelas ini, terdapat beberapa elemen penting:

a. Properti Kelas:

\$nama: Menyimpan nama individu dan memiliki tipe data string.

\$alamat: Merupakan properti nullable yang dapat menyimpan alamat individu.

\$negara: Properti ini memiliki nilai default "Indonesia" dan menunjukkan negara asal individu.

b. Metode Kelas:

sayHello(string \$nama): Metode ini mencetak pesan sapaan menggunakan nama yang diberikan.

sayHelloNull(?string \$nama): Metode ini memberikan sapaan yang berbeda tergantung apakah parameter nama null atau tidak.

info(): Metode ini mencetak informasi tentang penulis kelas menggunakan konstanta AUTHOR.

__construct(string \$nama, ?string \$alamat): Metode konstruktor digunakan untuk menginisialisasi objek dengan memberikan nilai awal ke properti nama dan alamat.

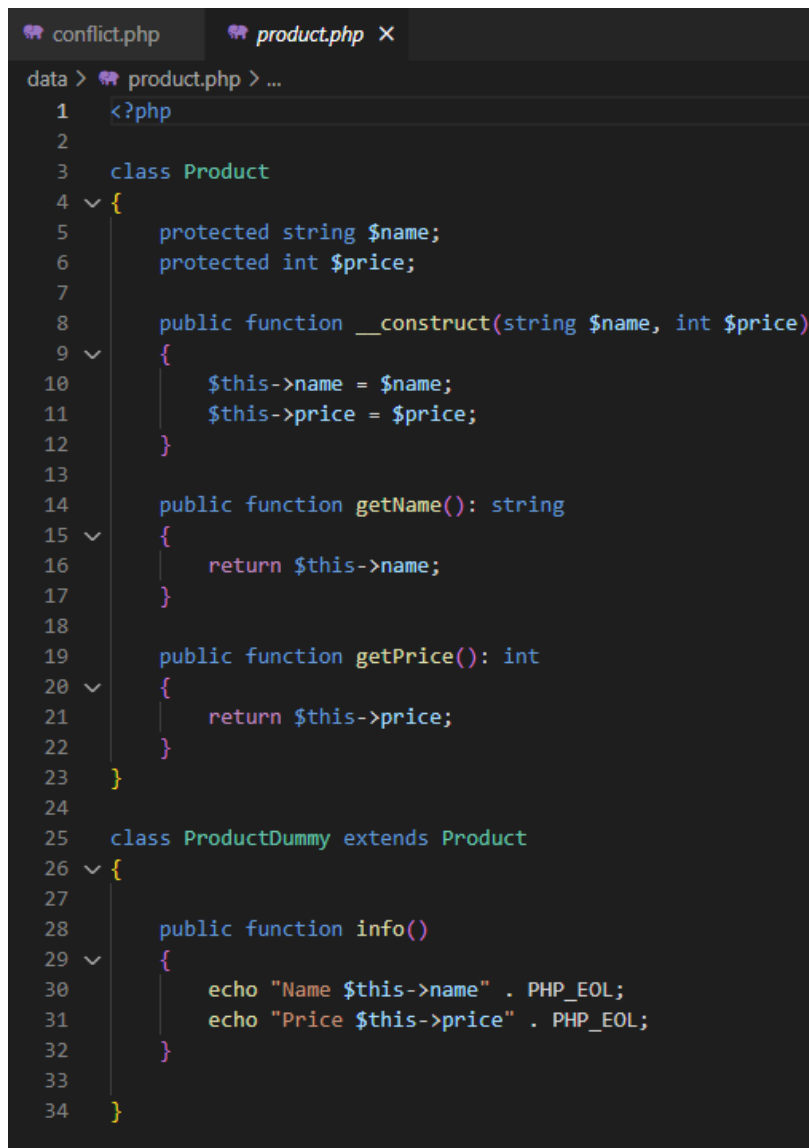
__destruct(): Metode destruktur mencetak pesan ketika objek Person dihancurkan atau keluar dari lingkup.

c. Konstanta Kelas:

AUTHOR: Konstanta ini menyimpan nama penulis kelas, yaitu "Ahmad Fadhila". Konstanta bersifat tetap dan tidak dapat diubah setelah dideklarasikan

Dengan struktur seperti ini, kelas Person dapat digunakan untuk merepresentasikan individu dengan berbagai atribut dan perilaku yang berguna dalam suatu program.

- 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 }

```

Penjelasan

Kode di atas mengilustrasikan penggunaan konsep dasar dalam pemrograman berorientasi objek (OOP) menggunakan bahasa pemrograman PHP. Dalam konteks ini, terdapat dua kelas, yaitu `Product` dan `ProductDummy`, yang memanfaatkan konsep pewarisan.

Kelas `Product`: Properti: `protected string $name` dan `protected int $price` digunakan untuk menyimpan nama dan harga produk. Properti tersebut diberi tingkat akses `protected`, yang berarti mereka dapat diakses oleh kelas turunannya (`ProductDummy`), namun tidak dapat diakses secara langsung dari luar kelas. Metode Konstruktor: Metode `__construct` digunakan untuk menginisialisasi objek dengan memberikan nilai awal ke properti nama (`$name`) dan harga (`$price`).

Metode `getName()`: Mengembalikan nilai properti `$name`.

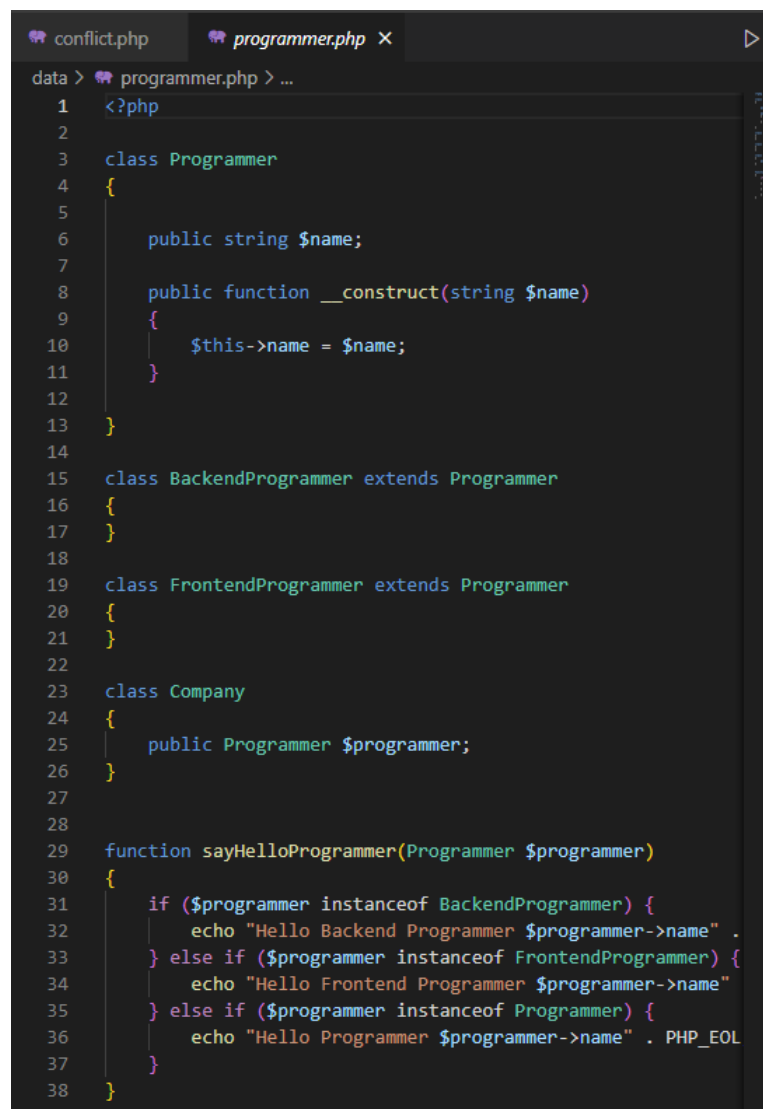
Metode `getPrice()`: Mengembalikan nilai properti `$price`.

Kelas ProductDummy (Turunan dari Product): Pewarisan: Kelas ProductDummy mewarisi sifat dan metode dari kelas Product. Dengan demikian, ProductDummy memiliki akses ke properti dan metode yang dilindungi (\$name dan \$price).

Metode info(): Mencetak informasi produk, termasuk nama dan harga, ke layar menggunakan properti yang diwarisi.

Dengan menggunakan konsep pewarisan, kode ini mendemonstrasikan cara mengorganisir dan mengakses properti serta metode dari kelas dasar (Product) dalam kelas turunan (ProductDummy). Ini membantu dalam menciptakan hierarki kelas yang lebih terstruktur dan memungkinkan penggunaan kembali kode.

- programmer.php



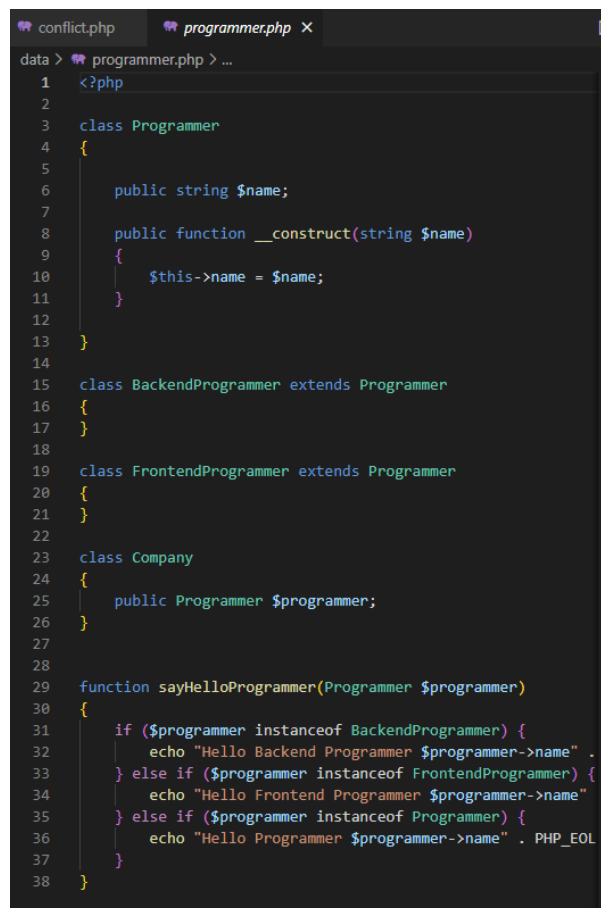
```
1 <?php
2
3 class Programmer
4 {
5
6     public string $name;
7
8     public function __construct(string $name)
9     {
10         $this->name = $name;
11     }
12 }
13
14
15 class BackendProgrammer extends Programmer
16 {
17 }
18
19 class FrontendProgrammer extends Programmer
20 {
21 }
22
23 class 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 $programmer->name" .
33     } else if ($programmer instanceof FrontendProgrammer) {
34         echo "Hello Frontend Programmer $programmer->name"
35     } else if ($programmer instanceof Programmer) {
36         echo "Hello Programmer $programmer->name" . PHP_EOL
37     }
38 }
```

Penjelasan

Kode di atas mengilustrasikan konsep dasar dalam pemrograman berorientasi objek (OOP) dengan menggunakan bahasa pemrograman PHP. Terdapat beberapa kelas, yaitu Programmer, BackendProgrammer, FrontendProgrammer, dan Company. Selain itu, terdapat fungsi sayHelloProgrammer.

Dengan menggunakan konsep pewarisan, program ini menunjukkan bagaimana kelas turunan (BackendProgrammer dan FrontendProgrammer) dapat memiliki sifat dan perilaku yang sama dengan kelas dasarnya (Programmer). Fungsi sayHelloProgrammer juga menunjukkan penggunaan instanceof untuk mengecek tipe objek dan memberikan respons yang sesuai. Hal ini membantu dalam membuat kode yang lebih dinamis dan fleksibel.

- shape.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 class BackendProgrammer extends Programmer
15 {
16 }
17
18 class FrontendProgrammer extends Programmer
19 {
20 }
21
22 class Company
23 {
24     public Programmer $programmer;
25 }
26
27
28
29 function sayHelloProgrammer(Programmer $programmer)
30 {
31     if ($programmer instanceof BackendProgrammer) {
32         echo "Hello Backend Programmer $programmer->name" .
33     } else if ($programmer instanceof FrontendProgrammer) {
34         echo "Hello Frontend Programmer $programmer->name"
35     } else if ($programmer instanceof Programmer) {
36         echo "Hello Programmer $programmer->name" . PHP_EOL
37     }
38 }
```

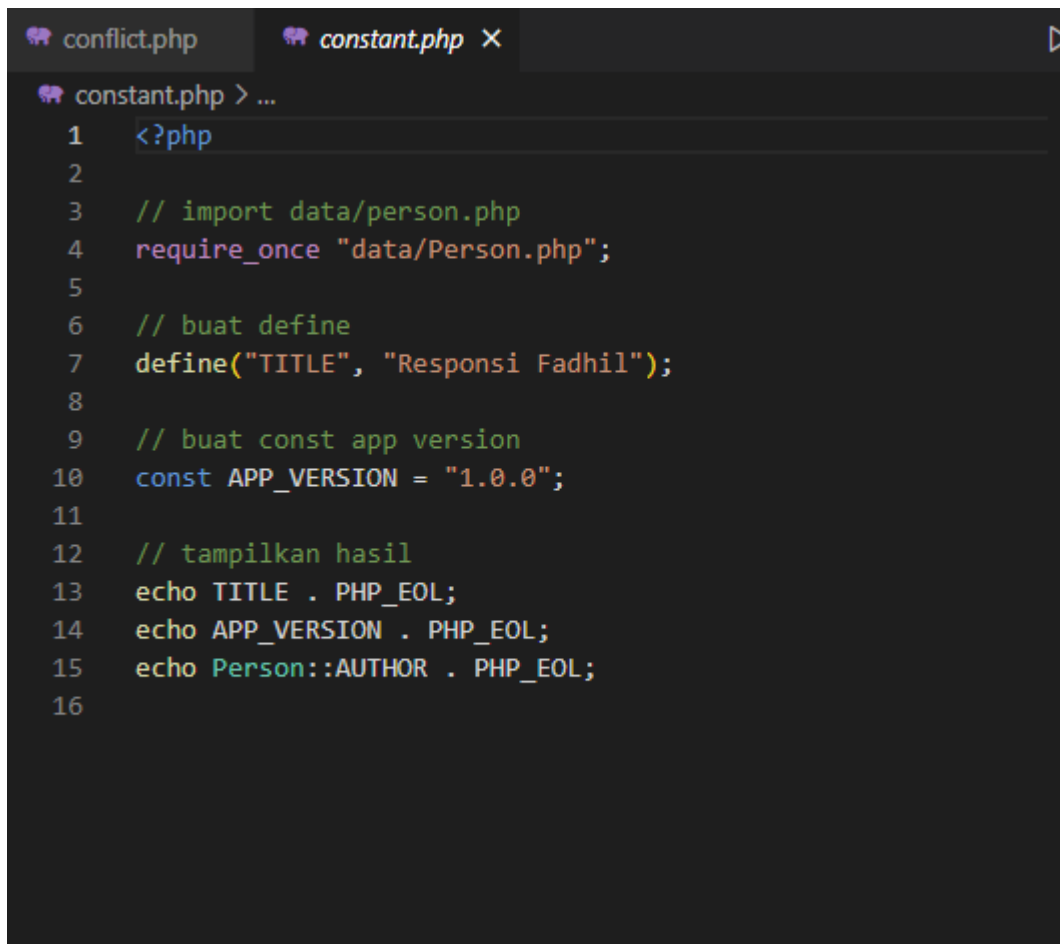
Penjelasan

Kelas Shape: Kelas ini memiliki metode getCorner yang mengembalikan nilai -1. Ini adalah metode yang kemungkinan akan dioverride oleh kelas turunannya.

Kelas Rectangle (Turunan dari Shape): Kelas ini merupakan turunan dari kelas Shape. Mewarisi sifat dan metode dari kelas Shape.

Mendefinisikan ulang metode getCorner dengan mengembalikan nilai 4. Ini adalah contoh penggunaan polimorfisme, di mana kelas turunan dapat memberikan implementasi yang berbeda untuk metode yang sudah ada di kelas dasar.

- constant.php



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

Penjelasan

Gambar di atas menunjukkan beberapa konsep dalam pemrograman PHP, seperti penggunaan `require_once` untuk mengimpor kelas dari file eksternal, penggunaan `define` untuk membuat konstanta, dan penggunaan `const` untuk membuat konstanta kelas.

Menggunakan `require_once` untuk mengimpor kelas `Person` dari file eksternal `"data/Person.php"` ke dalam skrip saat ini.

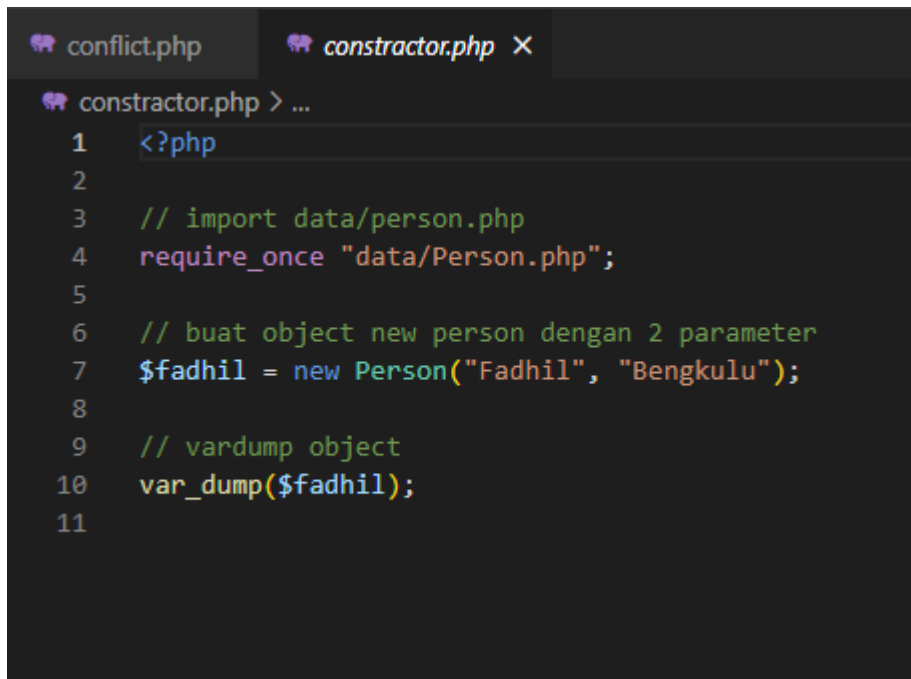
Membuat definisi konstan dengan `define` untuk variabel `TITLE` dengan nilai `"Responsi Fadhil"`.

Membuat konstanta kelas dengan `const` untuk variabel `APP_VERSION` dengan nilai `"1.0.0"`.

Menampilkan nilai dari konstanta `TITLE` dan `APP_VERSION` menggunakan perintah `echo`.

Menampilkan nilai konstanta kelas `AUTHOR` dari kelas `Person` menggunakan notasi `::`.

- 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 $fadhil = new Person("Fadhil", "Bengkulu");
8
9 // vardump object
10 var_dump($fadhil);
11
```

Penjelasan

Menggunakan `require_once` untuk mengimpor kelas `Person` dari file eksternal `"data/Person.php"` ke dalam skrip saat ini.

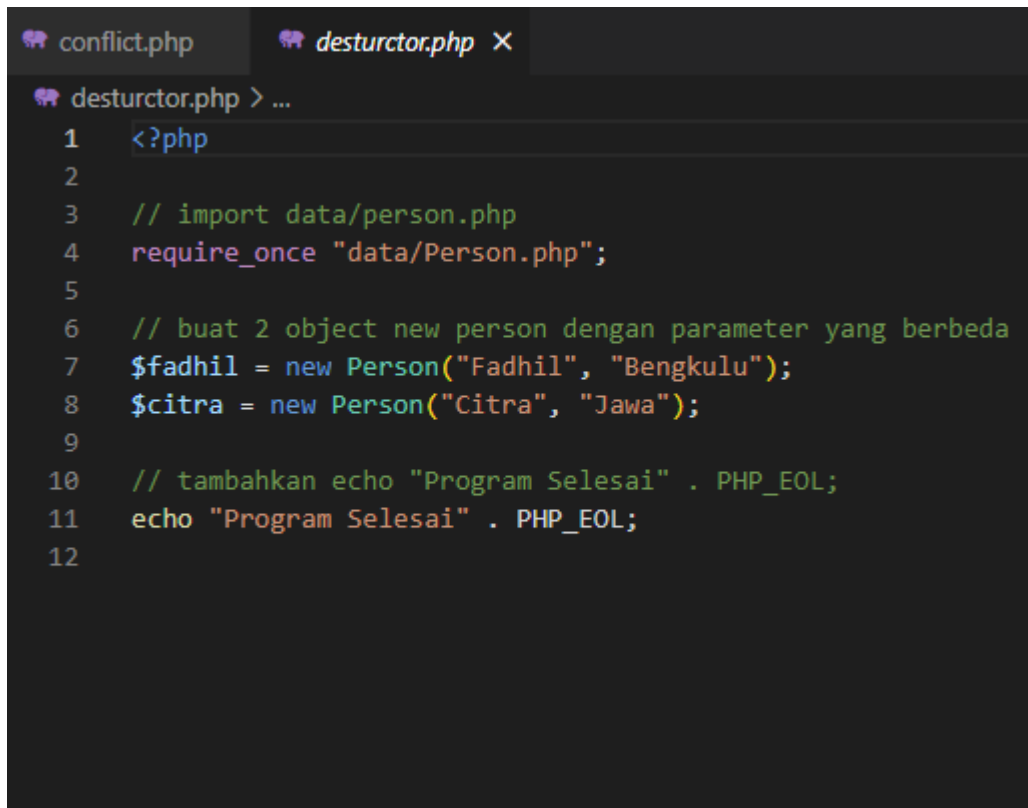
Membuat objek baru dari kelas `Person` dengan nama `$fadhil`.

Menggunakan konstruktor kelas `Person` yang membutuhkan dua parameter, yaitu nama dan alamat, untuk menginisialisasi properti objek.

Menggunakan fungsi `var_dump` untuk menampilkan informasi lengkap tentang objek `$fadhil`.

Fungsi ini berguna untuk debugging dan memberikan detail tentang tipe dan nilai dari properti objek.

- `Destructor.php`



```
conflict.php  destructor.php X
destructor.php > ...
1  <?php
2
3  // import data/person.php
4  require_once "data/Person.php";
5
6  // buat 2 object new person dengan parameter yang berbeda
7  $fadhil = new Person("Fadhil", "Bengkulu");
8  $citra = new Person("Citra", "Jawa");
9
10 // tambahkan echo "Program Selesai" . PHP_EOL;
11 echo "Program Selesai" . PHP_EOL;
12
```

Penjelasan

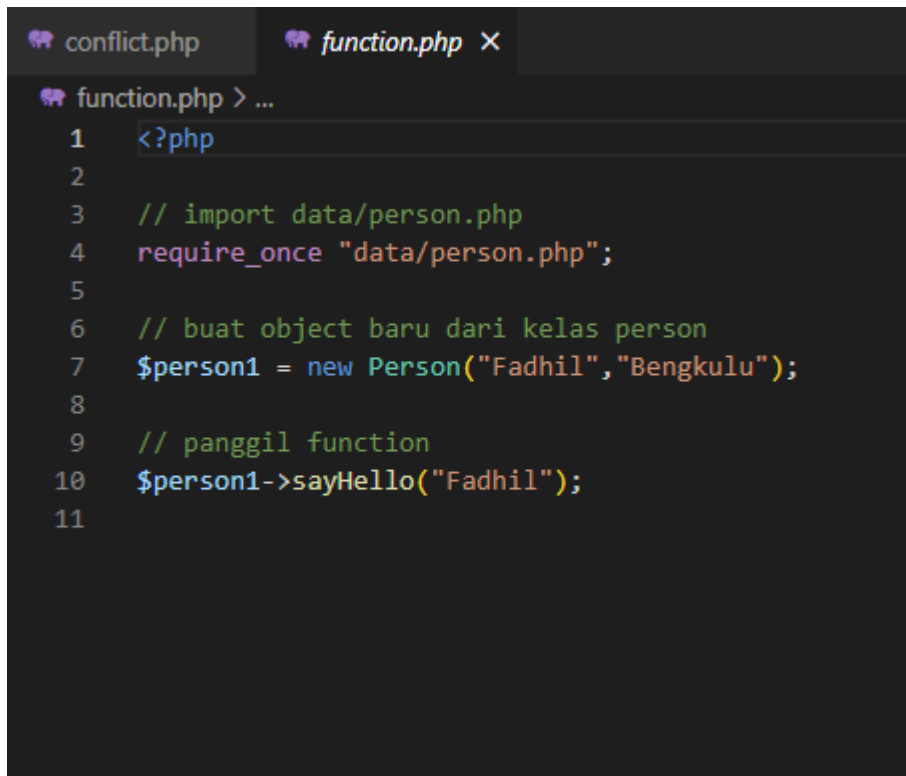
Menggunakan `require_once` untuk mengimpor kelas `Person` dari file eksternal `"data/Person.php"` ke dalam skrip saat ini.

Membuat dua objek baru dari kelas `Person` dengan nama `$fadhil` dan `$citra`.

Menggunakan konstruktor kelas `Person` yang memerlukan dua parameter, yaitu nama dan alamat, untuk menginisialisasi properti objek.

Menampilkan pesan `"Program Selesai"` ke output menggunakan perintah `echo`.

- `Function.php`



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

Penjelasan

Menggunakan `require_once` untuk mengimpor kelas `Person` dari file eksternal `"data/person.php"` ke dalam skrip saat ini.

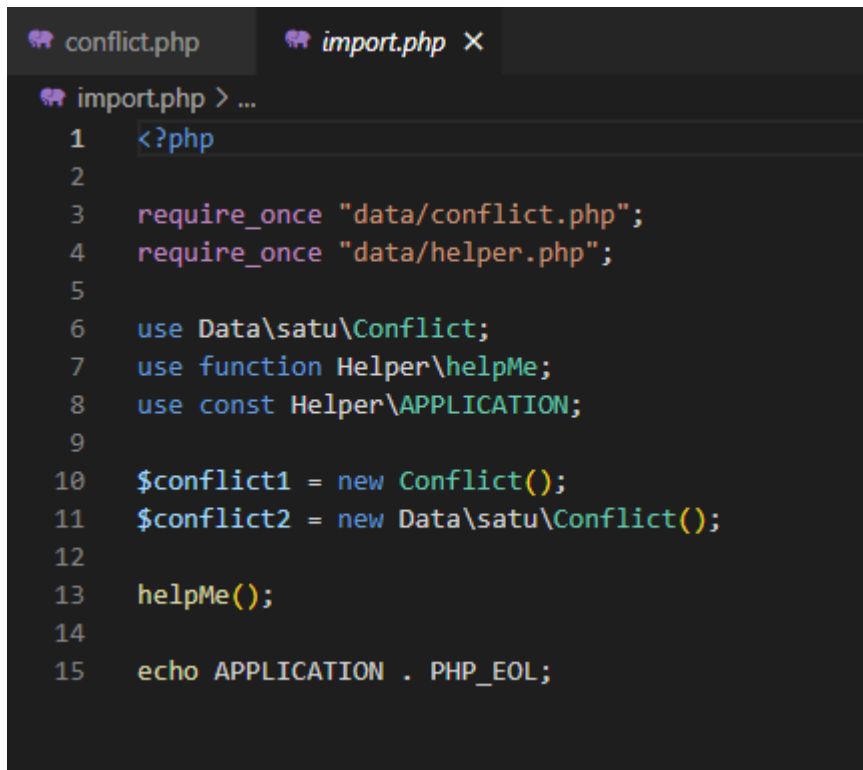
Membuat objek baru dari kelas `Person` dengan nama `$person1`.

Menggunakan konstruktor kelas `Person` yang memerlukan dua parameter, yaitu nama dan alamat, untuk menginisialisasi properti objek.

Memanggil metode `sayHello` dari objek `$person1`.

Metode `sayHello` mencetak pesan sapaan dengan memanfaatkan nilai yang telah diinisialisasi pada pembuatan objek.

- `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\APPLICATION;
9
10 $conflict1 = new Conflict();
11 $conflict2 = new Data\satu\Conflict();
12
13 helpMe();
14
15 echo APPLICATION . PHP_EOL;
```

Penjelasan

Menggunakan `require_once` untuk mengimpor file eksternal "data/conflict.php" dan "data/helper.php" ke dalam skrip saat ini.

Menggunakan `use` untuk mengimpor namespace `Data\satu` dan kelas `Conflict` dari namespace tersebut.

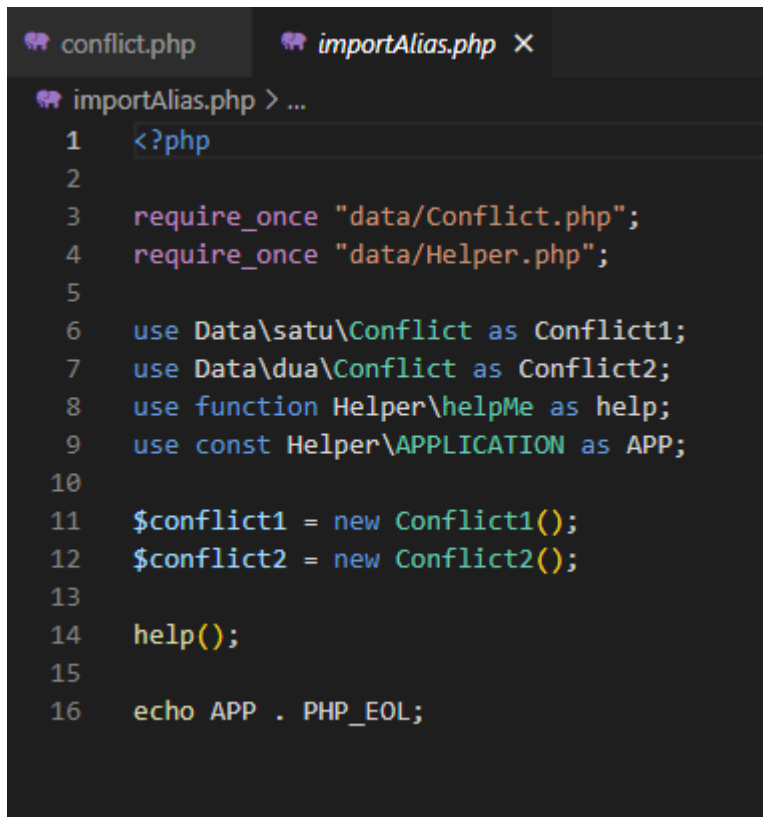
Menggunakan `use` untuk mengimpor fungsi `helpMe` dan konstanta `APPLICATION` dari namespace `Helper`.

Membuat dua objek dari kelas yang bernama `Conflict`, yang satu berada di root namespace, dan yang lainnya berada di namespace `Data\satu`.

Memanggil fungsi `helpMe` yang diimpor dari namespace `Helper`.

Menampilkan nilai konstanta `APPLICATION` yang diimpor dari namespace `Helper`. Melalui kode ini, penggunaan namespace memungkinkan pengelompokkan kode yang serupa, dan `use` digunakan untuk mempermudah penggunaan fungsi dan konstanta dari namespace lain.

- `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\APPLICATION as APP;
10
11 $conflict1 = new Conflict1();
12 $conflict2 = new Conflict2();
13
14 help();
15
16 echo APP . PHP_EOL;
```

Penjelasan

Menggunakan use untuk mengimpor kelas Conflict dari namespace Data\satu dan Data\dua dengan memberikan alias Conflict1 dan Conflict2 masing-masing.

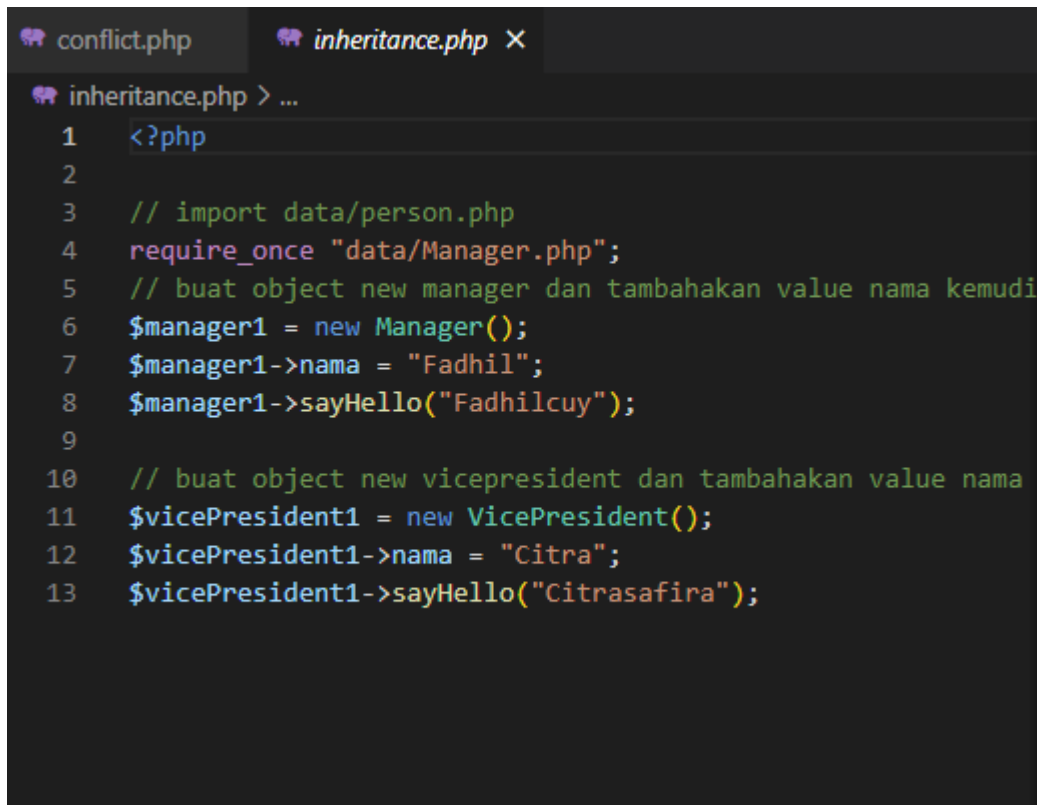
Menggunakan use untuk mengimpor fungsi helpMe dan konstanta APPLICATION dari namespace Helper dengan memberikan alias help dan APP masing-masing.

Membuat dua objek dari kelas yang bernama Conflict, masing-masing dari namespace Data\satu dan Data\dua, dengan menggunakan alias.

Memanggil fungsi help yang merupakan alias untuk fungsi helpMe dari namespace Helper.

Menampilkan nilai konstanta APP yang merupakan alias untuk konstanta APPLICATION dari namespace Helper

- inheritance.php



```
1 <?php
2
3 // import data/person.php
4 require_once "data/Manager.php";
5 // buat object new manager dan tambahkan value nama kemudi
6 $manager1 = new Manager();
7 $manager1->nama = "Fadhil";
8 $manager1->sayHello("Fadhilcuy");
9
10 // buat object new vicepresident dan tambahkan value nama
11 $vicePresident1 = new VicePresident();
12 $vicePresident1->nama = "Citra";
13 $vicePresident1->sayHello("Citrasafira");
```

Penjelasan

Menggunakan `require_once` untuk mengimpor kelas `Manager` dan `VicePresident` dari file eksternal `"data/Manager.php"` ke dalam skrip saat ini.

Membuat objek baru dari kelas `Manager` dengan nama `$manager1`.

Mengisi nilai properti `nama` pada objek.

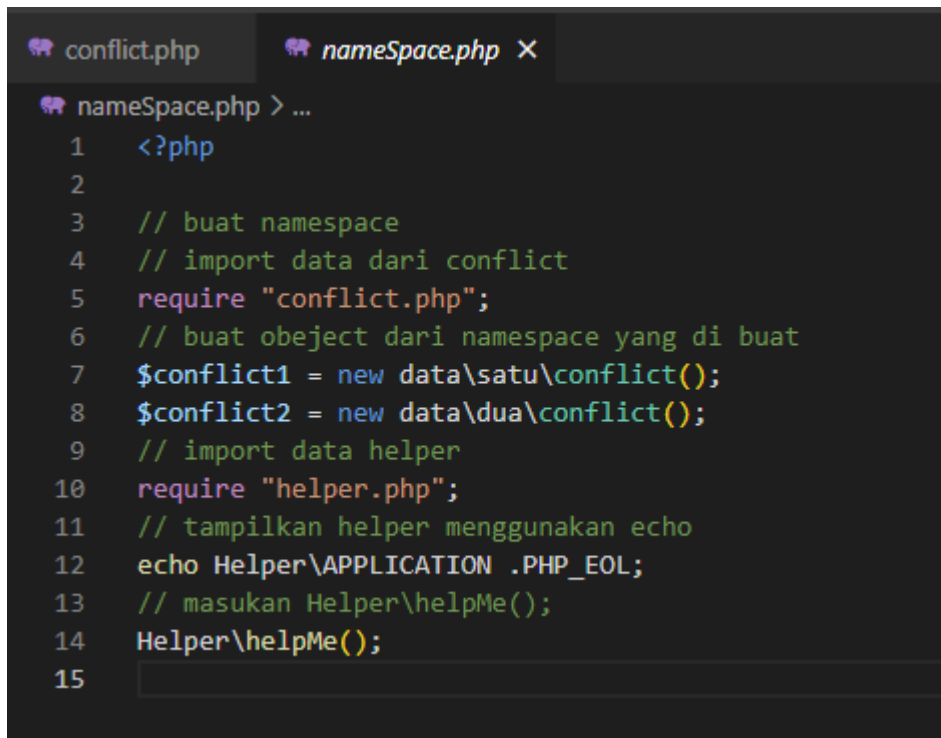
Memanggil metode `sayHello` dari objek `Manager` untuk menampilkan pesan sapaan.

Membuat objek baru dari kelas `VicePresident` dengan nama `$vicePresident1`.

Mengisi nilai properti `nama` pada objek.

Memanggil metode `sayHello` dari objek `VicePresident` untuk menampilkan pesan sapaan.

- `namespace.php`



```
nameSpace.php > ...
1  <?php
2
3  // buat namespace
4  // import data dari conflict
5  require "conflict.php";
6  // buat object dari namespace yang di buat
7  $conflict1 = new data\satu\conflict();
8  $conflict2 = new data\dua\conflict();
9  // import data helper
10 require "helper.php";
11 // tampilkan helper menggunakan echo
12 echo Helper\APPLICATION .PHP_EOL;
13 // masukan Helper\helpMe();
14 Helper\helpMe();
15
```

Penjelasan

Menggunakan require untuk mengimpor file eksternal "conflict.php", yang berisi definisi namespace dan kelas conflict.

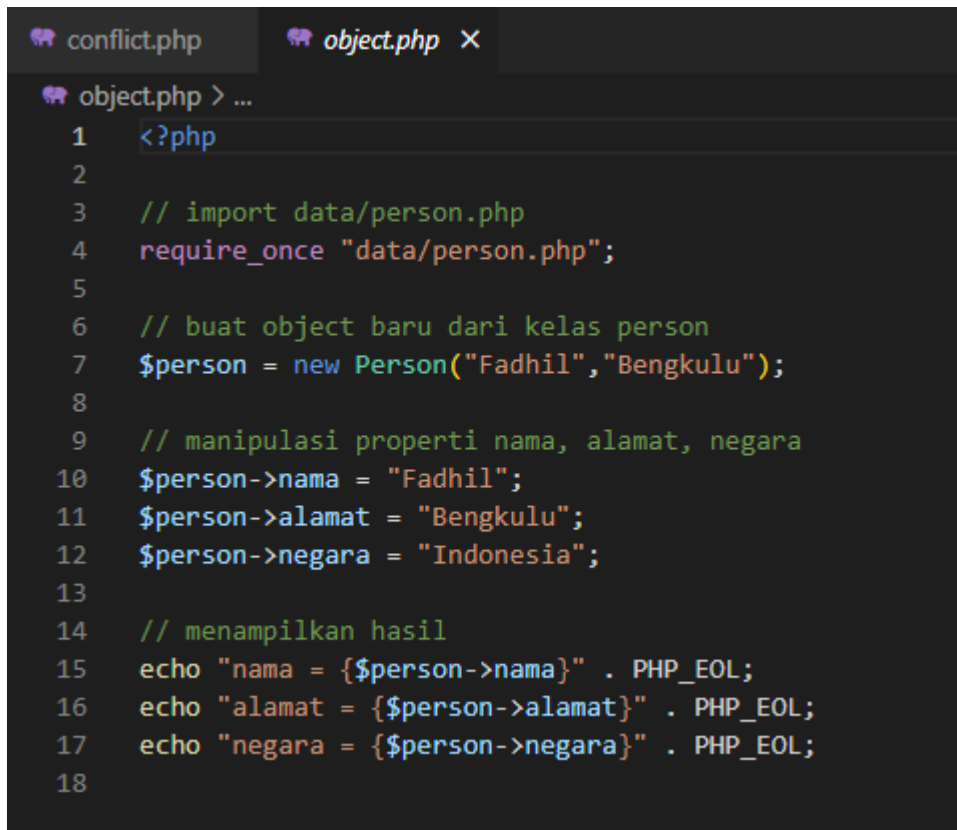
Membuat dua objek dari kelas conflict yang berbeda namespace, yaitu data\satu dan data\dua.

Menggunakan require untuk mengimpor file eksternal "helper.php", yang berisi definisi konstanta dan fungsi dalam namespace Helper.

Menampilkan nilai konstanta APPLICATION dari namespace Helper menggunakan echo.

Memanggil fungsi helpMe() dari namespace Helper.

- object.php



```
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("Fadhil","Bengkulu");
8
9  // manipulasi properti nama, alamat, negara
10 $person->nama = "Fadhil";
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
```

Penjelasan

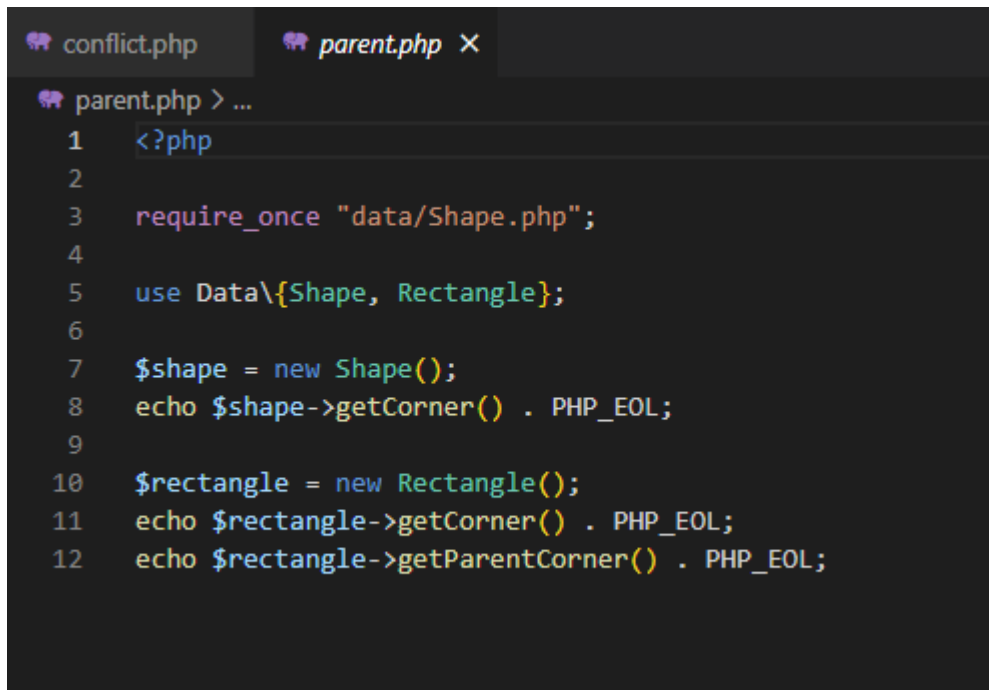
Menggunakan `require_once` untuk mengimpor kelas `Person` dari file eksternal "data/person.php" ke dalam skrip saat ini.

Membuat objek baru dari kelas `Person` dengan nama `$person`.

Menggunakan konstruktor kelas `Person` yang memerlukan dua parameter, yaitu nama dan alamat, untuk menginisialisasi properti objek.

Menggunakan perintah `echo` untuk menampilkan nilai properti yang telah dimanipulasi pada objek `Person`.

- `parent.php`



```
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;
```

Penjelasan

Menggunakan use untuk mengimpor namespace Data dan kelas-kelas Shape dan Rectangle dari namespace tersebut.

Membuat objek \$shape dari kelas Shape.

Memanggil metode getCorner dari objek \$shape untuk menampilkan hasil.

Membuat objek \$rectangle dari kelas Rectangle.

Memanggil metode getCorner dari objek \$rectangle untuk menampilkan hasil.

Memanggil metode getParentCorner dari objek \$rectangle, yang mengandalkan pewarisan dari kelas Shape.

Metode ini menggunakan parent::getCorner() untuk mendapatkan nilai dari metode getCorner di kelas Shape.

- polymorphism.php

Penjelasan

Menggunakan `require_once` untuk mengimpor kelas-kelas `Programmer`, `BackendProgrammer`, `FrontendProgrammer`, dan `Company` dari file eksternal `"data/programmer.php"` ke dalam skrip saat ini.

Membuat objek `$company` dari kelas `Company`.

Mengisi properti `$programmer` pada objek `$company` dengan objek dari kelas `Programmer`.

Menggunakan `var_dump` untuk menampilkan informasi struktur objek.

Mengisi properti `$programmer` pada objek `$company` dengan objek dari kelas `BackendProgrammer`.

Kembali menggunakan `var_dump` untuk menampilkan informasi struktur objek.

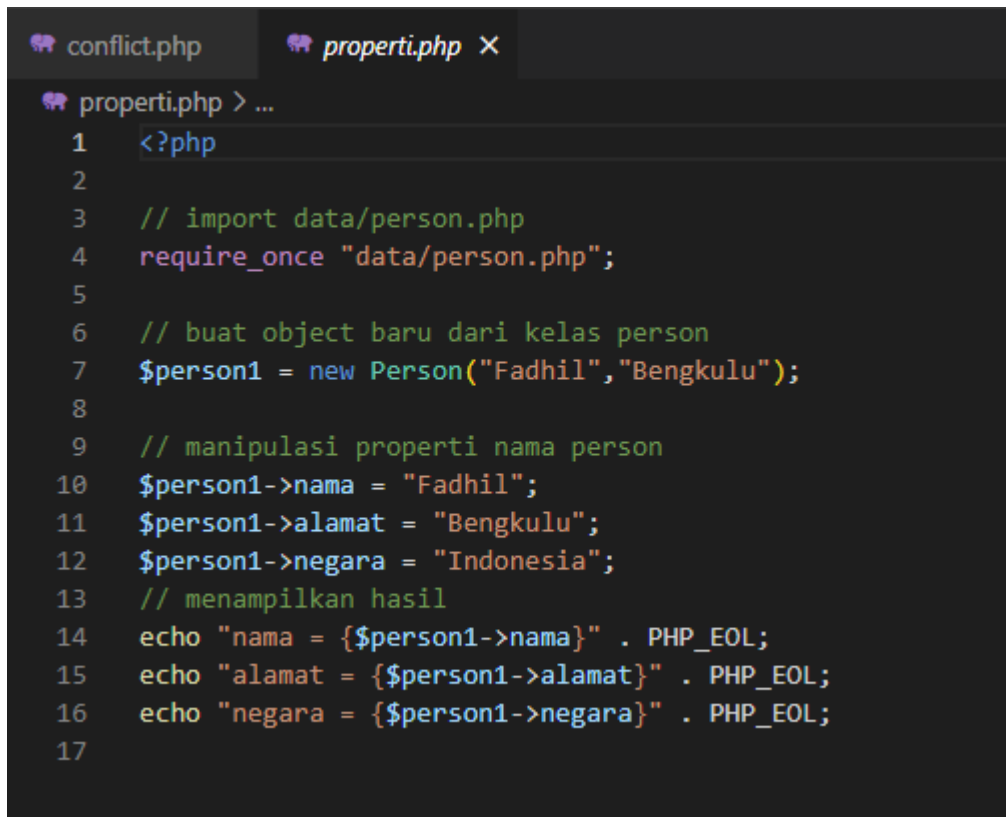
Mengisi properti `$programmer` pada objek `$company` dengan objek dari kelas `FrontendProgrammer`.

Sekali lagi menggunakan `var_dump` untuk menampilkan informasi struktur objek.

Memanggil fungsi `sayHelloProgrammer` dengan berbagai objek yang berasal dari kelas `Programmer`, `BackendProgrammer`, dan `FrontendProgrammer`.

Fungsi tersebut menampilkan pesan sapaan sesuai dengan jenis programmer yang diberikan sebagai parameter.

- `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("Fadhil","Bengkulu");
8
9 // manipulasi properti nama person
10 $person1->nama = "Fadhil";
11 $person1->alamat = "Bengkulu";
12 $person1->negara = "Indonesia";
13 // menampilkan hasil
14 echo "nama = {$person1->nama}" . PHP_EOL;
15 echo "alamat = {$person1->alamat}" . PHP_EOL;
16 echo "negara = {$person1->negara}" . PHP_EOL;
17
```

Penjelasan

Menggunakan `require_once` untuk mengimpor kelas `Person` dari file eksternal `"data/person.php"` ke dalam skrip saat ini.

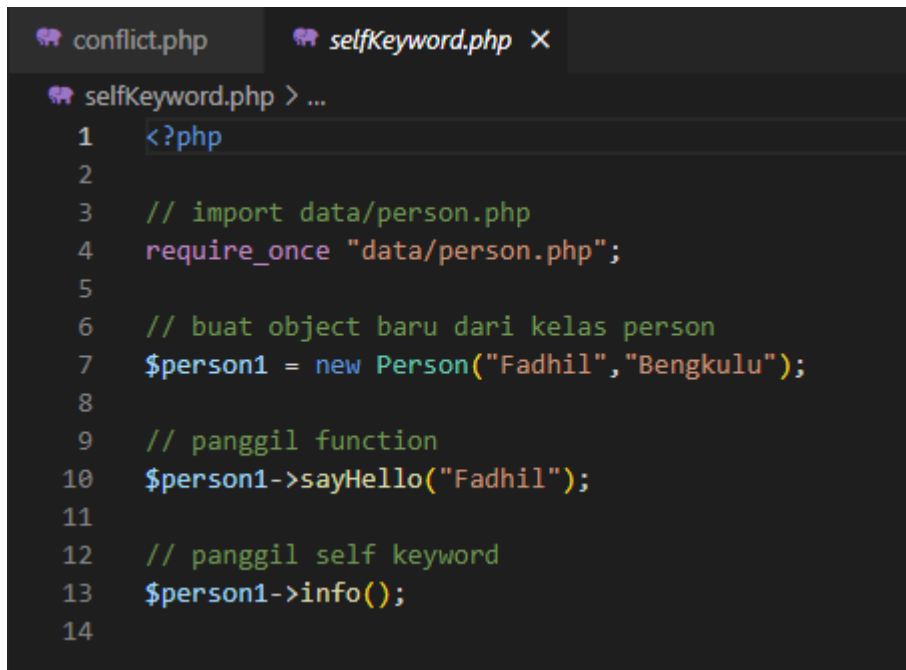
Membuat objek baru dari kelas `Person` dengan nama `$person1`.

Menggunakan konstruktor kelas `Person` yang memerlukan dua parameter, yaitu nama dan alamat, untuk menginisialisasi properti objek.

Mengakses dan memanipulasi properti objek `Person` dengan memberikan nilai baru pada properti nama, alamat, dan negara.

Menggunakan perintah `echo` untuk menampilkan nilai properti yang telah dimanipulasi pada objek `Person`.

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

Penjelasan

Menggunakan require_once untuk mengimpor kelas Person dari file eksternal "data/person.php" ke dalam skrip saat ini.

Membuat objek baru dari kelas Person dengan nama \$person1.

Menggunakan konstruktor kelas Person yang memerlukan dua parameter, yaitu nama dan alamat, untuk menginisialisasi properti objek.

Memanggil metode sayHello dari objek \$person1.

Metode ini mencetak pesan sapaan dengan memanfaatkan nilai yang telah diinisialisasi pada pembuatan objek.

Memanggil metode info dari objek \$person1 yang menggunakan self keyword.

Metode ini mencetak informasi konstanta AUTHOR yang terdapat di dalam kelas Person menggunakan self keyword.

- thisKeyword.php

```
conflict.php  thisKeyword.php X
thisKeyword.php > ...
1  <?php
2
3  // import data/person.php
4  require_once "data/person.php";
5
6  // buat object dari kelas person
7  $fadhil = new Person("Fadhil", "Bengkulu");
8
9  // tambahkan value nama di object
10 $fadhil->nama = "Fadhil";
11
12 // panggil function sayHelloNull dengan parameter
13 $fadhil->sayHelloNull("Citra Safira");
14
15 // buat object dari kelas person
16 $citra = new Person("Citra", "Jawa");
17
18 // tambahkan value nama di object
19 $citra->nama = "Cookie Pratama";
20
21 // panggil function sayHelloNull dengan parameter null
22 $citra->sayHelloNull(null);
23
```

Penjelasan

Menggunakan `require_once` untuk mengimpor kelas `Person` dari file eksternal `"data/person.php"` ke dalam skrip saat ini.

Membuat objek baru dari kelas `Person` dengan nama `$fadhil`.

Menggunakan konstruktor kelas `Person` yang memerlukan dua parameter, yaitu nama dan alamat, untuk menginisialisasi properti objek.

Mengakses dan memanipulasi properti nama objek `$fadhil` dengan memberikan nilai baru.

Memanggil metode `sayHelloNull` dari objek `$fadhil` dengan memberikan parameter `"Citra Safira"`.

Metode ini mencetak pesan sapaan dengan parameter yang diberikan atau default jika parameter `null`.

Membuat objek baru dari kelas `Person` dengan nama `$citra`.

Menggunakan konstruktor kelas `Person` untuk menginisialisasi properti objek.

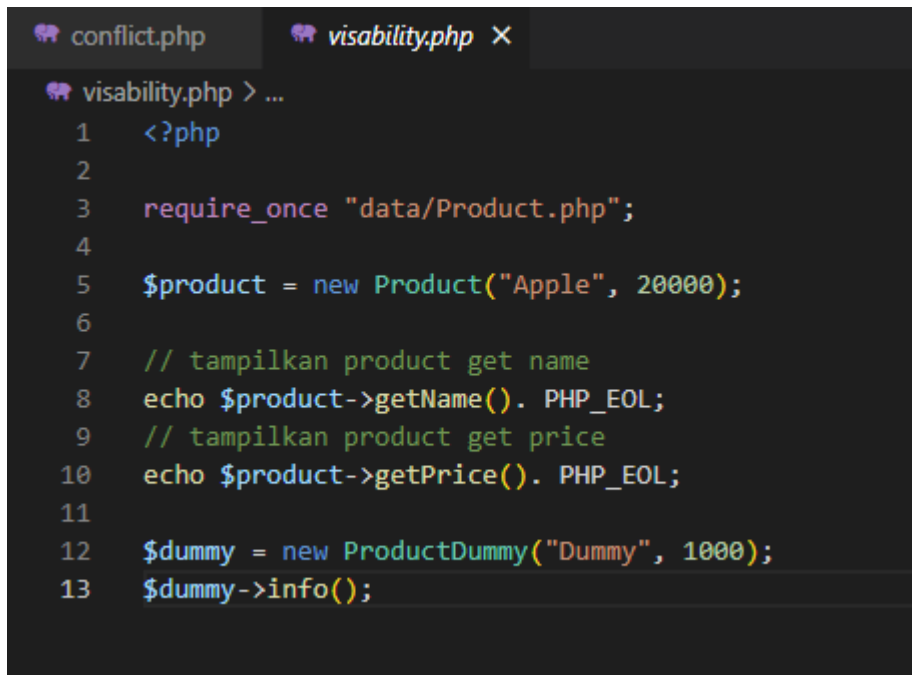
Memanipulasi properti nama objek `$citra`.

Memanggil metode `sayHelloNull` dari objek `$citra` dengan memberikan parameter `null`.

Metode ini mencetak pesan sapaan dengan menggunakan nilai default jika parameter `null`.

Melalui kode ini, konsep OOP terlihat dalam pembuatan objek, manipulasi properti, dan pemanggilan metode pada objek. Pemakaian metode sayHelloNull dengan parameter dan parameter null juga menunjukkan fleksibilitas dalam pemrograman dengan PHP.

- visibility.php



```
visibility.php > ...
1  <?php
2
3  require_once "data/Product.php";
4
5  $product = new Product("Apple", 20000);
6
7  // tampilkan product get name
8  echo $product->getName(). PHP_EOL;
9  // tampilkan product get price
10 echo $product->getPrice(). PHP_EOL;
11
12 $dummy = new ProductDummy("Dummy", 1000);
13 $dummy->info();
```

Penjelasan

Menggunakan require_once untuk mengimpor kelas Product dan ProductDummy dari file eksternal "data/Product.php" ke dalam skrip saat ini.

Membuat objek baru dari kelas Product dengan nama \$product.

Menggunakan konstruktor kelas Product yang memerlukan dua parameter, yaitu nama produk dan harga, untuk menginisialisasi properti objek.

Mengakses metode getName dan getPrice dari objek \$product untuk mendapatkan nama dan harga produk.

Menampilkan hasil menggunakan perintah echo.

Membuat objek baru dari kelas ProductDummy dengan nama \$dummy.

Menggunakan konstruktor kelas ProductDummy untuk menginisialisasi properti objek.

Memanggil metode info dari objek \$dummy, yang mencetak informasi nama dan harga produk.

Melalui kode ini, konsep OOP terlihat dalam pembuatan objek, penggunaan konstruktor, dan pemanggilan metode pada objek. Metode info pada kelas ProductDummy menunjukkan bagaimana kelas turunan dapat menambahkan fungsionalitas atau memodifikasi perilaku dari kelas induk (inheritance).