Belajar Santai OOP PHP

Memahami Konsep OOP dengan Cara yang Mudah

Muhamad Surya Iksanudin

Belajar Santai OOP PHP

Memahami Konsep OOP dengan Cara yang Mudah

Muhamad Surya Iksanudin

Contents

Pengertian Object Oriented Programming

I. Apa itu OOP

Pemrograman berbasis *object* (OOP)[1] adalah sebuah paradigma pemrograman yang berorientasikan kepada *object*. Semua data dan fungsi dalam paradigma ini dibungkus dengan *class-class* atau *object-object*.

Dalam pemrograman berbasis objek, kita diminta untuk memahami sebuah masalah dan memodelkan masalah tersebut menjadi sebuah *class* dan kemudian *class* akan diinisiasi menjadi sebuah *object* pada saat *runtime*.

Setiap *class/object* dalam pemrograman berbasis *object* dapat saling berinteraksi dan berkomunikasi satu sama lain untuk mendukung sebuah solusi dari suatu masalah.

II. Kelebihan OOP dibandingkan *procedural*

Kelebihan OOP dibandingkan dengan *procedural* antara lain:

- Lebih terstruktur dan mudah dibaca.
- *Class-Class* dapat digunakan kembali pada *project* yang lain (*reuse*).

[1] https://id.wikipedia.org/wiki/Pemrograman_berorientasi_objek

- Pemetaan masalah jadi lebih mudah sehingga mudah untuk membuat solusinya.
- Pembatasan akses terhadap suatu fungsi dapat meningkatkan keamaan *code*.
- Interaksi antara *code* lebih terasa.

Satu untuk semua

Karena pemrograman berbasis objek adalah sebuah konsep.

Jika Anda menguasainya, Anda dapat menguasai bahasa pemrograman lain yang mendukung OOP dengan mudah.

III. Bahasa yang mendukung konsep Pemrograman berbasis objek

Bila Anda menguasai OOP, maka Anda akan lebih mudah mempelajari bahasa pemrograman yang mendukung OOP. Adapun bahasa pemrograman yang mendukung OOP antara lain:

- PHP
- Java
- C++
- Python

- .Net
- Ruby
- Go

Dan masih banyak lagi.

Class dan *Object*

I. Pengertian *Class*

Secara gampang, *class* adalah sebuah model/cetakan sedangkan *object* adalah realisasinya. Dalam OOP, *Class* memiliki `property` dan `method`. `Property` adalah sesuatu yang dimiliki oleh *class*, sedangkan `method` adalah apa-apa saja yang bisa dilakukan oleh *class*.

Bila diibaratkan dengan **Mobil**, maka `property` adalah roda, kursi, pintu, dan lain sebagainya. Sedangkan `method` adalah maju, mundur, berbelok, mengerem dan lain sebagainya.

II. Contoh *Class*

Setelah kita memahami pengertian dari *class*, tidak lengkap rasanya kalau tidak ada contoh penggunaannya. Contoh dibawah ini akan memberikan gambaran lebih dalam tentang *class*.

```php
<?php

//Class
class Mobil
{
    //Property
    private $jumlahRoda;

    //Property
    private $jumlahKursi;

```

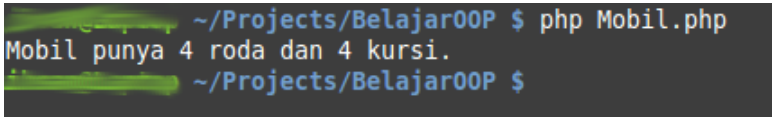
```
12      //Method
13      public function setJumlahRoda($jumlahRoda)
14      {
15          $this->jumlahRoda = $jumlahRoda;
16      }
17
18      //Method
19      public function setJumlahKursi($jumlahKursi)
20      {
21          $this->jumlahKursi = $jumlahKursi;
22      }
23
24      //Method
25      public function cetak()
26      {
27          echo 'Mobil punya '.$this->jumlahRoda.' roda da\
28  n '.$this->jumlahKursi.' kursi.';
29      }
30  }
31
32  $sedan = new Mobil();//Object
33  $sedan->setJumlahRoda(4);
34  $sedan->setJumlahKursi(4);
35  $sedan->cetak();
36  echo PHP_EOL;
```

Pada contoh diatas, *class* `Mobil` adalah sebuah *prototype/model* sedangkan `$sedan` adalah realisasinya.

Sementara `$jumlahRoda` dan `$jumlahKursi` adalah `property`.

Sedangkan `setJumlahRoda($jumlahRoda)` sampai pada `cetak()` dinamakan `method` (akan dijelaskan secara khusus pada bab tersendiri).

Bila program diatas dijalankan, maka outpunya akan tampak sebagai berikut:



Mobil

Perlu Anda ketahui, karena *class* hanya sebuah *prototype* atau *model*, maka *class* dapat diinstansiasi menjadi banyak *object*:

```
1  $suv = new Mobil();
2  $suv->setJumlahRoda(4);
3  $suv->setJumlahKursi(6);
4  $suv->cetak();
5  echo PHP_EOL;
```

Karena *class* bersifat *prototype*, maka *class* tidak akan di-*mapping* kedalam memori (RAM) dan *object*-lah yang akan di-*mapping* kedalam RAM. Karena pada dasarnya, *object* sama saja dengan variabel biasa pada PHP.

Kata Kunci

Class adalah cetakan, *object* adalah barang jadinya/realisasinya.

Proses membuat *object* disebut instansiasi

Keyword `$this` dan `self`

I. Pengantar

Sebenarnya saya agak ragu untuk membahas tentang *keyword* `$this` dan `self` sekarang, namun karena sudah dipakai pada pembahasan sebelumnya dan pastinya akan lebih sering dipakai lagi kedepannya, maka saya akan mencoba membahasnya pada pembahasan sekarang.

Saya harap, Anda tidak bingung tentang konsep kedua *keyword* ini dalam pemrograman OOP. Dan semoga apa yang saya jelaskan nantinya dapat memberikan gambaran tentang bagaimana cara kedua *keyword* ini bekerja.

II. Keyword `$this`

Pada bab-bab sebelumnya kita telah menggunakan keyword `$this` untuk mengakses sebuah *property* seperti pada contoh dibawah ini.

```
1  <?php
2
3  class Mobil
4  {
5      private $jumlahRoda;
6
7      public function setJumlahRoda($jumlahRoda)
8      {
9              $this->jumlahRoda = $jumlahRoda;
10     }
11 }
```

Dalam pemrograman berbasis *object*, keyword `$this` pasti ada, walaupun cara penulisan dan mungkin namanya berbeda.

Keyword `$this` dalam OOP adalah sebuah variabel yang merujuk pada *object* yang diinstansiasi.

Maksudnya *keyword* `$this` ini nantinya akan diganti dengan variabel apapun tergantung dari variabel *object* yang diinstansiasi. Perhatikan contoh dibawah ini.

```
1  <?php
2
3  class Mobil
4  {
5      private $jumlahRoda;
6
7      public function setJumlahRoda($jumlahRoda)
8      {
9              $this->jumlahRoda = $jumlahRoda;
10     }
11 }
12
13 $mobil = new Mobil();
14 $mobil->setJumlahRoda(4);
```

Pada contoh diatas, kita membuat object class `Mobil` dengan nama `$mobil`. Maka saat itu `$this` akan digantikan dengan variabel `$mobil`.

Dan bila kita membuat *object* lainnya misalnya `$mobil2` maka `$this` akan digantikan dengan `$mobil2`.

Dapat disimpulkan bahwa *keyword* `$this`, digunakan untuk merujuk pada *object* yang belum diketahui dan digunakan untuk mempermudah kita dalam menuliskan *code*.

Perlu Anda ketahui bahwa antara `$mobil` dan `$mobil2` itu adalah dua *object* yang berbeda sehingga ketika memanggil `$mobil->setJumlahRoda(4)`

dan `$mobil2->setJumlahRoda(7)` maka nilai `$jumlahRoda` pada `$mobil` tidak akan ditimpa oleh nilai `$jumlahRoda` pada `$mobil2`.

Untuk lebih jelas, perhatikan contoh dibawah ini:

```php
<?php

class Mobil
{
    private $jumlahRoda;

    public function setJumlahRoda($jumlahRoda)
    {
            $this->jumlahRoda = $jumlahRoda;
    }

    public function getJumlahRoda()
    {
        return $this->jumlahRoda;
    }
}

$mobil = new Mobil();
$mobil->setJumlahRoda(4);
$mobil2 = new Mobil();
$mobil2->setJumlahRoda(7);

echo $mobil->getJumlahRoda().'<br/>';//Output: 4
echo $mobil2->getJumlahRoda();//Output: 7
```

Keyword `$this` hanya dapat digunakan pada *internal class* dan tidak dapat dipanggil dari luar *class*. Selain itu, *keyword* `$this` juga tidak bisa ditimpa nilainya (*read-only variable*).

Kata Kunci

`$this` adalah *keyword* yang merujuk pada *object* itu sendiri

`$this` hanya dapat diakses dari *internal class*

`$this` tidak dapat dirubah nilainya (*read-only variable*)

III. *Keyword* `self`

Tidak jauh berbeda dengan *keyword* `$this`, *keyword* `self` pun memiliki karakteristik yang sama. Yang membedakan dengan *keyword* `$this` adalah bahwa *keyword* `self` digunakan hanya untuk memanggil *property* atau *method* yang bersifat *static*.

Contoh yang *property* yang bersifat *static* adalah *constanta*. Sehingga ketika kita manggil *constanta* didalam *class* maka kita memanggilnya dengan `self::NAMA_CONSTANTA`.

Pemahaman lebih dalam tentang sifat *static* pada *class*, akan dibahas pada bab tersendiri.

Namespace, Use dan As

I. Pengantar

Sebelum kita membahas tentang namespace, use dan as, boleh kiranya saya sedikit bercerita.

Saya punya teman yang bekerja di ***Perusahaan A***. Perusahaan tersebut beralamat di ***Kawasan Industri Makmur Sejahter Blok Makanan Kavling 27 No. 17 Kecamatan Tanjung Priuk - Jakarta Utara***.

Karena alamat tersebut susah sekali dihafalkan dan terlalu panjang untuk ditulis, maka perusahaan tersebut ***mengontrak*** kotak pos.

Setelah terjadi *MoU*, maka kantor pos memberikan alamat singkat yaitu ***PO BOX 14000***.

Sehingga sekarang, kalau saya ingin berkirim surat ke perusahaan teman saya, saya cukup menuliskan alamat *PO BOX 14000* maka surat tersebut akan sampai ke perusahaan teman saya tersebut.

II. Namespace

Pada PHP, namespace baru diperkenal pada PHP versi 5.3.X sehingga bagi Anda yang menggunakan PHP versi kurang dari 5.3 tidak dapat menggunakan fitur ini.

Namespace pada PHP sama seperti package pada Java yaitu fungsinya menyatukan *class-class* kedalam sebuah paket. Penggunaan namespace bertujuan agar tidak terjadi pendeklarasian nama *class* yang sama namun dengan fungsi yang berbeda.

Contoh penggunaan namespace dalam kehidupan riil adalah seperti blok pada perumahan. Dalam sebuah kawasan perumahan, pasti ada banyak rumah yang memakai no rumah 1. Misalnya Blok A No. 1, Blok B No. 1, Blok C No. 2 dan seterusnya.

Dapat dibayangkan, jika tanpa adanya blok-blok tersebut, pasti ketika seseorang mengirimkan surat ke alamat misalnya, Perumahan Permai Indah No. 1, surat tersebut bisa saja tidak sampai ke orang yang seharusnya dikarenakan banyak rumah yang memakai nomer rumah 1.

Namun dengan adanya blok, maka kita bisa tahu, kepada siapa surat tersebut harusnya diserahkan. Misal alamatnya jadi, Perumahan Permai Indah Blok A No. 1. Maka kita tahu bahwa surat tersebut adalah milik rumah di Blok A dengan nomer rumah 1.

Blok dalam sebuah perumahan adalah gabungan dari banyak rumah yang disatukan dalam sebuah kawasan. Seperti itulah kira-kira fungsi dari namespace yaitu menyatukan *class-class* kedalam sebuah paket. Dengan namespace kita bisa tahu dengan pasti alamat sebuah *class*.

Pada pengantar diatas, ***Kawasan Industri Makmur Sejahter Blok Makanan Kavling 27 No. 17 Kecamatan Tanjung Priuk - Jakarta Utara*** adalah sebuah namespace dari *class* ***Perusahaan A***.

Lalu bagaimana implementasi namespace pada OOP PHP? Berikut ada cara penggunaan namespace pada PHP:

```php
<?php

//filename: Kendaraan/Mobil/Bmw.php

namespace Kendaraan\Mobil;

class Bmw
{
    const MEREK = 'BMW';
}
```

```php
<?php

//filename: Kendaraan/Mobil/Lamborgini.php

namespace Kendaraan\Mobil;

class Lamborgini
{
    const MEREK = 'Lamborgini';
}
```

```php
<?php

//filename: Kendaraan/Mobil/Toyota.php

namespace Kendaraan\Mobil;

class Toyota
{
    const MEREK = 'Toyota';
}
```

Pada contoh diatas, kita memiliki tiga *class* dengan namespace yang sama yaitu `Kendaraan\Mobil`. *Class* `Bmw`, `Lamborgini` dan `Toyota` disebut *member* dari namespace tersebut.

Agar lebih jelas lagi tentang fungsi dari namespace, berikut adalah tiga *class* dengan nama yang sama namun dalam namespace yang berbeda.

```php
<?php

//filename: Sparepart/Mobil/Bmw.php

namespace Sparepart\Mobil;

class Bmw
{
    const MEREK = 'BMW';
}
```

```php
<?php

//filename: Sparepart/Mobil/Lamborgini.php

namespace Sparepart\Mobil;

class Lamborgini
{
    const MEREK = 'Lamborgini';
}
```

```php
<?php

//filename: Sparepart/Mobil/Toyota.php

namespace Sparepart\Mobil;

class Toyota
{
    const MEREK = 'Toyota';
}
```

Dengan namespace kita dapat mendefinisikan nama *class* yang sama, namun dengan namespace yang berbeda. Bila tanpa menggunakan namespace, jika kita mendefinisikan nama *class* yang sama maka akan terjadi *error* karena dianggap *redeclare class* atau mendefinisikan ulang *class* dengan nama yang sama.

III. Use

Setelah kita memahami tentang konsep namespace maka selanjutnya adalah bagaimana cara memanggil atau menggunakan namespace dalam sebuah program.

Jadi untuk memanggil sebuah namespace dalam program kita, kita harus menggunakan *keyword* use.

Berikut adalah contoh penggunaannya:

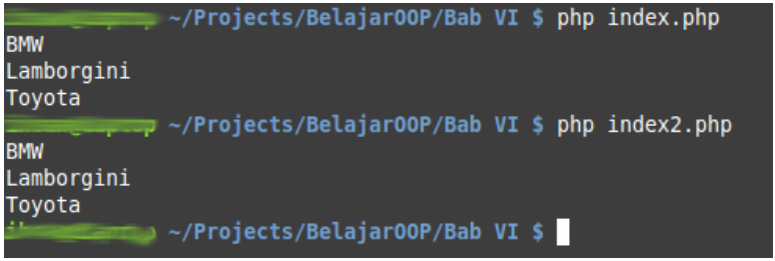
```php
<?php

//filename: index.php

require __DIR__.'/Kendaraan/Mobil/Bmw.php';
require __DIR__.'/Kendaraan/Mobil/Lamborgini.php';
require __DIR__.'/Kendaraan/Mobil/Toyota.php';

use Kendaraan\Mobil\Bmw;
use Kendaraan\Mobil\Lamborgini;
use Kendaraan\Mobil\Toyota;

echo Bmw::MEREK.PHP_EOL;
echo Lamborgini::MEREK.PHP_EOL;
echo Toyota::MEREK.PHP_EOL;
```

Pada *code* diatas, `Kendaraan\Mobil\Bmw`, `Kendaraan\Mobil\Lamborgini` dan `Kendaraan\Mobil\Toyota` merujuk pada *class* `Bmw`, `Lamborgini` dan `Toyota` yang ketiganya memiliki `namespace` yang sama yaitu `Kendaraan\Mobil`.

Bila tanpa menggunakan `use` maka *code* diatas akan menjadi seperti berikut:

```php
<?php

//filename: index2.php

require __DIR__.'/Kendaraan/Mobil/Bmw.php';
require __DIR__.'/Kendaraan/Mobil/Lamborgini.php';
require __DIR__.'/Kendaraan/Mobil/Toyota.php';

echo \Kendaraan\Mobil\Bmw::MEREK.PHP_EOL;
echo \Kendaraan\Mobil\Lamborgini::MEREK.PHP_EOL;
echo \Kendaraan\Mobil\Toyota::MEREK.PHP_EOL;
```

Kedua program diatas, jika dijalankan maka *output*-nya akan sama yaitu sebagai berikut:



Namespace dan Use

IV. As

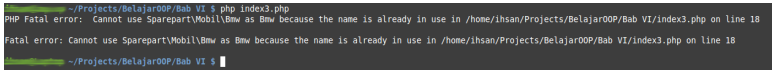
Sebelum saya menjelaskan fungsi dari *keyword* `as`, terlebih dahulu Anda akan saya ajak untuk melakukan percobaan sederhana berikut.

Kita *load* semua class `Bmw`, `Lamborgini` dan `Toyota` baik yang ada pada *namespace* `Kendaraan\Mobil` maupun yang ada pada *namespace* `Sparepart\Mobil` sebagai berikut:

```
1  <?php
2
3  //filename: index3.php
4
5  require __DIR__.'/Kendaraan/Mobil/Bmw.php';
6  require __DIR__.'/Kendaraan/Mobil/Lamborgini.php';
7  require __DIR__.'/Kendaraan/Mobil/Toyota.php';
8
9  require __DIR__.'/Sparepart/Mobil/Bmw.php';
10 require __DIR__.'/Sparepart/Mobil/Lamborgini.php';
11 require __DIR__.'/Sparepart/Mobil/Toyota.php';
12
13
```

```
14  use Kendaraan\Mobil\Bmw;
15  use Kendaraan\Mobil\Lamborgini;
16  use Kendaraan\Mobil\Toyota;
17
18  use Sparepart\Mobil\Bmw;
19  use Sparepart\Mobil\Lamborgini;
20  use Sparepart\Mobil\Toyota;
21
22  echo Bmw::MEREK.PHP_EOL;
23  echo Lamborgini::MEREK.PHP_EOL;
24  echo Toyota::MEREK.PHP_EOL;
25
26  echo Bmw::MEREK.PHP_EOL;
27  echo Lamborgini::MEREK.PHP_EOL;
28  echo Toyota::MEREK.PHP_EOL;
```

Kemudian kita jalankan program tersebut. Apakah yang terjadi? Ternyata yang terjadi adalah *error* sebagai berikut:



Fatal Error

Adakah yang aneh dengan *error* tersebut? Kita *kan* belum menggunakan *keyword* `use`, tapi kenapa dalam pesan *error* muncul tulisan "*Cannot use SparepartMobilBmw* ***as*** *Bmw because the name is already in use*".

Jadi seperti ini, ternyata secara *default*, PHP sebenarnya menggunakan *keyword* `as` ketika menggunakan *keyword* `use`. Atau dengan bahasa lain, ketika kita menggunakan *keyword* `use` secara tidak langsung kita juga menggunakan *keyword* `as`.

Fungsi dari *keyword* `as` adalah memberikan alias kepada *class* ketika dipanggil dalam program. Seperti yang terlihat dalam contoh, secara *default* alias dari suatu *class* adalah nama *class* itu

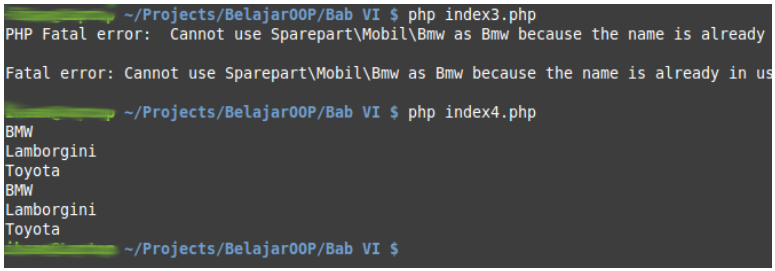
sendiri.

Sehingga bila kita ingin program diatas tidak *error*, maka yang perlu kita lakukan adalah memberikan alias untuk *class* yang sama namanya. Sehingga program diatas akan menjadi seperti berikut:

```php
<?php

//filename: index4.php

require __DIR__.'/Kendaraan/Mobil/Bmw.php';
require __DIR__.'/Kendaraan/Mobil/Lamborgini.php';
require __DIR__.'/Kendaraan/Mobil/Toyota.php';

require __DIR__.'/Sparepart/Mobil/Bmw.php';
require __DIR__.'/Sparepart/Mobil/Lamborgini.php';
require __DIR__.'/Sparepart/Mobil/Toyota.php';


use Kendaraan\Mobil\Bmw as KendaraanBmw;
use Kendaraan\Mobil\Lamborgini as KendaraanLamborgini;
use Kendaraan\Mobil\Toyota as KendaraanToyota;

use Sparepart\Mobil\Bmw as SparepartBmw;
use Sparepart\Mobil\Lamborgini as SparepartLamborgini;
use Sparepart\Mobil\Toyota as SparepartToyota;

echo KendaraanBmw::MEREK.PHP_EOL;
echo KendaraanLamborgini::MEREK.PHP_EOL;
echo KendaraanToyota::MEREK.PHP_EOL;

echo SparepartBmw::MEREK.PHP_EOL;
echo SparepartLamborgini::MEREK.PHP_EOL;
echo SparepartToyota::MEREK.PHP_EOL;
```

Sekarang, bila program diatas dijalankan maka hasilnya akan seperti

berikut:



Alias

Bagaimana? Apakah sekarang sudah mengerti fungsi dari *keyword* `as`?

Pada pengantar diatas, `as` diibaratkan dengan ***mengontrak*** dan ***PO BOX 14000*** adalah nama dari aliasnya.

Sebenarnya pada contoh diatas, kita bisa saja hanya memberikan alias hanya pada tiga *class* saja sudah cukup, namun agar tidak timbul kebingungan, akan lebih baik jika kita memberikan alias untuk semua *class* sehingga kita tahu darimana *class-class* tersebut berasal.