Belajar Santai

OOP PHP

Belajar Santai OOP PHP

Memahami Konsep OOP dengan Cara yang Mudah

Muhamad Surya Iksanudin

Buku ini dijual di http://leanpub.com/belajar-santai-oop-php

Versi ini diterbitkan pada 2016-07-29

Contents

Daftar Bab yang dibahas

Bab-XXII-Singleton-pattern.txt

Bab-XXIII-Observer-pattern.txt

Bab-XXIV-Factory-pattern.txt

Bab-XXV-Membuat-framework-sederhana.txt

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 diinstansiasi 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*.

```
1  <?php
2
3  //Class
4  class Mobil
5  {
6      //Property
7      private $jumlahRoda;
8
9      //Property
10     private $jumlahKursi;
11
```

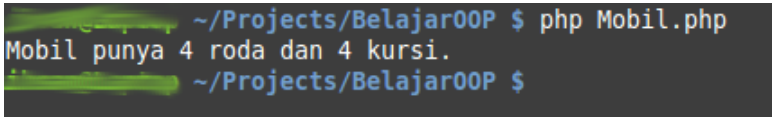
```
12      //Method
13      public function setJumlahRoda($jumlahRoda)
14      {
15          $this->jumlahRoda = $jumlahRoda;
16      }
17
18      //Method
19      public function setJumlahKursi($jumlahKursi)
20      {
21          $this->jumlahKursi = $jumlahKursi;
22      }
23
24      //Method
25      public function cetak()
26      {
27          echo 'Mobil punya '.$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 *output*-nya akan tampak sebagai berikut:



Mobil

Oiya, sekedar informasi, untuk semua contoh dalam buku dijalankan menggunakan *terminal* pada Linux dan Mac atau *command prompt* pada windows. Sehingga Anda perlu memastikan bahwa Komputer atau PC Anda dapat menjalankan *command* PHP. Anda dapat mengeceknya dengan menjalankan perintah berikut:

```
1  php --version
```

Bila *command* diatas tidak dikenali, maka Anda perlu mendaftakan ***PATH*** PHP Anda ke *environment variable*. Anda dapat *searching* menggunakan *keyword* add php to environment variable[2] .

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.

[2] https://www.google.com/search?q=add+php+to+environment+variable

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 misal `$mobil2` maka `$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();//Output: 4
echo PHP_EOL;
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:

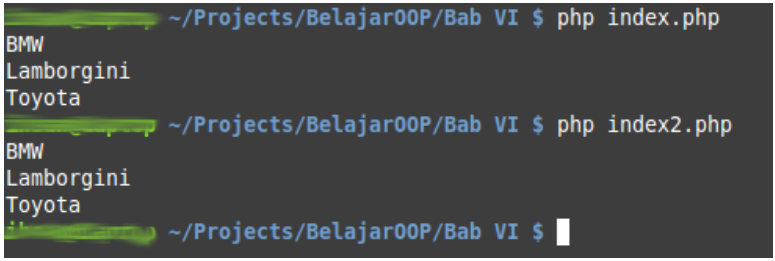
```php
<?php

//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

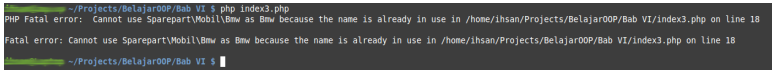
Sebelum saya menjelaskan fungsi dari *keyword* `as`, terlebih dahulu Anda akan saya ajak untuk melakukan percobaan sederhana berikut.

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

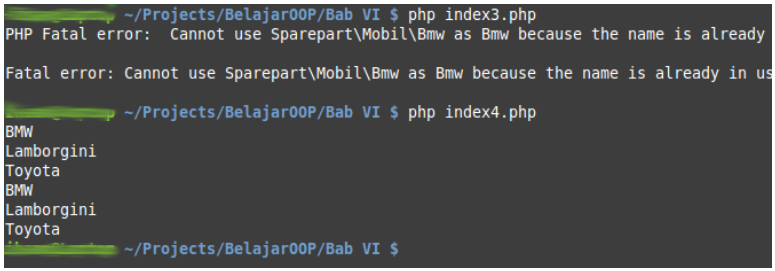
sendiri.

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:

```
1  <?php
2
3  //filename: index4.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 as KendaraanBmw;
15 use Kendaraan\Mobil\Lamborgini as KendaraanLamborgini;
16 use Kendaraan\Mobil\Toyota as KendaraanToyota;
17
18 use Sparepart\Mobil\Bmw as SparepartBmw;
19 use Sparepart\Mobil\Lamborgini as SparepartLamborgini;
20 use Sparepart\Mobil\Toyota as SparepartToyota;
21
22 echo KendaraanBmw::MEREK.PHP_EOL;
23 echo KendaraanLamborgini::MEREK.PHP_EOL;
24 echo KendaraanToyota::MEREK.PHP_EOL;
25
26 echo SparepartBmw::MEREK.PHP_EOL;
27 echo SparepartLamborgini::MEREK.PHP_EOL;
28 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.

Anonymous Function dan *Anonymous Class*

I. Pengantar

Pernahkah Anda menggunakan jQuery? Jika pernah berarti Anda tidak asing dengan *syntax* dibawah ini:

```
1  $(document).ready(function () {
2          //body
3  });
```

Taukah Anda nama dari baris *code* berikut:

```
1  function () {
2          //body
3  }
```

Yup, nama dari baris *code* diatas adalah *anonymous function* atau *fungsi tanpa nama*. Dalam PHP, konsep tersebut diadaptasi dan diimplementasikan lebih dalam lagi sehingga pada PHP tidak hanya ada *anonymous function* tapi juga ada *anonymous class*.

Pada pembahasan kali ini, kita akan lebih dalam membahas tentang *anonymous function* dan *anynomous class*.

II. *Anonymous Function*

Anonymous function atau dikenal juga dengan *closure* adalah sebuah *function* yang tidak memiliki nama secara spesifik. Dia hanya

memiliki *body* saja seperti *callback function* pada *javascript*. Untuk lebih jelas, perhatikan contoh dibawah ini:

```
1  <?php
2
3  $tambah = function ($a, $b)
4  {
5      return $a + $b;
6  };
7
8  echo $tambah(4, 6);
9  echo PHP_EOL;
```

Pada contoh diatas, kita membuat *anonymous function* yang di-*assign* kedalam variabel `$tambah` sehingga variabel `$tambah` menjadi `callable`. Sehingga bila kita mengetest dengan *function* `is_callable()` seperti dibawah ini:

```
1  <?php
2
3  $tambah = function ($a, $b)
4  {
5      return $a + $b;
6  };
7
8  echo $tambah(4, 6);
9  echo PHP_EOL;
10 var_dump(is_callable($tambah));
```

Maka hasilnya adalah sebagai berikut:

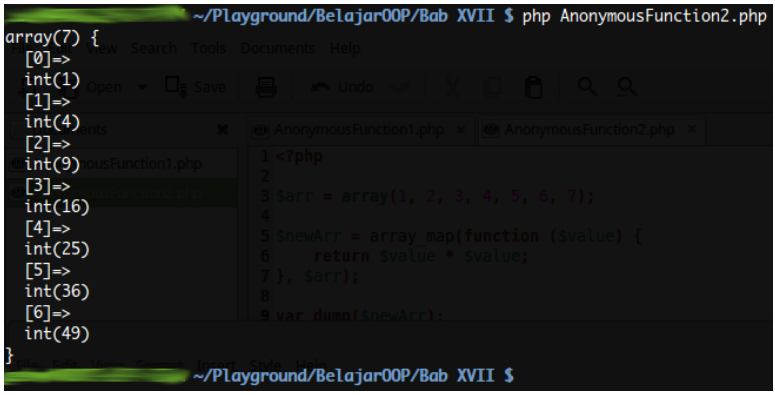
```
~/Playground/BelajarOOP/Bab XVII $ php AnonymousFunction1.php
10
bool(true)
~/Playground/BelajarOOP/Bab XVII $
```

Anonymous Function

Kita juga bisa menggunakan *anonymous function* sebagai *callback* seperti pada *javascript*. Berikut adalah contohnya:

```
1  <?php
2
3  $arr = array(1, 2, 3, 4, 5, 6, 7);
4
5  $newArr = array_map(function ($value) {
6      return $value * $value;
7  }, $arr);
8
9  var_dump($newArr);
```

Pada contoh diatas, kita menggunakan `array_map`[3] untuk memangkatkan *value* yang dari `$arr`. Bila *code* diatas dieksekusi maka hasilnya adalah sebagai berikut:



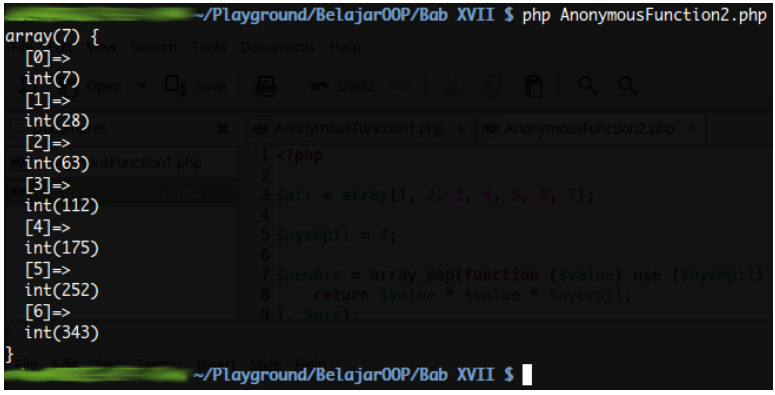
Array Map

Kita juga dapat memasukkan variabel dari luar kedalam *anonymous function* agar dapat dipanggil didalam *anonymous function* sebagai berikut:

[3] http://php.net/manual/en/function.array-map.php

```php
<?php

$arr = array(1, 2, 3, 4, 5, 6, 7);

$nyempil = 7;

$newArr = array_map(function ($value) use ($nyempil) {
    return $value * $value * $nyempil;
}, $arr);

var_dump($newArr);
```

Sehingga bila *code* diatas dijalankan, maka *output*-nya sebagai berikut:



Anonymous plus variabel luar

Bagaimana, mudah bukan? Tidak jauh berbeda dengan jQuery, bukan?

III. *Anonymous Class*

Anonymous class pertama kali diperkenalkan pada PHP versi 7.0. Fitur ini adalah fitur baru sehingga belum banyak diimplemen-

tasikan. Tujuan dari *anonymous class* adalah untuk mempermudah dalam pembuatan *object* yang simpel.

Karena baru diperkenalkan pada PHP versi 7.0, maka untuk mencoba fitur ini, pastikan versi PHP yang terinstall pada komputer Anda lebih baru atau minimal versi 7.0. Untuk memastikan hal tersebut, jalankan perintah sederhana dibawah ini:

```
1  php --version
```

Maka *output*-nya adalah sebagai berikut:

```
1  PHP 7.0.2 (cli) (built: Jan 11 2016 11:47:31) ( NTS )
2  Copyright (c) 1997-2015 The PHP Group
3  Zend Engine v3.0.0, Copyright (c) 1998-2015 Zend Techno\
4  logies
```

Terlihat versi PHP yang saya pakai adalah versi 7.0.2 sehingga saya dapat menggunakan fitur *anonymous class* ini.

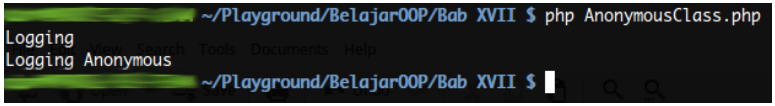
Untuk lebih jelas tentang *anonymous class*, perhatikan contoh berikut:

```
1  <?php
2
3  class Logger
4  {
5      public function log($message)
6      {
7          echo $message;
8      }
9  }
10
11 class Util
12 {
```

```
13      private $logger;
14
15      public function setLogger($logger)
16      {
17          $this->logger = $logger;
18      }
19
20      public function log($message)
21      {
22          $this->logger->log($message);
23      }
24  }
25
26  $util = new Util();
27  $util->setLogger(new Logger());
28  $util->log('Logging');
29  echo PHP_EOL;
30
31  $util->setLogger(new class {
32      public function log($message)
33      {
34          echo $message;
35      }
36  });
37  $util->log('Logging Anonymous');
38  echo PHP_EOL;
```

Pada *code* diatas, kita membuat *class* `Logger` yang nantinya akan dimasukkan kedalam *class* `Util`. Pada percobaan pertama, kita mencoba memasukkan *object* `Logger` kedalam *object* `Util`. Sedangkan pada percobaan kedua, kita menggunakan *anonymous class* untuk menggantikan *object* `Logger` pada percobaan pertama.

Bila program diatas dijalankan, maka *output*-nya adalah sebagai berikut:



Anonymous Class

Tidak hanya itu saja, kita juga bisa memasukkan *constructor parameter* serta membuat *property* pada *anonymous class*. Perhatikan contoh berikut:

```php
<?php

class Util
{
    private $logger;

    public function setLogger($logger)
    {
        $this->logger = $logger;
    }

    public function log()
    {
        $this->logger->log();
    }
}

$util = new Util();
$util->setLogger(new class('Logging Anonymous') {
    private $message;

    public function __construct($message)
    {
        $this->message = $message;
    }

```

```
27      public function log()
28      {
29          echo $this->message;
30      }
31  });
32  $util->log();
33  echo PHP_EOL;
```

Bila *code* diatas, dijalankan maka hasilnya adalah sebagai berikut:

```
~/Playground/BelajarOOP/Bab XVII $ php AnonymousClass2.php
Logging Anonymous
~/Playground/BelajarOOP/Bab XVII $
```

Anonymous constructor

Bukan hanya itu saja, kita juga *extends class*, *implements interface* bahkan *use trait* pada *anonymous class*. Istilah mudahnya, apa yang Anda dapat lakukan pada sebuah *class*, pada *anonymous class* pun dapat Anda lakukan.

Meski sangat *powerful*, namun saya tidak menyarankan untuk menggunakan fitur ini pada implementasi yang kompleks. Anda lebih baik menggunakan *real class* daripada menggunakan *anonymous class* untuk implementasi yang kompleks.

Ini karena, semakin kompleks *anonymous class* akan semakin sulit Anda ketika memaintain *source code* yang Anda tulis dikemudian hari.

Membuat *Framework* Sederhana

I. Pengantar

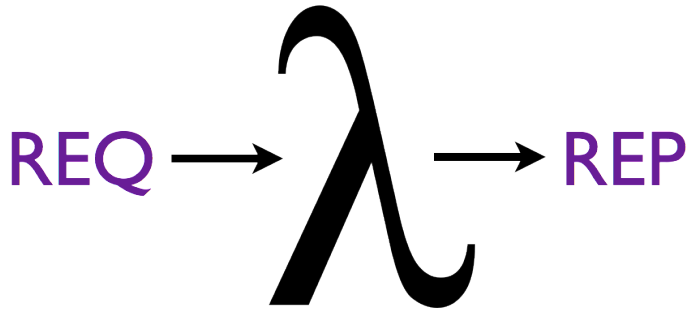
Setelah kita belajar tentang OOP hingga *design pattern*, maka saatnya kita mencoba mengimplementasikan semua yang kita pelajari kedalam sebuah aplikasi yang *real*.

Sebagai *final project*, kita akan membuat sebuah *framework* sederhana berbasis OOP dan menggunakan *composer* sebagaimana yang telah kita pelajari sebelumnya.

II. *Http Request* dan *Http Response*

Request adalah segala sesuatu yang dikirimkan oleh *client* kepada aplikasi, sedangkan *response* adalah segala sesuatu yang dikembalikan oleh aplikasi kepada *client*.

Secara mudah, diagram proses dari *request* hingga *response* dapat digambarkan sebagai berikut:



Request dan Response

Tanda *lamba*, adalah representasi dari aplikasi kita.

III. *Framework Dependencies*

Sebelum memulai, mari kita definisikan terlebih dahulu *dependency* dari *framework* yang akan kita buat. Dalam *framework* yang akan kita buat, kita akan menggunakan *Symfony Component* untuk mempermudah kita dalam membangun *framework* kita.

Kenapa *Symfony Component*? Karena *Symfony Component* terutama *Http Kernel Component* telah menjadi *standard* pada *StackPHP*[4] serta telah digunakan oleh banyak *project open source* seperti *Laravel, Silex* dan *Drupal.*

Symfony Component yang akan kita gunakan adalah sebagai berikut:

[4] http://stackphp.com

```
{
    "name": "belajar-oop/framework",
    "require": {
        "symfony/http-foundation": "3.*",
        "symfony/http-kernel": "3.*",
        "symfony/routing": "3.*",
        "symfony/event-dispatcher": "3.*"
    },
    "autoload": {
        "psr-4": {
            "BelajarOOP\\Framework": "src/"
        }
    }
}
```

Pada `composer.json` diatas terlihat bahwa *framework* kita mempunyai empat *dependency* yang semuanya berasal dari *Symfony Component*.

Berikut adalah sedikit penjelasan kegunaan masing-masing *component*:

- *Http Foundation*: *Component* ini berisi *element* dasar *Http* seperti *request*, *response*, dan *session*.
- *Http Kernel*: *Component* ini adalah *component* utama kita yang menjadi inti dari *framework* yang kita buat. *Component* ini bertugas meng-*handle request*, *routing* hingga *response*.
- *Routing*: *Component* ini berisi *class* untuk mengatur *routing* aplikasi.
- *Event Dispatcher*: *Component* ini adalah sebuah *component* yang mengimplementasikan *observer*

pattern. *Component* ini kita perlukan agar *framework* yang kita buat lebih fleksibel.

Setelah kita mendefinisikan *dependency* seperti diatas, sekarang kita jalankan *composer update* untuk men-*download* semua *dependency* diatas.

Setelah selesai, maka berikut adalah susunan *folder framework* kita:

Susunan Folder

IV. Membuat *Kernel Framework*

Langkah pertama yang akan kita lakukan dalam membuat *framework* adalah mengimplementasikan `HttpKernelInterface` dari *HttpKernel Component*. Berikut adalah *class* `Kernel` yang mengimplementasikan `HttpKernelInterface`sebagai berikut:

```
1  <?php
2
3  namespace BelajarOOP\Framework\Http;
4
5  use Symfony\Component\HttpFoundation\Request;
6  use Symfony\Component\HttpKernel\HttpKernelInterface;
7
8  class Kernel implements HttpKernelInterface
9  {
10     public function __construct()
11     {
12         // TODO: Implement __construct() method.
13     }
```

```
14
15      public function handle(Request $request, $type = se\
16  lf::MASTER_REQUEST, $catch = true)
17      {
18          // TODO: Implement handle() method.
19      }
20  }
```

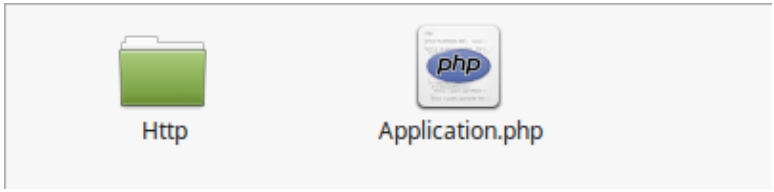
Setelah itu, membuat *class* `Application` yang meng-*extends class* `Kernel`. Hal ini kita lakukan agar, *class* `Kernel` lebih bisa fokus hanya mengurusi *request* dan *response* saja. Berikut adalah *class* `Application` yang kita buat:

```
1   <?php
2
3   namespace BelajarOOP\Framework;
4
5   use BelajarOOP\Framework\Http\Kernel;
6
7   class Application extends Kernel
8   {
9       protected $configs;
10
11      public function __construct(array $configs = array(\
12  ))
13      {
14          parent::__construct();
15          $this->configs = $configs;
16      }
17
18      public function setConfig($key, $value)
19      {
20          $this->configs[$key] = $value;
21      }
```

```
22
23      public function getConfig($key)
24      {
25          if (array_key_exists($key, $this->configs)) {
26              return $this->configs[$key];
27          }
28
29          return null;
30      }
31  }
```

Class `Application` diatas nantinya digunakan untuk menyimpan konfigurasi dari *framework* yang kita buat. Seperti yang sudah dijelaskan sebelumnya, ini akan membuat kita lebih fokus karena *class* `Kernel` hanya mengurusi *request* dan *response*, sedangkan *class* `Application` mengurusi konfigurasi dari *framework* kita.

Sehingga sekarang susuna *folder* `src` kita tampak sebagai berikut:



Folder Src

Sampai disini *framework* kita belum bisa terlihat apapun.

V. *Routing*

Langkah selanjutnya, kita akan membuat *routing system* pada *framework* kita. *Routing system* diperlukan mengarahkan *request* menuju ke *controller* yang meng-*handle*-nya.

Karena kita menggunakan *Routing Component* dari *Symfony*, maka kita tidak perlu lagi membuat *routing system* sendiri, kita hanya cukup mengimplementasikan *routing system* tersebut kedalam *framework* kita.

Berikut adalah *update* dari *class* `Kernel` setelah kita mengimplementasikan *routing system*:

NOT AVAILABLE

Sampai disini, sebenarnya *framework* kita sudah jadi. Serius, *core framework* sebenarnya memang sesimpel itu. Yang membuat *framework* terkesan kompleks adalah karena adanya proses validasi dan lain-lain pada *core* atau *kernel* dari framework tersebut.

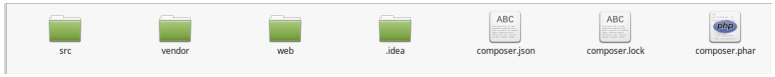
Untuk *framework* yang kita buat, proses-proses tersebut ditiadakan agar Anda dapat fokus memahami bahwa sebenarnya membuat *framework* itu sangatlah mudah.

VI. *Front Controller*

Berikutnya yang akan kita buat adalah *file* pemanggil atau *front controller*. *File* inilah yang nantinya akan dieksekusi oleh *client* ketika hendak masuk ke *system* kita.

Hampir semua *framework modern* menggunakan *front controller* sebagai *gateway* atau pintu gerbang untuk masuk ke *system*. Keuntungan dari *front controller* adalah kita hanya mempunyai satu *file* atau pintu untuk *client* masuk sehingga lebih mudah untuk di*manage*.

Untuk *file front controller* agar mudah, saya menamainya dengan `index.php` dan saya simpan pada *folder* `web` yang sejajar dengan *folder* `src` seperti tampak pada gambar:



Front Controller

Dan berikut adalah *file* `index.php` atau *front controller* dari *framework* yang kita buat:

NOT AVAILABLE

Seperti yang saya sudah saya jelaskan diatas, bahwa *framework* kita sudah jadi. Untuk memastikan kebenaran tersebut, mari kita coba jalankan *framework* tersebut.

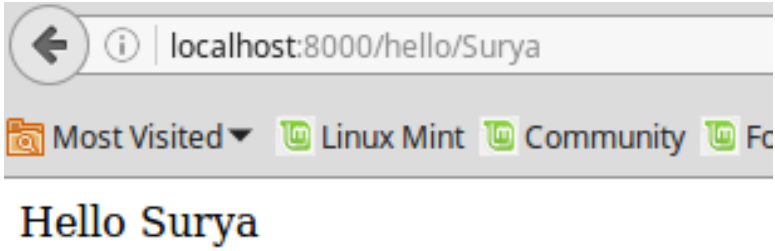
Untuk mempermudah dan agar tidak perlu setting *vhost* dan *web server*, saya menggunakan *built-in web server* untuk menjalankan *framework* kita. Cukup mengetikan `php -S localhost:8000` dari *folder* `web` maka aplikasi kita sudah berjalan.

Berikut adalah tampilan *command* untuk menjalankan *web server*:

```
~/Projects/BelajarOOP/Bab XXIV/web $ php -S localhost:8000
PHP 5.6.17 Development Server started at Thu Jun 16 07:24:41 2016
Listening on http://localhost:8000
Document root is /[redacted]n/Projects/BelajarOOP/Bab XXIV/web
Press Ctrl-C to quit.
```

Built In Web Server

Dan tampilan dari *framework* kita, ketika ketika mengetikkan `localhost:8000/hello/<Nama>` adalah sebagai berikut:



Framework

Bagaimana? Mudah sekali bukan sebenarnya. Untuk langkah selanjutnya, kita akan menerapkan *observer pattern* pada *framework* kita agar, *framework* yang kita buat makin fleksibel.

VII. *Event Dispatcher*

NOT AVAILABLE

VIII. Kesimpulan

NOT AVAILABLE