



# PEMROGRAMAN BERBASIS OBJEK MODERN DENGAN PHP

Muhamad Surya Iksanudin

---

# Table of Contents

## Judul

|       |     |
|-------|-----|
| Judul | 1.1 |
|-------|-----|

## Prakata

|                                                                    |     |
|--------------------------------------------------------------------|-----|
| Kata Pengantar                                                     | 2.1 |
| Tentang Penulis                                                    | 2.2 |
| Testimonial (Yayan Sopiyan, Mizno, Peter Kembay, Hendro Wicaksono) | 2.3 |

## Pengenalan PHP

|                             |     |
|-----------------------------|-----|
| Sejarah PHP                 | 3.1 |
| Market Share PHP            | 3.2 |
| PHP 7 - The Next Generation | 3.3 |
| Perbandingan PHP 7          | 3.4 |

## Pemrograman Berbasis Objek

|                                       |     |
|---------------------------------------|-----|
| Apa itu Pemrograman Berbasis Objek    | 4.1 |
| Kelebihan Pemrograman Berbasis Objek  | 4.2 |
| Kekurangan Pemrograman Berbasis Objek | 4.3 |
| Kenapa harus belajar OOP              | 4.4 |
| Fitur Dasar Yang Ada pada OOP         | 4.5 |

## Class dan Object

|                                |     |
|--------------------------------|-----|
| Apa itu Class                  | 5.1 |
| Contoh Class pada PHP          | 5.2 |
| Anatomi Class                  | 5.3 |
| Pembuatan Object (Instansiasi) | 5.4 |

## Property dan Method

|                              |     |
|------------------------------|-----|
| Apa itu Property             | 6.1 |
| Apa itu Method               | 6.2 |
| Urutan Parameter pada Method | 6.3 |

## Visibilitas

---

|                                |     |
|--------------------------------|-----|
| Apa itu Visibilitas            | 7.1 |
| Tingkatan Visibilitas pada PHP | 7.2 |
| Tips Visibilitas               | 7.3 |

---

## Konsep Statis dan Konstanta

|                                   |     |
|-----------------------------------|-----|
| Apa itu Statis                    | 8.1 |
| Property dan Method statis        | 8.2 |
| Contoh Penerapan Konsep Statis    | 8.3 |
| Apa itu Konstanta                 | 8.4 |
| Contoh Penerapan Konsep Konstanta | 8.5 |

---

## Keyword \$this dan self

|                                   |     |
|-----------------------------------|-----|
| Apa itu \$this                    | 9.1 |
| Apa itu self                      | 9.2 |
| Perbedaan \$this dan self         | 9.3 |
| Contoh Penggunaan \$this dan self | 9.4 |

---

## Return Value

|                                |      |
|--------------------------------|------|
| Apa itu Return Value           | 10.1 |
| Contoh Penggunaan Return Value | 10.2 |

---

## Late Static Bindings

|                                        |      |
|----------------------------------------|------|
| Apa itu Late Static Bindings           | 11.1 |
| Contoh Penggunaan Late Static Bindings | 11.2 |

---

## Constructor dan Destructor

|                                              |      |
|----------------------------------------------|------|
| Apa itu Constructor                          | 12.1 |
| Apa itu Desctructor                          | 12.2 |
| Contoh Penggunaan Constructor dan Destructor | 12.3 |

---

## Enkapsulasi Data

|                               |      |
|-------------------------------|------|
| Apa itu Enkapsulasi           | 13.1 |
| Contoh Penggunaan Enkapsulasi | 13.2 |

---

## Pewarisan

|                   |      |
|-------------------|------|
| Apa itu Pewarisan | 14.1 |
| Keyword parent    | 14.2 |

---

---

|                            |      |
|----------------------------|------|
| Contoh Penerapan Pewarisan | 14.3 |
|----------------------------|------|

---

## Overloading dan Overriding

|                               |      |
|-------------------------------|------|
| Apa itu Overloading           | 15.1 |
| Contoh Penggunaan Overloading | 15.2 |
| Apa itu Overriding            | 15.3 |
| Contoh Penggunaan Overriding  | 15.4 |

## Pengelompokan Berkas

|                                        |      |
|----------------------------------------|------|
| Pengantar                              | 16.1 |
| Keyword namespace dan cara pemanggilan | 16.2 |
| Memberikan alias dengan keyword as     | 16.3 |

## Abstract Class dan Abstract Method

|                                                      |      |
|------------------------------------------------------|------|
| Apa itu Abstract Class                               | 17.1 |
| Apa itu Abstract Method                              | 17.2 |
| Kegunaan Abstract Class dan Abstract Method          | 17.3 |
| Contoh Penggunaan Abstract Class dan Abstract Method | 17.4 |

## Interface dan Trait

|                                |      |
|--------------------------------|------|
| Apa itu Interface              | 18.1 |
| Contoh Penggunaan Interface    | 18.2 |
| Apa itu Trait                  | 18.3 |
| Contoh Penggunaan Trait        | 18.4 |
| Best Practice Penggunaan Trait | 18.5 |

## Parameter Casting

|                                                          |      |
|----------------------------------------------------------|------|
| Apa itu Parameter Casting                                | 19.1 |
| Primitive Type Casting (string, int, float, bool, array) | 19.2 |
| Object Type Casting                                      | 19.3 |

## Exception Handling

|                                                      |      |
|------------------------------------------------------|------|
| Apa itu Exception Handling                           | 20.1 |
| Contoh Exception Handling Menggunakan Blok try catch | 20.2 |

## Anonymous Function dan Anonymous Class

|                            |      |
|----------------------------|------|
| Apa itu Anonymous Function | 21.1 |
|----------------------------|------|

|                                                          |      |
|----------------------------------------------------------|------|
| Apa itu Anonymous Class                                  | 21.2 |
| Contoh Penggunaan Anonymous Function dan Anonymous Class | 21.3 |

## Cara Membuat Chaining Method

|                                |      |
|--------------------------------|------|
| Apa itu Chaining Method        | 22.1 |
| Cara Pembuatan Chaining Method | 22.2 |

## Magic Method pada PHP

|                                |       |
|--------------------------------|-------|
| Apa itu Magic Method           | 23.1  |
| Penggunaan Magic Method        | 23.2  |
| __construct() dan __destruct() | 23.3  |
| __set() dan __get()            | 23.4  |
| __isset() dan __unset()        | 23.5  |
| __sleep() dan __wakeup()       | 23.6  |
| __call() dan __callStatic()    | 23.7  |
| __toString()                   | 23.8  |
| __invoke()                     | 23.9  |
| __clone()                      | 23.10 |

## Final Class dan Final Method

|                                                |      |
|------------------------------------------------|------|
| Apa itu Final Class                            | 24.1 |
| Apa itu Final Method                           | 24.2 |
| Contoh Penggunaan Final Class dan Final Method | 24.3 |

## Object Cloning dan Object Comparing

|                                    |      |
|------------------------------------|------|
| Apa itu Object Cloning             | 25.1 |
| Contoh Penggunaan Object Cloning   | 25.2 |
| Apa itu Object Comparing           | 25.3 |
| Contoh Penggunaan Object Comparing | 25.4 |

## Studi Kasus Perhitungan Pajak PPH21 dengan OOP

|                              |      |
|------------------------------|------|
| Persiapan Proyek             | 26.1 |
| Cara Perhitungan Pajak PPH21 | 26.2 |
| Pengelompokan Masalah        | 26.3 |
| Perancangan Solusi           | 26.4 |
| Penulisan Code               | 26.5 |

## Coding Standard

---

|                                                        |      |
|--------------------------------------------------------|------|
| Apa itu FIG dan PSR                                    | 27.1 |
| Kenapa harus menerapkan PSR                            | 27.2 |
| Aturan Penulisan Class dan Namespace (PSR-0 dan PSR-4) | 27.3 |
| Aturan Penulisan Syntax (PSR-1 dan PSR-2)              | 27.4 |

## Package Management dengan Composer

|                             |      |
|-----------------------------|------|
| Apa itu Composer            | 28.1 |
| Kenapa Menggunakan Composer | 28.2 |
| Minimum System Requirement  | 28.3 |
| Instalasi Composer          | 28.4 |
| Tentang composer.json       | 28.5 |
| Anatomi composer.json       | 28.6 |
| Autoload dengan Composer    | 28.7 |

## Membuat Package Sendiri

|                                   |      |
|-----------------------------------|------|
| Pembuatan composer.json           | 29.1 |
| Daftar Github                     | 29.2 |
| Upload ke Github                  | 29.3 |
| Daftar Packagist                  | 29.4 |
| Pendaftaran Package               | 29.5 |
| Sinkronisasi Github dan Packagist | 29.6 |

## Studi Kasus Membuat Framework Sederhana

|                                        |      |
|----------------------------------------|------|
| Persiapan Proyek                       | 30.1 |
| Skop Proyek                            | 30.2 |
| Tentang Front Controller (index.php)   | 30.3 |
| Tentang HTTP Request dan HTTP Response | 30.4 |
| Mengarahkan request dengan Router      | 30.5 |
| Membuat Controller Class               | 30.6 |
| Memisahkan Logic                       | 30.7 |
| Menambahkan Middleware                 | 30.8 |
| Kesimpulan                             | 30.9 |

## Studi Kasus Todo List Menggunakan OOP dan MVC

|                                                      |      |
|------------------------------------------------------|------|
| Persiapan Proyek                                     | 31.1 |
| Goal dari Proyek Todo List                           | 31.2 |
| Pembuatan Database                                   | 31.3 |
| Menambahkan Fitur Manipulasi Database pada Framework | 31.4 |
| Membuat Todo Class                                   | 31.5 |

---

|                                                     |      |
|-----------------------------------------------------|------|
| Membuat Model Class                                 | 31.6 |
| Membuat Controller Class dan Mendaftarkan ke Router | 31.7 |
| Menambahkan Template Engine pada Framework          | 31.8 |
| Kesimpulan                                          | 31.9 |

## Penutup

# **Pemrograman Berbasis Objek Modern**

## **Menggunakan PHP7**

**Belajar Pemrograman Berbasis Objek (OOP) modern dengan cara yang santai,  
menyenangkan dan mudah diimplementasikan**

**Muhamad Surya Iksanudin**

## Kata Pengantar

Segala puji bagi Allah SWT, yang senantiasa melimpahkan rahmat serta karunia-Nya kepada penulis sehingga dapat menyelesaikan buku dengan judul "Pemrograman Berbasis Objek Modern Menggunakan PHP7" ini. Shalawat serta salam penulis haturkan kepada Rasulullah Sholallahu Alaihi Wassalam yang menjadi teladan terbaik bagi umat manusia. Rasul yang membawa kita dari jalan gelap menuju cahaya.

Terima kasih penulis sampaikan kepada seluruh pihak yang telah membantu terwujudnya buku ini terutama saudara-saudara seperjuangan di Komunitas Symfony Framework Indonesia yang tidak pernah lelah membantu penulis dalam penyusunan buku ini. Terima kasih juga penulis ucapkan khususnya kepada Bapak Hendro Wicaksono yang telah bersedia mengoreksi isi dari buku ini. Terima kasih penulis haturkan kepada Muhibbudin Suretno yang telah membuatkan sampul untuk buku ini.

Penulis menyadari dalam penulisan buku ini terdapat banyak kekurangan. Penulis mengharapkan kritik dan saran dari pembaca. Harapan penulis makalah ini bisa bermanfaat bagi pembaca dan bagi penelitian berikutnya.

Jakarta, Juli 2018

Muhamad Surya Iksanudin

## Tentang Penulis



*Muhamad Surya Iksanudin*

Penulis lahir di Pemalang, 3 April 1988. Penulis tercatat sebagai salah satu member di [Komunitas Symfony Framework Indonesia](#) sebuah forum diskusi berbagai permasalahan yang ada di dunia pemrograman khususnya PHP dan lebih khusus lagi tentang [Framework Symfony](#).

Selain itu penulis juga bekerja sebagai pekerja lepas sebagai *System Administrator* maupun *Web Developer*. Penulis menulis buku ini sebagai salah satu bentuk kontribusi penulis dalam ekosistem PHP yang telah membesarkan penulis.

# Pengenalan PHP

## Sejarah Bahasa Pemrograman PHP

Menurut [wikipedia](#), Pada awalnya PHP merupakan kependekan dari Personal Home Page (Situs personal). PHP pertama kali dibuat oleh Rasmus Lerdorf pada tahun 1995. Pada waktu itu PHP masih bernama Form Interpreted (FI), yang wujudnya berupa sekumpulan skrip yang digunakan untuk mengolah data formulir dari web.

Selanjutnya Rasmus merilis kode sumber tersebut untuk umum dan menamakannya PHP/FI. Dengan perilsan kode sumber ini menjadi sumber terbuka, maka banyak pemrogram yang tertarik untuk ikut mengembangkan PHP.

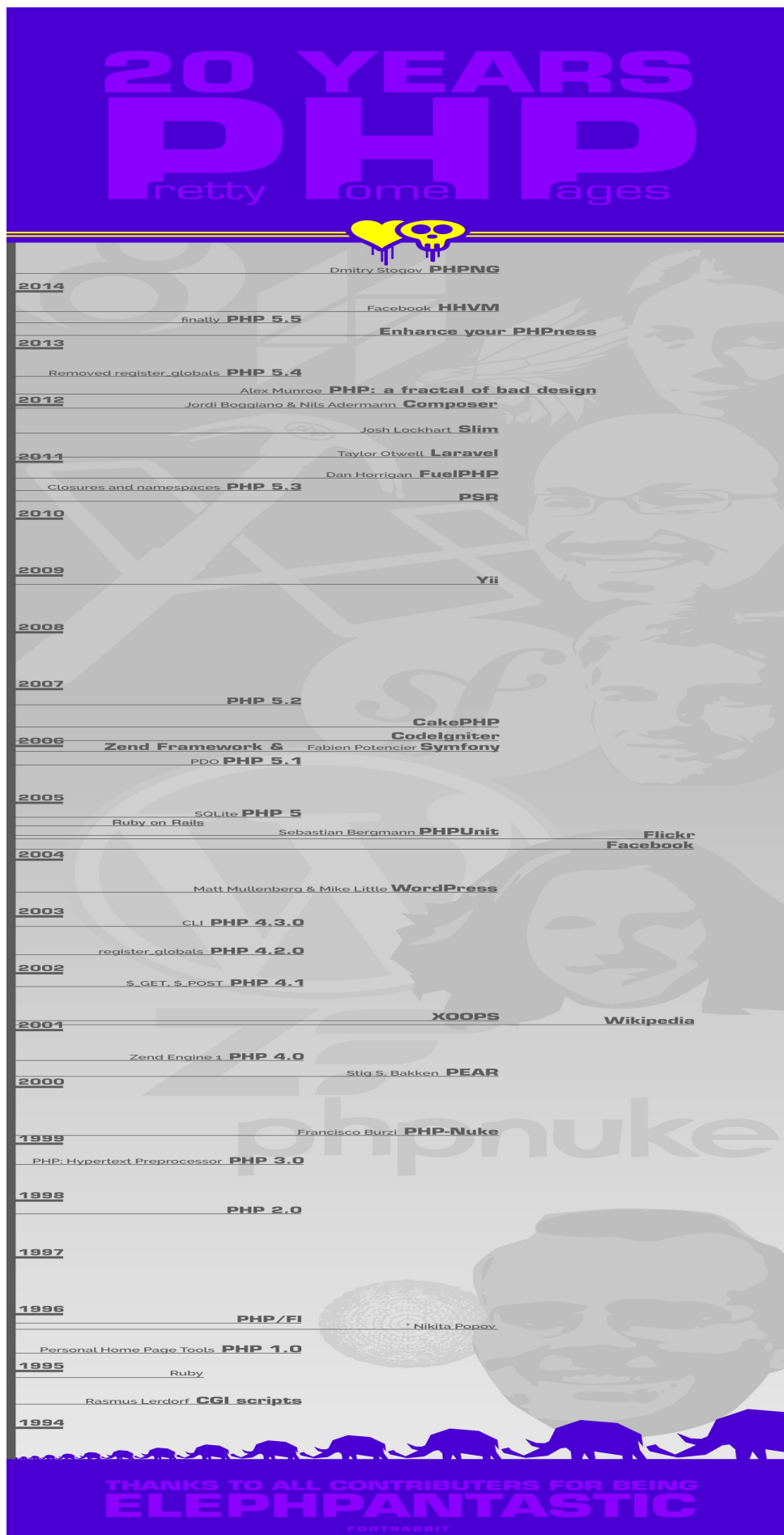
Pada November 1997, dirilis PHP/FI 2.0. Pada rilis ini, interpreter PHP sudah diimplementasikan dalam program C. Dalam rilis ini disertakan juga modul-modul ekstensi yang meningkatkan kemampuan PHP/FI secara signifikan.

Pada tahun 1997, sebuah perusahaan bernama Zend menulis ulang interpreter PHP menjadi lebih bersih, lebih baik, dan lebih cepat. Kemudian pada Juni 1998, perusahaan tersebut merilis interpreter baru untuk PHP dan meresmikan rilis tersebut sebagai PHP 3.0 dan singkatan PHP diubah menjadi akronim berulang PHP: Hypertext Preprocessing.

Pada pertengahan tahun 1999, Zend merilis interpreter PHP baru dan rilis tersebut dikenal dengan PHP 4.0. PHP 4.0 adalah versi PHP yang paling banyak dipakai pada awal abad ke-21. Versi ini banyak dipakai disebabkan kemampuannya untuk membangun aplikasi web kompleks tetapi tetap memiliki kecepatan dan stabilitas yang tinggi.

Pada Juni 2004, Zend merilis PHP 5.0. Dalam versi ini, inti dari interpreter PHP mengalami perubahan besar. Versi ini juga memasukkan model pemrograman berorientasi objek ke dalam PHP untuk menjawab perkembangan bahasa pemrograman ke arah paradigma berorientasi objek. Server web bawaan ditambahkan pada versi 5.4 untuk mempermudah pengembang menjalankan kode PHP tanpa menginstall software server.

Pada saat buku ini ditulis, PHP telah mencapai versi 7.2 dengan penambahan ekstensi dan perbaikan performa yang menjanjikan. Berikut adalah info grafis tentang sejarah dan perkembangan PHP serta ekosistemnya dari awal hingga tahun 2015 yaitu ketika PHP 7 atau yang juga dikenal dengan PHP Next Generation (PHPNG) dirilis publik.

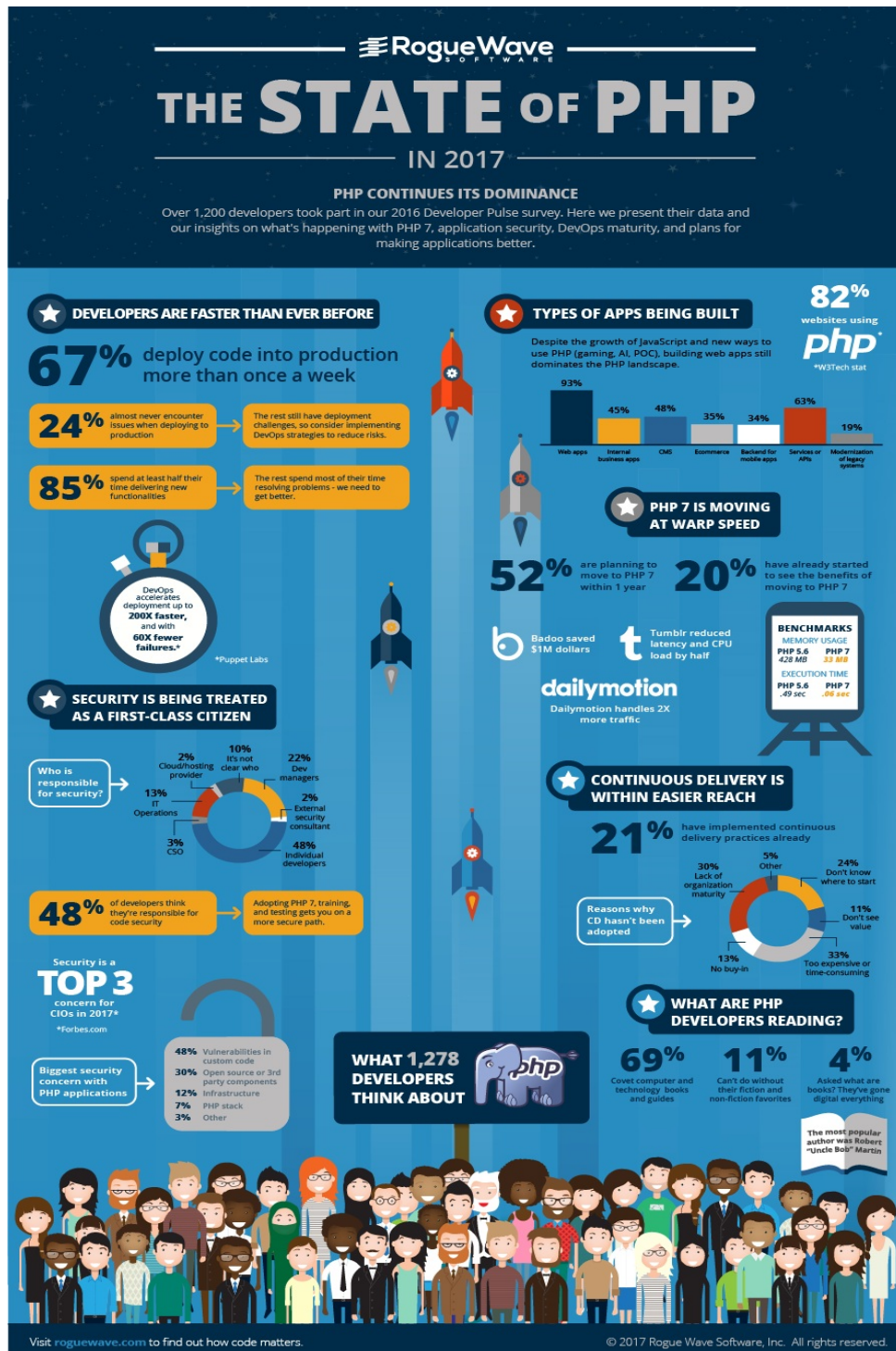


Sumber: <https://blog.fortrabbbit.com>

## Market Share PHP

Banyak yang menyangsikan PHP akan bertahan dan mampu bersaing di era Web 3.0 seperti sekarang ini. Namun komitmen PHP untuk terus menghadirkan inovasi dan menjawab kebutuhan user (Developer) tetap membawa PHP menjadi pemimpin sebagai bahasa yang paling digunakan untuk pembuatan website maupun aplikasi berbasis internet. Tidak tanggung-tanggung, PHP meninggalkan bahasa pemrograman lain dengan *market share* lebih dari 80% diikuti oleh ASP.Net dan Java.

Pencapaian ini membuktikan bahwa PHP masih tetap diminati, terlebih dengan hadirnya PHP generasi terbaru (PHP 7) yang menghadirkan banyak fitur baru dari segi kecepatan eksekusi, kestabilan serta keamanan. Berikut adalah info grafis tentang *market share* PHP.



Sumber: <http://zend.com>

## PHP 7 - *The Next Generation*

Melihat kebelakang ketika PHP 5 pertama kali dirilis, versi ini adalah versi PHP pertama yang melahirkan banyak perubahan besar pada ekosistem PHP. Banyak sekali framework dan maupun tool pengembangan yang dirilis beberapa bulan setelah versi ini dirilis secara publik. *Framework* seperti Zend, Symfony, Prado, Cake, CodeIgniter, Yii dan Laravel adalah sebagian dari banyak framework yang mengiringi perjalanan PHP 5 selama lebih dari satu dekade. *Tool* dan *library* lain seperti PHPUnit, PHP Code Sniffer, Doctrine, Propel pun ikut meramaikan ekosistem PHP.

Ketika awal kemunculannya, PHP 5 yang mengusung konsep OOP, belum mampu menjawab beberapa kebutuhan seperti belum adanya `namespace`, `static type hinting`, `return type`, `closure` dan masih banyak lagi. Oleh karena ini, PHP terus berevolusi dan menambahkan fitur-fitur tersebut sedikit demi sedikit disamping juga tetap memberikan perbaikan keamanan dan stabilitas. Hingga PHP 5.6 yaitu versi terakhir dari PHP 5, PHP telah memiliki beberapa fitur seperti `namespace`, `array type hinting` dan `closure`.

Namun isu besar muncul pada PHP 5, yaitu masalah pada penggunaan memory yang boros serta waktu eksekusi yang lama. Masalah tersebut sebenarnya telah dijawab oleh Facebook dengan membuat *engine* yang diberinama HHVM (Hip Hop Virtual Machine) yaitu mesin yang mampu merubah *code* PHP menjadi *bytecode* seperti pada Java, dan kemudian mengeksekusinya. HHVM sendiri selain *men-support* PHP, juga *men-support* Hacklang, sebuah bahasa pemrograman yang *syntax*-nya sangat mirip dengan PHP yang dikembangkan oleh Facebook.

Kemunculan HHVM sendiri sebenarnya memberikan keresahan bagi Zend selaku pengembang PHP. Hal ini karena, selain HHVM, Facebook juga mempromosikan bahasa pemrograman baru yaitu Hacklang yang memang diciptakan untuk "menyaingi" PHP dan berjalan dengan optimal pada HHVM. Oleh karena itu, Zend kemudian *me-refactor* *Zend Engine* agar bisa lebih baik dari HHVM.

Setelah proses pengembangan yang cukup lama, pada Februari 2015 PHP 7 lahir sebagai *The Next Generation of PHP*. Banyak sekali perombakan yang dilakukan oleh Zend terutama pada performa dan penggunaan *memory* yang lebih kecil. Selain itu, beberapa fitur ditambahkan untuk melengkapi fitur OOP yang ada di PHP seperti `return type`, `static type hinting` selain juga perbaikan beberapa fitur. Dan berikut adalah info grafis tentang fitur yang ada pada PHP 7.



# RELEASE IN OCTOBER 2015

## SOME OF THE NEW PHP7 FEATURES

The PHPNG branch has been merged into master and has been used as the base for PHP 7.0. This branch is based on refactored Zend Engine, that's why PHP applications will become faster and faster.

### Removing PHP4 style constructors

PHP 7 will emit E\_DEPRECATED whenever a PHP 4 constructor is defined. When the method name matches the class name, the class is not in a namespace, and a PHP 5 constructor (`__construct`) is not present then an E\_DEPRECATED will be emitted.

Reclassify the few existing E\_STRICT notices and remove this error category. The E\_STRICT constant will be retained for better compatibility, it will simply no longer have meaning in PHP 7.

### Reclassify E\_STRICT notices

### Spaceship Operator <=>

New operator for combined comparison. Similar to `strcmp()` or `version_compare()` in behavior, but it can be used on all generic PHP alues with the same semantics as `<`, `<=`, `==`, `>=`, `>`.



Developers will be able to declare the return type of a function. Declaring return types prevent sub-types from breaking the expected return type of the super-type; prevent unintended return values.

```
function foo(): array {
    return [];
}
```

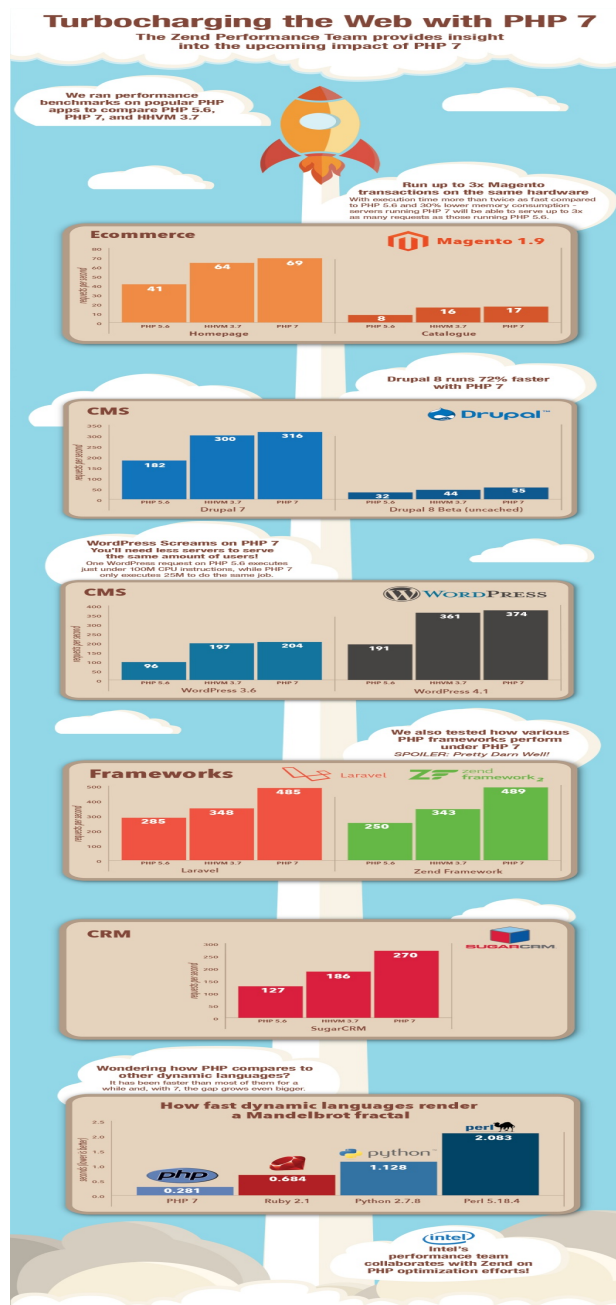
### Return Types

### Replacing current json extension

The current Json Parser in the json extension does not have a free license which is a problem for many Linux distros. So the json extension will be replaced with code that is based on the PECL jsond extension.

## Perbandingan PHP 7 dengan HHVM dan PHP 5

Seperti yang sudah disebutkan diatas, PHP 7 memiliki banyak keunggulan terutama yang sangat menonjol adalah performa dan penggunaan *memory* yang lebih kecil. Perbaikan performa ini menjadikan PHP kembali memimpin dan mengungguli HHVM dari sisi waktu eksekusi dalam beberapa *benchmark*. Dan berikut adalah info grafis perbandingan antara PHP 7, HHVM maupun PHP 5 menggunakan beberapa *software* baik itu *framework*, CMS, maupun PHP 7 dibandingkan dengan bahasa pemrograman lain.



Sumber: <http://www.zend.com>

Dari info grafis diatas, terbukti bahwa PHP 7 mampu menjawab tantangan dan kebutuhan pasar. Dan ini menunjukkan komitmen dari Zend selaku pengembang PHP untuk terus memberikan yang terbaik untuk pada developer PHP di seluruh dunia.

# Pengenalan Pemrograman Berbasis Objek

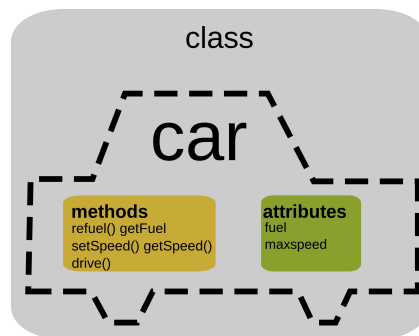
## Apa itu Pemrograman Berbasis Objek

Menurut [wikipedia](#), Pemrograman berorientasi objek merupakan paradigma pemrograman yang berorientasikan kepada objek. Semua data dan fungsi di dalam paradigma ini dibungkus dalam kelas-kelas atau objek-objek.

Dalam pemrograman berbasis objek, kita dituntut untuk memahami dan memecahkan masalah kedalam *class* serta memecah masalah kedalam *class-class* yang lebih kecil dan simpel agar solusi yang dibuat lebih spesifik. Selanjutnya, *class-class* tersebut akan saling berkomunikasi dan berkolaborasi untuk memecahkan masalah yang kompleks. *Class-class* ini nantinya akan dirubah menjadi objek-objek pada saat *runtime*.

Setiap *class* dalam OOP mempunyai *method* atau fungsi serta *property* atau atribut. *Method* dalam *class* secara mudah diartikan sebagai segala kemampuan dari *class* atau apa saja yang dapat dilakukan oleh sebuah *class*. Sedangkan *property* adalah segala sesuatu yang dimiliki oleh *class*. Dalam OOP, *property* dan *method* dalam *class* saling bekerjasama membangun sebuah solusi dari suatu masalah.

Dalam beberapa referensi, *method* disebut juga sebagai *function* sedangkan *property* sering disebut juga sebagai *attribute*. Sehingga Anda tidak perlu bingung bila nantinya dibuku lain, Anda menemui istilah *function* dan *attribute* sebagai pengganti *method* dan *property*. Yang perlu Anda pahami bahwa *function* atau *method* adalah fitur atau kemampuan dari sebuah *class* sedangkan *property* atau *attribute* adalah segala sesuatu yang dimiliki oleh sebuah *class*.



Sumber: <https://github.com>

## Kelebihan Pemrograman Berbasis Objek

Pemrograman berbasis objek atau biasa disebut OOP, memiliki banyak keunggulan dibandingkan paradigma pemrograman lainnya. Keunggulan pemrograman berbasis objek antara lain sebagai berikut:

- Modularitas: program yang dibuat dapat dipecah menjadi modul-modul yang lebih kecil dan nantinya digabungkan menjadi solusi yang utuh.
- Fleksibilitas: karena setiap solusi dibuat dalam bentuk *class*, ketika terjadi perubahan maka hanya *class* tersebut saja yang perlu dirubah.
- Ekstensibilitas: penambahan *method* atau *property* dapat dilakukan dengan sangat mudah.
- Reuse: *class* dapat digunakan berkali-kali untuk proyek maupun modul yang lain.
- Mudah dimaintain: karena setiap *class* berdiri sendiri, maka untuk memaintain *class* tersebut jauh lebih mudah.
- Keamanan code: adanya visibilitas memberikan fitur keamanan dimana developer lain tidak bisa dengan bebas menggunakan fitur yang ada pada sebuah objek.
- Waktu development lebih cepat: karena *reusable* otomatis dapat mempersingkat waktu pengembangan program.

Dan masih banyak lagi keunggulan OOP yang lainnya.

## Kekurangan Pemrograman Berbasis Objek

Selain kelebihan, pemrograman berbasis objek juga mempunyai kekurangan antara lain sebagai berikut:

- Kompleksitas program meningkat.
- *Learning curve* yang lumayan tinggi.
- Ukuran program jauh lebih besar.
- Pemakaian *memory* lebih besar.

## Kenapa Harus Belajar OOP

Setelah mengetahui kelebihan dan kekurangan OOP, sekarang kita sampai kepada kesimpulan kenapa kita harus belajar OOP. Tapi sebelum kita membahas kenapa kita harus belajar OOP, ada baiknya kita coba melihat realita yang ada. Di ekosistem PHP sekarang ini banyak sekali *framework* mulai dari yang kecil seperti Slim, Lumen, dan Zend Expressive hingga *framework* berskala besar seperti Zend Framework, dan Symfony. Bila kita cermati, hampir semua *framework* dibangun dengan menggunakan paradigma OOP sehingga dengan menguasai OOP, kita dapat memaksimalkan fitur yang ada dalam *framework* tersebut.

Dan berikut adalah beberapa alasan kenapa kita harus belajar OOP:

- OOP sangat cocok untuk pembuatan aplikasi berskala besar.
- OOP merupakan gerbang untuk menguasai *framework* dengan maksimal.
- OOP dapat digunakan diberbagai bahasa pemrograman yang mendukung OOP tidak hanya pada PHP.
- Dengan OOP kita dapat menerapkan *design pattern* dengan lebih mudah. caracara

Dan masih banyak lagi alasan untuk belajar OOP.

## Fitur Dasar Yang Ada pada OOP

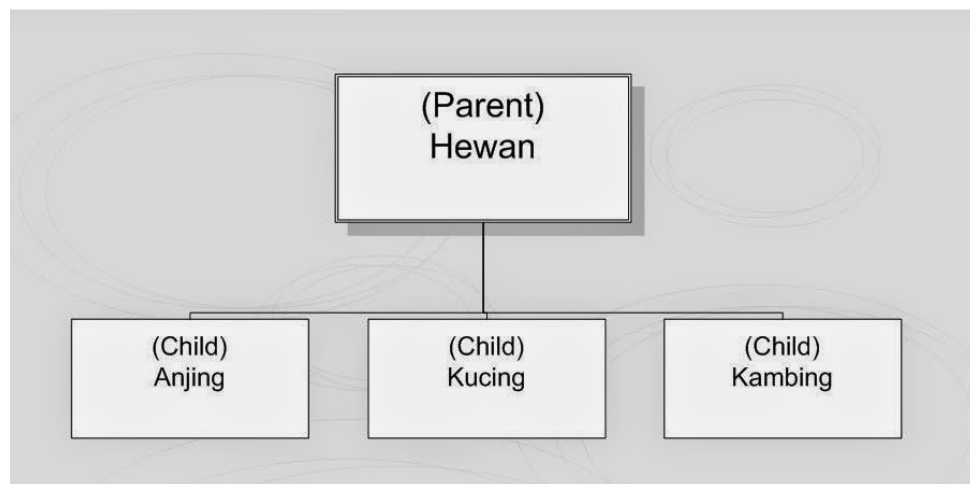
Berikut adalah beberapa fitur dasar yang harus ada dalam bahasa pemrograman yang mendukung konsep OOP.

- Enkapsulasi

Enkapsulasi adalah sebuah fitur yang menggabungkan antara fungsionalitas dan data untuk menyembunyikan informasi. Enkapsulasi memungkinkan kita menggunakan fungsi dari sebuah objek tanpa perlu mengetahui detail dari apa yang terjadi dalam fungsi tersebut. Enkapsulasi mengatur bagaimana kita menggunakan objek, fungsi atau atribut apa saja yang dapat digunakan dan yang tidak dapat digunakan.

- Pewarisan

Pewarisan adalah sebuah fitur yang memungkinkan kita menggunakan fitur dari suatu *class* tanpa perlu mendefinisikan ulang semua *method* dan *property* yang ada pada *class* tersebut. Pewarisan ini sangat bermanfaat jika kita ingin mempunyai sebuah *class* yang secara fitur mirip dengan *class* lain namun ada sebuah spesifikasi khusus yang spesifik dari *class* tersebut. Dalam OOP, *class* yang mewariskan sifat atau fitur disebut sebagai *parent class* sedangkan *class* yang diwarisi sifat atau fitur disebut *child class*. *Child class* memiliki semua fitur yang ada pada *parent class* ditambah dengan fitur spesifik miliknya sendiri.



Sumber: <http://www.adityawiraha.com>

- Polimorfisme

Polimorfisme adalah sebuah fitur yang erat kaitannya dengan fitur sebelumnya yaitu pewarisan. Fitur ini secara mudah adalah sebuah kemampuan dari objek untuk merespon atau mengolah secara berbeda terhadap input yang sama.

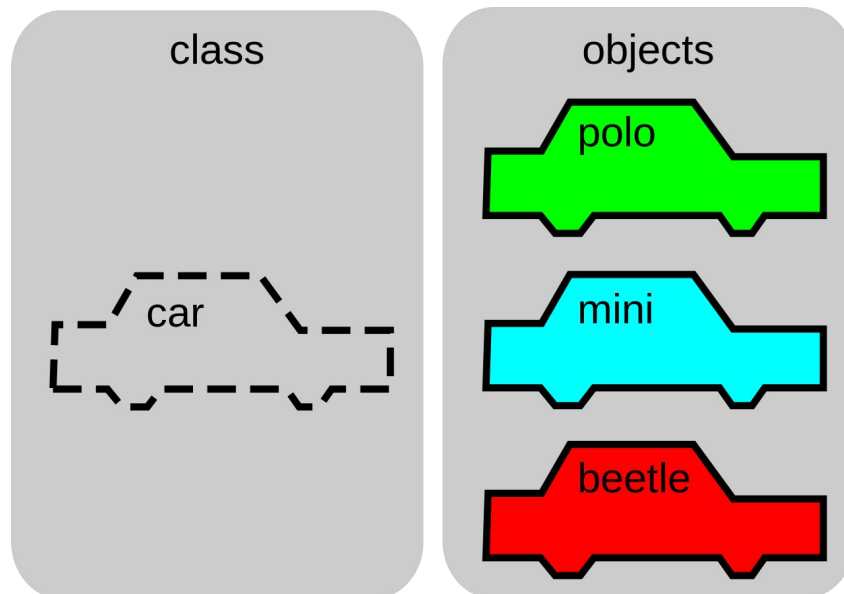
Saya agak kesulitan dalam menjelaskan bagaimana polimorfisme bekerja, namun nanti kita akan memahami hal tersebut ketika membahas tentang pewarisan, *abstract class*, *abstract method* dan *interface*. Untuk sekarang, Anda cukup pahami bahwa polimorfisme adalah salah satu fitur yang harus ada dalam OOP dimana sebuah *method* yang sama pada *class* yang berbeda akan memberikan *output* yang berbeda pula jika digunakan.

Itulah 3 konsep dasar yang harus ada pada bahasa pemrograman yang mendukung konsep OOP. Meski secara konsep sama, biasanya setiap bahasa pemrograman mempunyai cara implementasi yang berbeda dari fitur-fitur diatas.

## Class dan Object

### Apa itu Class

Secara mudah, *class* adalah cetakan atau *blueprint* dari objek. Didalam *class* terdapat *property* dan *method*. Dalam OOP, *class* merupakan kerangka dasar yang harus dibuat sebelum kita membuat *real object*. Untuk lebih jelas, perhatikan gambar berikut:



Sumber: [cloudfront.net](https://cloudfront.net)

Bila diperhatikan dari gambar diatas, dapat kita simpulkan bahwa sebuah *class* dapat memiliki banyak *object*. Dalam prakteknya, sebuah *class* memang dapat memiliki banyak *object* tergantung dari berapa kali kita melakukan instansiasi. Instansiasi adalah proses atau mekanisme pembuatan *object* dalam OOP. Pada OOP, instansiasi ditandai dengan penggunaan *keyword* `new` diikuti nama dari *class* yang akan kita instansiasi.

### Contoh Class pada PHP

Untuk membuat sebuah *class* pada PHP kita menggunakan *keyword* `class` diikuti nama dari *class* tersebut. Sebagai contoh kita akan membuat sebuah *class* `Mobil`, maka kita dapat membuatnya sebagai berikut:

```
<?php  
  
class Mobil  
{  
}
```

Kita juga dapat menambahkan *method* atau *function* pada *class* sehingga *class* `Mobil` akan menjadi seperti berikut:

```
<?php  
  
class Mobil  
{  
    public function jalan()  
    {  
        echo 'Mobil berjalan';  
    }  
}
```

Seperti yang sudah disebutkan pada pembahasan sebelumnya, bahwa selain memiliki *method*, *class* juga dapat memiliki *property*. Untuk membuat *property* pada *class* kita dapat menambah

Mudah sekali kan? kan langsung seperti berikut:

```
<?php

class Mobil
{
    public $roda;

    public function jalan()
    {
        echo 'Mobil berjalan';
    }
}
```

## Anatomi Class

Untuk lebih memahami tentang struktur dari *class*, mari kita coba membedah kembali *class* Mobil diatas.

```
class Mobil
{
}
```

**Keyword** `class` adalah sebuah *keyword* yang digunakan PHP untuk menandai bahwa sebuah *class* didefinisikan. **Keyword** `class` diikuti oleh nama *class* dalam hal ini adalah `Mobil` dan diikuti oleh kurung kurawal `{}`. Didalam kurung kurawal itulah segala *method* maupun *property* dari sebuah *class* didefinisikan.

```
public $roda;
```

**Keyword** `public` menandai bahwa sebuah *property* atau *method* memiliki visibilitas *public*. Di PHP terdapat 4 visibilitas yaitu *public*, *protected*, *private* dan *default* (tidak didefinisikan). Tentang visibilitas ini akan dibahas secara tersendiri pada pembahasan berikutnya. Setelah **keyword** `public` diikuti oleh *variable* `$roda` yang dalam konsep OOP disebut sebagai *property* yaitu sebuah *variable* yang dapat digunakan dalam lingkup *object*.

```
public function jalan()
{
    echo 'Mobil berjalan';
}
```

Pada contoh diatas, terdapat **keyword** `public` yang fungsinya telah dijelaskan sebelumnya. Setelah **keyword** `public` diikuti **keyword** `function` yang digunakan untuk menandai bahwa sebuah fungsi atau *method* didefinisikan. **Keyword** `function` diikuti oleh nama *method* yaitu `jalan` dan diikuti dengan tanda kurung `()` dilanjutkan dengan kurung kurawal `{}`.

Diantara kurung kurawal `{}` terdapat baris program yaitu `echo 'oleh _object_ Mobil berjalan';`, itulah yang disebut sebagai badan *method*. Didalam badan *method* kita dapat mendefinisikan *variable*, memanggil *method* lain atau bahwa memanggil *class* lain. Mudahnya, didalam *method* kita dapat mendefinisikan apa saja yang kita dapat definisikan diluar *method*. Namun yang perlu diingat adalah bahwa segala *variable* yang didefinisikan dalam *method* bersifat *local* sehingga hanya dapat digunakan pada lingkup *method* tersebut. Jika Anda ingin membuat *variable* yang dapat diakses dalam lingkup *object* maka gunakanlah *property*.

Diantara tanda kurung `()` sebenarnya kita dapat mendefinisikan parameter yaitu *variable* yang akan dikirimkan ketika *method* tersebut dipanggil. Untuk contoh penggunaan parameter akan lebih jelas lagi pada pembahasan-pembahasan selanjutnya. Untuk saat ini kita cukup tahu dulu bahwa sebuah *method* dapat memiliki parameter.

## Pembuatan *Object* (Instansiasi)

Seperti yang sudah dijelaskan sebelumnya bahwa sebuah *class* hanyalah *blueprint*, maka untuk membuatnya menjadi *real* kita perlu melakukan instansiasi. Proses instansiasi atau pembuatan *object* pada PHP ditandai dengan *keyword* `new` diikuti dengan nama *class*. Perhatikan contoh berikut:

```
<?php

class Mobil
{
    public $roda;

    public function jalan()
    {
        echo 'Mobil berjalan';
    }
}

$avanza = new Mobil();
```

Pada contoh diatas, kita menginstansiasi *class* `Mobil` dan memasukkannya kedalam *variable* `$avanza`. Selain cara diatas, kita juga dapat melakukan instansiasi tanpa menggunakan tanda kurung `()` setelah nama *class*.

```
<?php

class Mobil
{
    public $roda;

    public function jalan()
    {
        echo 'Mobil berjalan';
    }
}

$avanza = new Mobil;
```

Untuk memanggil *method* maupun *property* yang bersifat *public* dapat dilakukan sebagai berikut:

```
<?php

class Mobil
{
    public $roda;

    public function jalan()
    {
        echo 'Mobil berjalan';
    }
}

$avanza = new Mobil();
$avanza->roda = 4;

echo $avanza->jalan();
echo PHP_EOL;
echo $avanza->roda;
echo PHP_EOL;
```

Sehingga bila program diatas dijalankan maka akan menghasilkan *output* sebagai berikut:

php Mobil.php

Output:

Mobil berjalan

4

Bagaimana mudah sekali bukan? Mari kita masuki dunia OOP lebih dalam lagi.

## Property dan Method

### Apa itu Property

Seperti yang sudah disinggung pada pembahasan sebelumnya, *property* adalah sebuah *variable* dapat digunakan dalam lingkup *class* atau *object*. *Property* sering disebut juga sebagai segala sesuatu yang dimiliki oleh *class*. *Property* memiliki visibilitas serta dapat memiliki nilai *default*. Perhatikan kembali contoh *class Mobil* berikut:

```
<?php

class Mobil
{
    public $roda;
}
```

Pada contoh diatas, *property* `$roda` memiliki visibilitas *public* dan tanpa nilai *default* atau nilai awal. Untuk memberikan nilai awal pada *property* `$roda`, kita dapat melakukannya sebagai berikut:

```
<?php

class Mobil
{
    public $roda = 4;
}
```

Pada contoh diatas, kita memberikan nilai awal pada *property* `$roda` dengan nilai `4`. Maka ketika kita melakukan instansiasi, secara otomatis *property* `$roda` akan langsung mempunyai nilai `4` sebagaimana yang telah kita definisikan.

```
<?php

class Mobil
{
    public $roda = 4;
}

$avanza = new Mobil();
echo $avanza->roda;
echo PHP_EOL;
```

Maka ketika program dieksekusi akan menghasilkan *output* sebagai berikut:

```
php Mobil.php

Output:
4
```

### Apa itu Method

Jika *property* adalah segala sesuatu yang dimiliki oleh *class* atau *object*, maka *method* adalah segala sesuatu yang dapat dilakukan oleh *class* atau *object*. Sama seperti *property*, *method* juga memiliki visibilitas serta dapat memiliki parameter. Parameter dapat memiliki nilai awal atau *default value*. Bila parameter tidak memiliki *default value* maka parameter tersebut dianggap sebagai *mandatory parameter*. Perhatikan kembali *class Mobil* berikut:

```
<?php

class Mobil
{
    public function jalan()
    {
        echo 'Mobil berjalan';
    }
}
```

Pada contoh diatas, *method* `jalan()` memiliki visibilitas *public* dan tidak memiliki parameter. Untuk menambahkan parameter, kita dapat mendefinisikannya sebagai berikut:

```
<?php

class Mobil
{
    public function jalan($arah = 'depan')
    {
        echo 'Mobil berjalan ke arah '.$arah;
    }
}
```

Pada contoh diatas, kita menambahkan parameter `$arah` pada *method* `jalan()` serta menambahkan *default value* `depan` pada parameter tersebut. Kita dapat mengetes *code* diatas dengan cara sebagai berikut:

```
<?php

class Mobil
{
    public function jalan($arah = 'depan')
    {
        echo 'Mobil berjalan ke arah '.$arah;
    }
}

$avanza = new Mobil();

echo $avanza->jalan();
echo PHP_EOL;
echo $avanza->jalan('belakang');
echo PHP_EOL;
```

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

```
php Mobil.php

Output:
Mobil berjalan ke arah depan
Mobil berjalan ke arah belakang
```

Bila kita perhatikan, *default value* akan ditimpa oleh nilai yang kita masukkan. Dalam contoh diatas, kita memasukkan nilai `belakang` sehingga parameter `$arah` yang tadinya berisi `depan` berganti menjadi `belakang`. Dalam contoh *real*, *default value* dapat digunakan untuk mengeset nilai awal `$port` MySQL dari koneksi *database* yang secara *default* adalah `3306` serta `$host` yang biasanya memiliki nilai awal `localhost`. Perhatikan contoh berikut:

```
<?php

class Koneksi
{
    public function connect($username, $password, $host = 'localhost', $port = 3306)
    {
        //Logic koneksi
    }
}
```

Pada contoh diatas, parameter `$username` dan `$password` bersifat *mandatory* sehingga bila kita tidak memasukkan nilai tersebut ketika memanggil fungsi `connect()` maka akan terjadi *error*. Perhatikan code dibawah ini:

```
<?php

class Koneksi
{
    public function connect($username, $password, $host = 'localhost', $port = 3306)
    {
        //Logic koneksi
    }
}

$koneksi = new Koneksi();
$koneksi->connect();
```

Bila kita menjalankan program diatas maka akan terjadi *error* `PHP Fatal error: Uncaught ArgumentCountError: Too few arguments to function Koneksi::connect()`. Ini terjadi karena kita tidak memasukkan nilai `$username` dan `$password` sedangkan parameter tersebut bersifat *mandatory*. Untuk memperbaikinya, kita cukup memasukkan nilai sebagai berikut:

```
<?php

class Koneksi
{
    public function connect($username, $password, $host = 'localhost', $port = 3306)
    {
        //Logic koneksi
    }
}

$koneksi = new Koneksi();
$koneksi->connect('root', '');
```

Bila kita mengeksekusi program diatas maka tidak akan terjadi *error* dan tidak mengeluarkan *output* apapun karena memang kita hanya membuat sebuah *method* kosong.

## Urutan Parameter pada *Method*

Sebenarnya tidak ada aturan yang baku yang mengatur tentang urutan parameter pada sebuah *method*. Kita bisa saja mendefinisikan sesuka kita parameter-parameter dari sebuah *method*, namun tips berikut dapat dipertimbangkan untuk menghindari *error* dan memudahkan kita pada saat pemanggilan *method*.

- Parameter yang *mandatory* diletakkan didepan parameter yang memiliki *default value*.
- Penamaan parameter harus spesifik dan memiliki arti yang jelas serta menunjukkan kegunaan dari parameter tersebut.
- Gunakan *type hinting* (akan dibahas pada bab terpisah) pada parameter untuk menghindari kesalahan ketika memberikan nilai pada parameter.

Bila Anda perhatikan, apakah `class Koneksi` telah menerapkan tips diatas? Jawabannya adalah iya. Hanya saja, kita belum menerapkan *type hinting* karena memang belum dibahas.

# Visibilitas

## Apa itu Visibilitas

Visibilitas adalah salah satu fitur penting yang ada pada OOP. Fitur ini mengatur hak akses terhadap *property* maupun *method* dari sebuah *class*. Hak akses disini berbeda dengan hak akses pada aplikasi, hak akses disini adalah hak akses yang ada pada level bahasa pemrograman.

Visibilitas dalam OOP berperan penting dalam menjamin keamanan informasi yang terdapat pada *property* maupun *method*. Dengan fitur ini, *programmer* dapat membatasi dan mengatur *programmer* lainnya tentang bagaimana mengakses sebuah *property* atau *method* dari sebuah *class* atau fitur yang dibuatnya.

## Tingkatan Visibilitas pada PHP

Dalam bahasa PHP, Visibilitas dibedakan menjadi 4 yaitu *private*, *protected*, *public* dan *default* atau tidak didefinisikan. Berikut adalah penjelasan masing-masing visibilitas.

- *Private*

Visibilitas *private* adalah visibilitas paling rendah pada OOP. Sebuah *property* atau *method* yang diberikan visibilitas *private* maka *property* atau *method* tersebut hanya dapat diakses dari lingkup *class* dimana *property* atau *method* tersebut didefinisikan. Untuk memberikan visibilitas *private* pada *property* atau *method* kita dapat menggunakan keyword `private` didepan *property* atau *method*. Perhatikan contoh berikut:

```
<?php

class Mobil
{
    private $roda = 4;

    private function jalan()
    {
        echo 'Mobil berjalan';
    }
}

$avanza = new Mobil();

echo $avanza->jalan();
echo PHP_EOL;
echo $avanza->roda;
echo PHP_EOL;
```

Bila program diatas dijalankan, maka akan muncul *error* PHP Fatal error: Uncaught Error: Call to private method Mobil::jalan() from context . Hal ini terjadi karena kita mencoba mengakses fungsi `jalan()` yang memiliki visibilitas *private* diluar lingkup *class* yaitu dipanggil dari lingkup *object*. Hal yang sama juga akan terjadi pada *property* `$roda` dimana *property* tersebut juga memiliki visibilitas *private*.

Untuk mengakses *property* atau *method* dalam lingkup *class* digunakan keyword `$this` yang akan dibahas secara spesifik pada bab terpisah. Perhatikan contoh berikut:

```
<?php

class Mobil
{
    private $roda = 4;

    private function jalan()
    {
        echo 'Mobil berjalan';
    }

    public function jumlahRoda()
    {
        echo $this->roda;
    }
}

$avanza = new Mobil();

echo $avanza->jumlahRoda();
echo PHP_EOL;
```

Bila program diatas dieksekusi, maka akan muncul *output* 4 sebagaimana nilai dari *property* \$roda .

- *Protected*

Sebuah *property* atau *method* yang diberikan visibilitas *protected* maka *property* atau *method* tersebut dapat diakses dari lingkup *class* dimana *property* atau *method* tersebut didefinisikan serta turunan dari *class* tersebut. Untuk memberikan visibilitas *protected* pada *property* atau *method* kita dapat menggunakan *keyword* `protected` didepan *property* atau *method*. Perhatikan contoh berikut:

```
<?php

class Mobil
{
    private $roda = 4;

    protected function jalan()
    {
        echo 'Mobil berjalan';
    }

    public function jumlahRoda()
    {
        echo $this->roda;
    }
}

$avanza = new Mobil();

echo $avanza->jalan();
echo PHP_EOL;
```

Bila program diatas dijalankan, maka akan muncul *error* PHP Fatal error: Uncaught Error: Call to protected method Mobil::jalan() from context . Hal ini terjadi karena kita mencoba mengakses fungsi `jalan()` yang memiliki visibilitas *protected* diluar lingkup *class* yaitu dipanggil dari lingkup *object*.

- *Public*

Visibilitas *public* adalah visibilitas tertinggi pada OOP. Sebuah *property* atau *method* yang diberikan visibilitas *public* maka *property* atau *method* tersebut dapat diakses baik dari lingkup *class* maupun *object*. Untuk memberikan visibilitas *public* pada *property* atau *method* kita dapat menggunakan *keyword* `public` didepan *property* atau *method*. Mari kita melihat kembali contoh berikut:

```
<?php

class Mobil
{
    private $roda = 4;

    public function jumlahRoda()
    {
        echo $this->roda;
    }
}

$avanza = new Mobil();

echo $avanza->jumlahRoda();
echo PHP_EOL;
```

Method `jumlahRoda()` diberikan visibilitas *public* sehingga *method* tersebut dapat diakses dari luar *class* yaitu dalam lingkup *object*.

- *Default* atau tidak didefinisikan

Bila kita tidak mendefinisikan visibilitas pada *property* atau *method* secara eksplisit, maka PHP secara *default* akan memberikan visibilitas pada *property* atau *method* tersebut dengan visibilitas *public*. Sehingga `public function jumlahRoda()` sama dengan `function jumlahRoda()` yaitu sama-sama memiliki visibilitas *public*.

## Tips Visibilitas

Untuk menghindari kebingungan dan menjaga konsistensi *code* maka ada baiknya Anda menerapkan tips berikut:

- Definisikan visibilitas secara eksplisit.
- Gunakan visibilitas *private* atau *protected* pada *property*.
- Hindari penggunaan visibilitas *public* pada *property*, gunakan hanya jika memang benar-benar diperlukan.
- Batasi visibilitas pada *method* jika memungkinkan.