

# Object Oriented





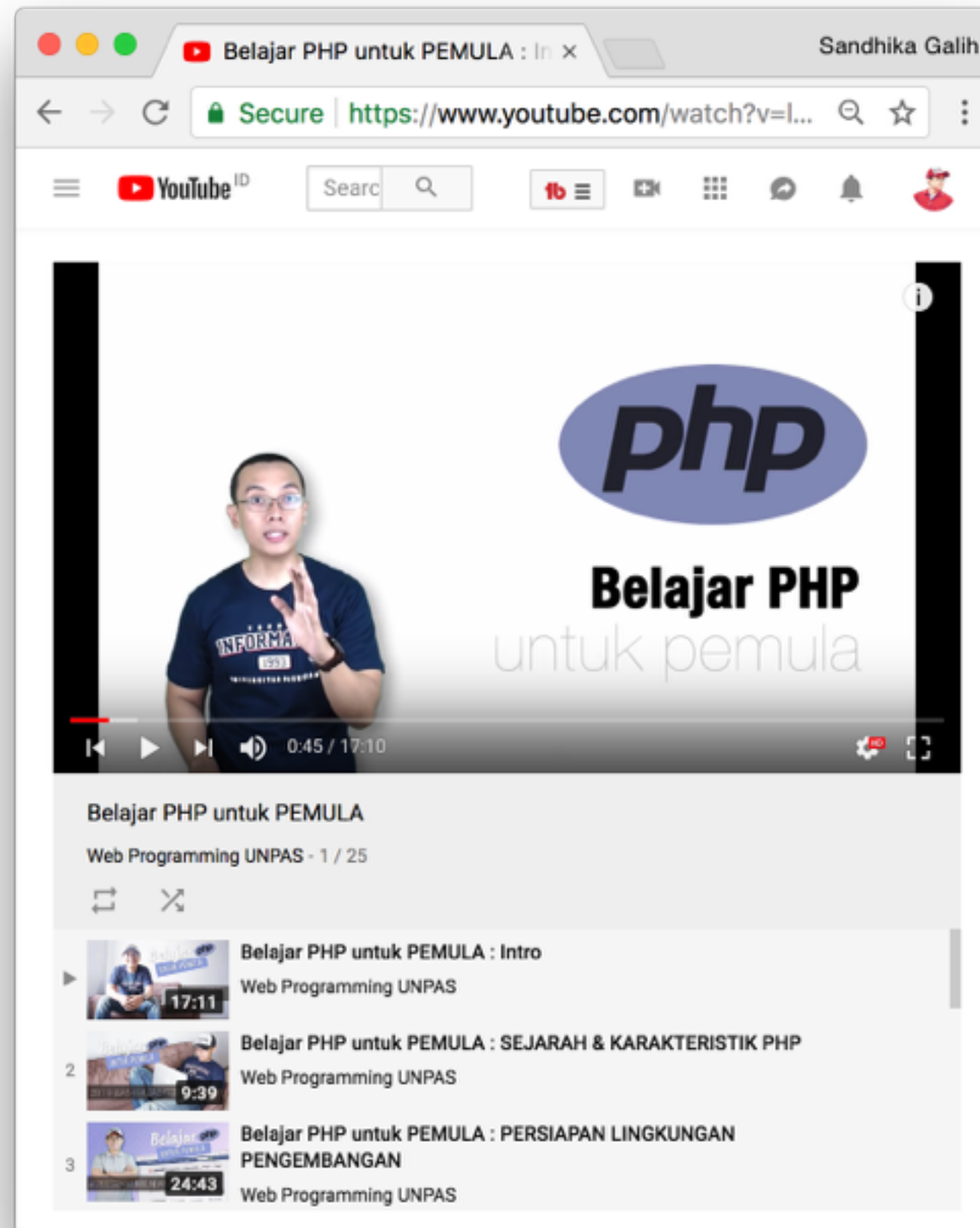
Berorientasi Object

# **Object Oriented** Programming?

# **Procedural** Programming?

=

Kuliah Pemrograman Web



<https://www.youtube.com/playlist?list=PLFIM0718LjIUqXfmEIBE3-uzERZPh3vp6>

# Procedural Programming

- Instruksi dilakukan langkah demi langkah
- Memecah program menjadi bagian-bagian kecil
- Disebut prosedur, subroutine atau function
- Linear / Top-to-Bottom
- Fortran, ALGOL, COBOL, Pascal, C, PHP, Javascript

# Kelebihan Procedural Programming

- To-the-point
- Simplicity & kemudahan implementasi  
(untuk compiler & interpreter)
- Mudah ditelusuri
- Membutuhkan lebih sedikit memory  
(dibandingkan dengan OOP)



# **Object Oriented** Programming?

# Object Oriented Programming

- Menyusun semua kode program dan struktur data sebagai objek
- Objek adalah unit dasar dari program
- Objek menyimpan data dan perilaku
- Objek bisa saling berinteraksi
- Java, Ruby, Python, C++, Javascript, PHP5

# Kelebihan Object Oriented Programming

- Representasi dunia nyata
- Enkapsulasi & Abstraksi Data
- Reusability
- Skalabilitas & Ekstensibilitas
- Kemudahan pengelolaan
- Kolaborasi
- Digunakan oleh framework

Konsep OOP  
**pada PHP ?**

# Basic

- Class & Object
- Property & Method
- Constructor
- Object Type
- Inheritance
- Visibility / Access Modifier
- Setter & Getter
- Static Method

# Advanced

- Abstract & Interface
- Interceptor
- Object Cloning
- Callbacks & Closures
- Namespaces & Autoloading
- ...

apa yang harus  
**disiapkan** ?

- Code Editor
- Web Server
- Web Browser
- Database Server



are you **ready** ?

# **Class & Object**

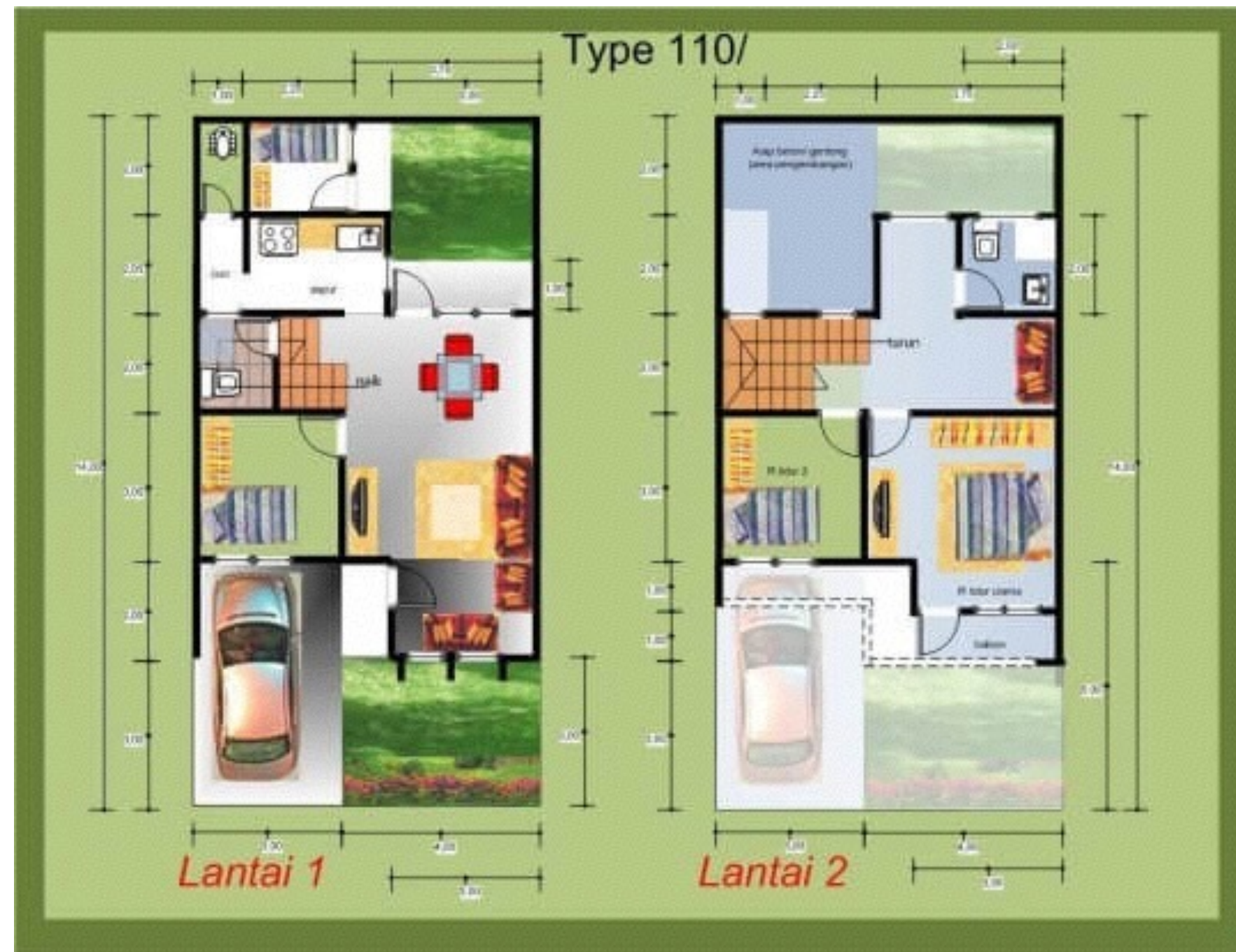
Type 110/









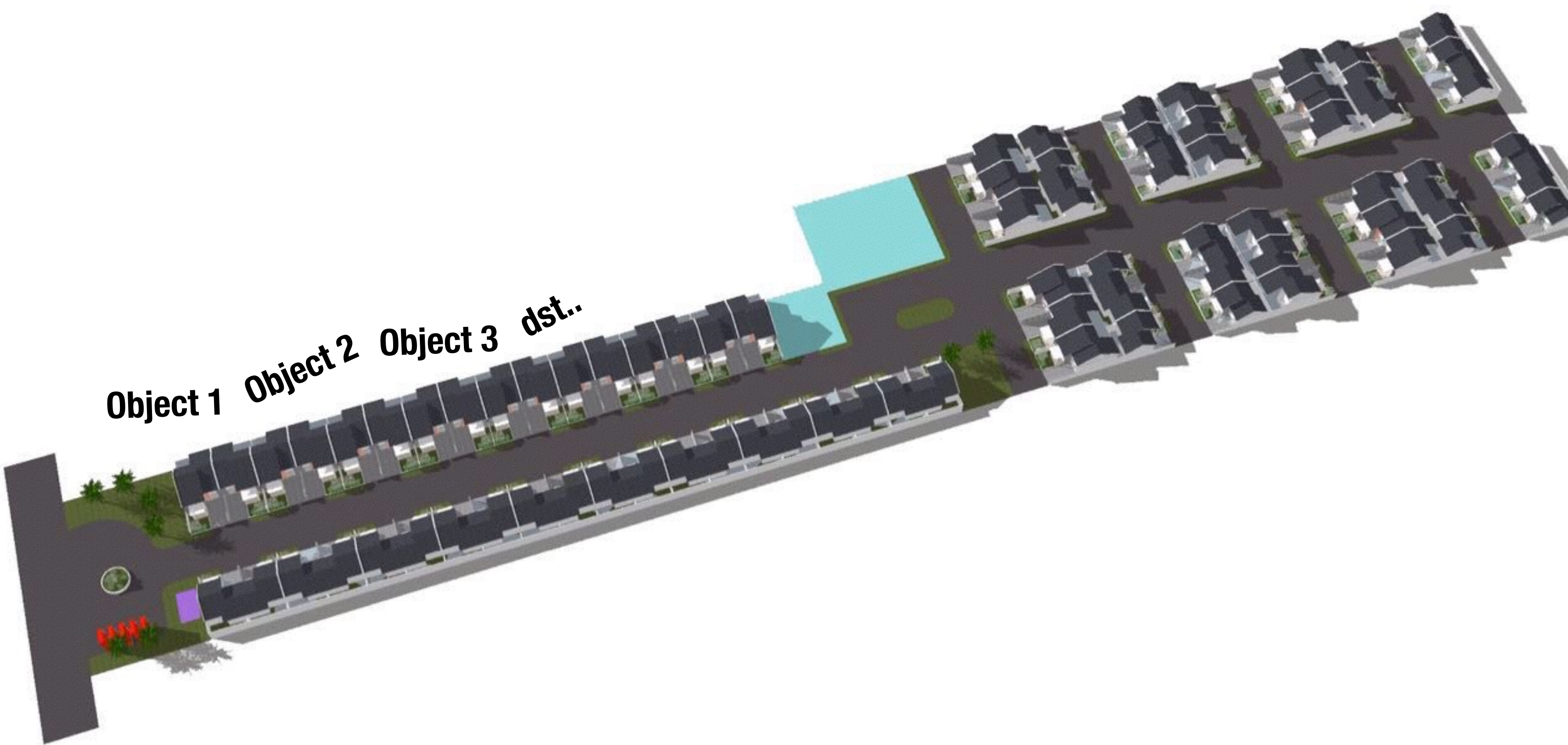


**Class**





**Object**



**Object 1   Object 2   Object 3   dst..**



# Class

- Blueprint / Template untuk membuat *instance* dari object
- Class mendefinisikan Object
- Menyimpan *data* dan *perilaku* yang disebut dengan *property* dan *method*

# Membuat Class

- Diawali dengan menuliskan keyword **class**, diikuti nama dan dibatasi dengan **{ }** untuk menyimpan property dan method
- Aturan penamaan class sama seperti variable

# Membuat Class

```
<?php
```

```
class Coba {
```

```
}
```

```
?>
```

# Membuat Class

```
<?php

class Coba {
    public $a; // property

    // method
    public function b() {

    }
}

?>
```

# Object

- *Instance* yang didefinisikan oleh Class
- Banyak object dapat dibuat menggunakan satu class
- Object dibuat dengan menggunakan keyword **new**

# Object

```
<?php

class Coba {

}

// membuat object instance dari class
$a = new Coba();
$b = new Coba();

?>
```

# Property & Method

# Property

- Merepresentasikan data / keadaan dari sebuah object
- Variabel yang ada di dalam object (*member variable*)
- Sama seperti variable di dalam PHP, ditambah dengan *visibility* di depannya



# Method

- Merepresentasikan perilaku dari sebuah object
- Function yang ada di dalam object
- Sama seperti function di dalam PHP, ditambah dengan *visibility* di depannya



## Mobil

### Property

- nama
- merk
- warna
- kecepatanMaksimal
- jumlahPenumpang

### Method

- tambahKecepatan
- kurangiKecepatan
- gantiTransmisi
- belokKiri
- belokKanan

# Class Mobil

```
<?php

class Mobil {
    public $nama,
    public $merk,
    public $warna,
    public $kecepatanMaksimal,
    public $jumlahPenumpang;

    public function tambahKecepatan() {

    }

    public function kurangiKecepatan() {

    }

    public function gantiTransmisi() {

    }
}
```

# Constructor

# Constructor

- Method yang otomatis dijalankan ketika sebuah object dibuat / class di instansiasi
- Dapat menerima parameter

# Constructor

```
class Produk {  
    public $judul,  
        $penulis,  
        $penerbit,  
        $harga;  
  
    public function __construct( $judul = "judul", $penulis = "penulis",  
        $penerbit = "penerbit", $harga = 0 ) {  
        $this->judul = $judul;  
        $this->penulis = $penulis;  
        $this->penerbit = $penerbit;  
        $this->harga = $harga;  
    }  
  
    public function getLabel() {  
        return "$this->penulis, $this->penerbit";  
    }  
}
```

# Inheritance

(Pewarisan)

# Inheritance

- Menciptakan hierarki antar kelas (***Parent & Child***)
- ***Child Class***, mewarisi semua properti dan method dari *parent*-nya (yang visible)
- ***Child Class***, *memperluas (extends)* fungsionalitas dari *parent*-nya





## Mobil

### Property

- nama
- merk
- warna
- kecepatanMaksimal
- jumlahPenumpang

### Method

- tambahKecepatan
- kurangiKecepatan
- gantiTransmisi
- belokKiri
- belokKanan



## Mobil

### Property

- nama
- merk
- warna
- kecepatanMaksimal
- jumlahPenumpang

### Method

- tambahKecepatan
- kurangiKecepatan
- gantiTransmisi
- belokKiri
- belokKanan



## Mobil Sport

### Property

- turbo

### Method

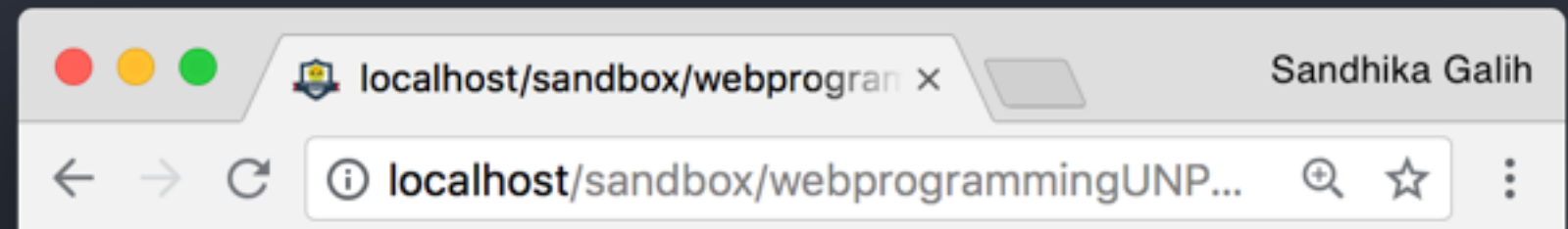
- jalankanTurbo

# Inheritance

```
class Mobil {  
    public $nama, $merk, $warna,  
        $kecepatanMaksimal,  
        $jumlahPenumpang;  
  
    public function tambahKecepatan() {  
        return "Kecepatan bertambah!";  
    }  
}
```

```
class MobilSport extends Mobil {  
    public $turbo = false;  
  
    public function jalankanTurbo() {  
        $this->turbo = true;  
        return "Turbo dijalankan!";  
    }  
}
```

```
$mobil1 = new MobilSport();  
echo $mobil1->tambahKecepatan();  
echo "<br>";  
echo $mobil1->jalankanTurbo();
```



Kecepatan bertambah!  
Turbo dijalankan!

# **Visibility**

( Access Modifier )

# Visibility

- Konsep yang digunakan untuk mengatur akses dari **property** dan **method** pada sebuah objek
- Ada 3 keyword visibility : **public**, **protected**, dan **private**

# Visibility

- **public**, dapat digunakan di mana saja, bahkan di luar kelas
- **protected**, hanya dapat digunakan di dalam sebuah kelas beserta turunannya
- **private**, hanya dapat digunakan di dalam sebuah kelas tertentu saja



# Kenapa ?

- Hanya memperlihatkan aspek dari class yang dibutuhkan oleh 'client'
- Menentukan kebutuhan yang jelas untuk object
- Memberikan kendali pada kode untuk menghindari 'bug'



# Setter & Getter

( Accessor Method )

# Visibility

- **public**, dapat digunakan di mana saja, bahkan di luar kelas
- **protected**, hanya dapat digunakan di dalam sebuah kelas beserta turunannya
- **private**, hanya dapat digunakan di dalam sebuah kelas tertentu saja

overloading

**\_\_set()** & **\_\_get()**

# Static Keyword

```
<?php
```

```
class Mobil {  
    public $nama,  
    public $merk,  
    public $warna,  
    public $kecepatanMaksimal,  
    public $jumlahPenumpang;  
  
    public function tambahKecepatan() {  
  
    }  
  
    public function kurangiKecepatan() {  
  
    }  
  
    public function gantiTransmisi() {  
  
    }  
}  
  
$avanza = new Mobil();
```

**Class**

**Object**

```
class Mahasiswa {  
    private $nama, $umur, $ipk;  
  
    public function __construct( $nama, $umur, $ipk ) {  
        $this->nama = $nama;  
        $this->umur = $umur;  
        $this->ipk = $ipk;  
    }  
  
    public function getNama() {  
        return $this->nama;  
    }  
}
```

**Class**

```
$mhs1 = new Mahasiswa('Sandhika', 20, 2.5);
```

**Object**

```
echo $mhs1->getNama();
```

**class** merupakan template  
dari **object**

kita bisa mengakses **property** dan **method**  
dalam konteks **class**



untuk apa **static keyword** ?

# Static Keyword

- Member yang terikat dengan class, bukan dengan object
- Nilai static akan selalu tetap meskipun object di-instansiasi berulang kali
- Membuat kode menjadi 'procedural'
- Biasanya digunakan untuk membuat fungsi bantuan / helper
- Atau digunakan di class-class utility pada Framework

# Constant

sebuah **identifier** untuk  
menyimpan **nilai**

variable ?

**nilai**-nya

tidak dapat berubah

**define()**

**const**

# define()

## const

*bisa disimpan di dalam class*





Magic **Constant**

# Magic Constant

- `__LINE__`
- `__FILE__`
- `__DIR__`
- `__FUNCTION__`
- `__CLASS__`
- `__TRAIT__`
- `__METHOD__`
- `__NAMESPACE__`

# **Abstract Class**

# Abstract Class

- Sebuah kelas yang **tidak dapat di-instansiasi**
- Kelas '*abstrak*'
- Mendefinisikan *interface* untuk kelas lain yang menjadi turunannya
- Berperan sebagai 'kerangka dasar' untuk kelas turunannya
- Memiliki minimal 1 **method abstrak**
- Digunakan dalam '*pewarisan*' / *inheritance* untuk '**memaksakan**' implementasi method abstrak yang sama untuk semua kelas turunannya

Contoh Kasus

```
class Buah {  
    private $warna;  
  
    public function makan() {  
        // kunyah  
        // nyam..nyam..nyam  
    }  
  
    public function setWarna($warna) {  
        $this->warna = $warna;  
    }  
}
```

```
class Apel extends Buah {  
    public function makan() {  
        //kunyah  
        //sampai bagian tengah  
    }  
}
```

```
class Jeruk extends Buah {  
    public function makan() {  
        // kupas  
        // kunyah  
    }  
}
```

```
$apel = new Apel();  
$apel->makan();
```



```
$buah = new Buah();  
$buah->makan();
```



Membuat Kelas Abstrak



```
abstract class Buah {
```

```
}
```

kelas abstrak

```
abstract class Buah {  
    private $warna;  
  
    abstract public function makan();  
  
    public function setWarna($warna) {  
        $this->warna = $warna;  
    }  
}
```

- method abstrak
- hanya interface-nya saja
- implementasinya, ada di kelas turunannya

Kelas Abstrak

# Abstract Class (1)

- Sebuah kelas yang **tidak dapat di-instansiasi**
- Kelas '*abstrak*'
- Mendefinisikan *interface* untuk kelas lain yang menjadi turunannya
- Berperan sebagai 'kerangka dasar' untuk kelas turunannya
- Biasanya memiliki minimal 1 **method abstrak**
- Digunakan dalam '*pewarisan*' / *inheritance* untuk '**memaksakan**' implementasi method abstrak yang sama untuk semua kelas turunannya

## **Abstract Class (2)**

- Semua kelas turunan, harus mengimplementasikan method abstrak yang ada di kelas abstraknya
- Kelas abstrak boleh memiliki property / method reguler
- Kelas abstrak boleh memiliki property / static method

# Contoh Kelas Abstrak

- `class Mobil Extends Kendaraan`
- `class Laptop Extends Komputer`
- `class Email Extends Komunikasi`
- ...

Kenapa kelas abstrak?

# Kenapa menggunakan kelas abstrak?

- Merepresentasikan ide atau konsep dasar
- *“Composition over Inheritance”*
- Salah satu cara menerapkan Polimorphism
- Sentralisasi logic
- Mempermudah pengerjaan tim



# Interface

# Interface

- **Kelas Abstrak** yang sama sekali tidak memiliki implementasi
- **Murni** merupakan template untuk kelas turunannya
- **Tidak boleh** memiliki property, hanya deklarasi method saja

masih ingat kelas abstrak?

kelas abstrak

```
abstract class Buah {  
    private $warna;  
  
    abstract public function makan();  
  
    public function setWarna($warna) {  
        $this->warna = $warna;  
    }  
}
```

- method abstrak
- hanya interface-nya saja
- implementasinya, ada di kelas turunannya

```
abstract class Buah {  
    private $warna;  
  
    abstract public function makan();  
  
    public function setWarna($warna) {  
        $this->warna = $warna;  
    }  
}
```

```
class Apel extends Buah {  
    public function makan() {  
        //kunyah  
        //sampai bagian tengah  
    }  
}
```


```
class Jeruk extends Buah {  
    public function makan() {  
        // kupas  
        // kunyah  
    }  
}
```

Menggunakan Interface

```
interface Buah {  
    public function makan();  
    public function setWarna($warna);  
}
```

```
interface Buah {  
    public function makan();  
    public function setWarna($warna);  
}
```

keyword 'implements'



```
class Apel implements Buah {  
    protected $warna;  
    public function makan() {  
        //kunyah  
        //sampai bagian tengah  
    }  
    public function setWarna($warna) {  
        $this->warna = $warna;  
    }  
}
```

```
class Jeruk implements Buah {  
    protected $warna;  
    public function makan() {  
        // kupas  
        // kunyah  
    }  
    public function setWarna($warna) {  
        $this->warna = $warna;  
    }  
}
```



Interface

# Interface (1)

- **Kelas Abstrak** yang sama sekali tidak memiliki implementasi
- **Murni** merupakan template untuk kelas turunannya
- **Tidak boleh** memiliki property, hanya deklarasi method saja
- Semua method harus dideklarasikan dengan visibility **public**
- Boleh mendeklarasikan **\_\_construct()**
- Satu kelas boleh mengimplementasikan **banyak interface**

```
interface Buah {  
    public function makan();  
    public function setWarna($warna);  
}
```

```
class Apel implements Buah {  
    protected $warna;  
    public function makan() {  
        //kunyah  
        //sampai bagian tengah  
    }  
    public function setWarna($warna) {  
        $this->warna = $warna;  
    }  
}
```

```
class Jeruk implements Buah {  
    protected $warna;  
    public function makan() {  
        // kupas  
        // kunyah  
    }  
    public function setWarna($warna) {  
        $this->warna = $warna;  
    }  
}
```

```
interface Buah {  
    public function makan();  
    public function setWarna($warna);  
}
```

```
interface Benda {  
    public function setUkuran($ukuran);  
}
```

```
class Apel implements Buah {  
    protected $warna;  
    public function makan() {  
        //kunyah  
        //sampai bagian tengah  
    }  
    public function setWarna($warna) {  
        $this->warna = $warna;  
    }  
}
```

```
class Jeruk implements Buah {  
    protected $warna;  
    public function makan() {  
        // kupas  
        // kunyah  
    }  
    public function setWarna($warna) {  
        $this->warna = $warna;  
    }  
}
```

```
interface Buah {  
    public function makan();  
    public function setWarna($warna);  
}
```

```
interface Benda {  
    public function setUkuran($ukuran);  
}
```

```
class Apel implements Buah, Benda {  
    protected $warna;  
    protected $ukuran;  
    public function makan() {  
        //kunyah  
        //sampai bagian tengah  
    }  
    public function setWarna($warna) {  
        $this->warna = $warna;  
    }  
    public function setukuran($ukuran) {  
        $this->ukuran = $ukuran;  
    }  
}
```

```
class Jeruk implements Buah {  
    protected $warna;  
    public function makan() {  
        // kupas  
        // kunyah  
    }  
    public function setWarna($warna) {  
        $this->warna = $warna;  
    }  
}
```

## Interface (2)

- Dengan menggunakan type-hinting dapat melakukan **‘Dependency Injection’**
- Pada akhirnya akan mencapai **Polymorphism**

# **Autoloading**

memanggil class (file) tanpa harus  
menggunakan **require**



1 class ditulis dalam 1 file

**require?**

Studi Kasus

```
spl_autoload_register();
```

# Namespace

sebuah cara untuk mengelompokkan program  
ke dalam sebuah **package** tersendiri

# Encapsulation

why ?



PHP tidak mengijinkan kita untuk memiliki  
2 Class dengan nama yang sama

tidak masalah jika kita bekerja sendiri.

bagaimana jika tim?

bagaimana jika kita menggunakan script /  
library dari pihak ke-tiga?

Studi Kasus

```
namespace Vendor\Namespace\SubNamespace;
```