MODUL PRAKTIKUM

PEMROGRAMAN BERORIENTASI OBJEK

Disusun Oleh:

Elsa Elvira Awalia, M.Kom

Yusuf Eka Wicaksana, M.Kom

PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS ILMU KOMPUTER
UNIVERSITAS BUANA PERJUANGAN KARAWANG
2022/2023

DAFTAR ISI

BAB I
MEMAHAMI KONSEP

"Orang yang memahami konsep dari suatu ilmu, berarti sudah memahami setengah dari ilmu tersebut"

1.1. Programmer dan Segala Kerumitannya

Pada dasarnya ketika membuat suatu software, kita dibebaskan untuk membuatnya dengan berbagai cara sesuai selera. Namun dalam suatu kasus, kita pasti membutuhkan bantuan orang lain dalam mengembangkan software yang mana "gaya coding" kita dengan orang lain pasti berbeda. Hal tersebut sangat menyulitkan ketika kita akan berkolaborasi dengan rekan satu tim yang memiliki perbedaan "gaya coding".

Atas dasar perbedaan "gaya coding" tersebut, para programmer pada akhirnya membuat sebuah konvensi / perjanjian dimana mereka menentukan "gaya coding" sesuai kesepakatan. Dimana ditinjau dari efisiensi kode, kemudahan pemahaman kode, performa, dan sebagainya. Hasil konvensi tersebut pada akhirnya menghasilkan sebuah kode yang terstruktur yang bernama *framework.*

Framework merupakan suatu kode yang terstruktur dimana segala hal yang dibutuhkan dalam pengembangan software sudah tersedia, sehingga kita bisa berfokus dalam pengerjaan software nya saja. Pada *framework*, terdapat paradigma pemrograman yang biasanya dikenal dengan *object oriented programming* (pemrograman berorientasi objek). Sebenarnya banyak sekali paradigma pemrograman yang lain seperti *functional programming*, *procedural*, *event-driven*, dan salah satunya yakni *object oriented programming*. Namun *object oriented programming* merupakan yang paling populer.

1.2. Pengertian *Object Oriented Programming*

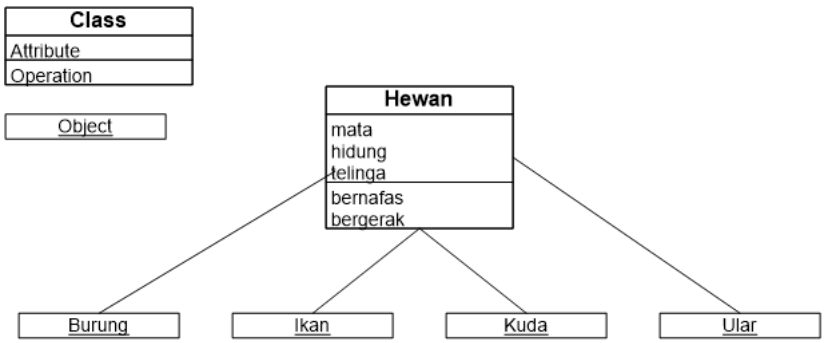
Object oriented programming (mulai sekarang kita akan menyebutnya OOP/PBO) merupakan suatu paradigma pemrograman yang berlandaskan atas object dari suatu entitas. Secara sederhana OOP hanya berkutat antara *class* dan *object*.

- Class : blueprint / cetakan dari suatu objek
- Object : hasil dari blueprint / cetakan pada class

Mudahnya mari kita analogikan class dan object dengan contoh kasus kue.

Pada gambar diatas dapat dilihat bahwa kue yang dicetak merupakan hasil dari cetakan kue tersebut, yang mana dari bentuk serta ukuran bisa sama persis seperti cetakan kue. Maka dapat disimpulkan class adalah cetakan kue, sedangkan object adalah kue hasil cetakan. Untuk lebih jelasnya mari kita gunakan contoh kasus hewan.

Hewan bisa kita katakan sebagai suatu entitas yang bersifat umum, yang mana begitu banyak hewan – hewan yang ada didunia. Burung, ikan, kuda, ular, dan sebagainya sama-sama termasuk hewan. Untuk beberapa hal mereka memiliki sifat yang serupa seperti bernafas, bergerak dan sebagainya. Maka dari itu kita dapat membuat class dengan nama hewan dengan object burung, ikan, kuda, dan juga ular. Untuk lebih jelasnya dapat dilihat pada gambar dibawah ini.



BAB II
PERSIAPAN PRAKTIKUM

2.1. Persiapan Diri

Untuk dapat memulai praktikum, maka mahasiswa diharuskan untuk mengikuti tata tertib selama praktikum berlangsung. Adapun tata tertib praktikum antara lain sebagai berikut:

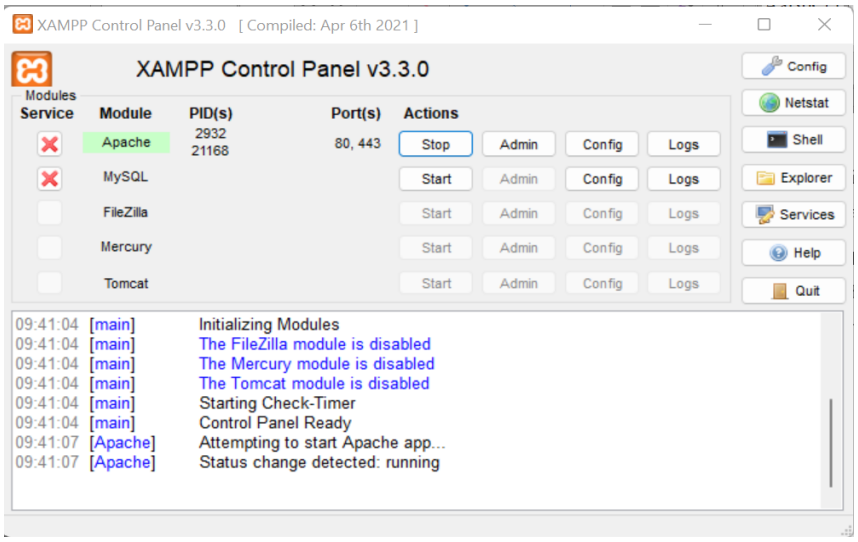
1. Memakai pakaian sopan dan rapi, tidak boleh memakai kaos
2. Tidak boleh makan dan minum selama praktikum berlangsung
3. Memakai jas almamater ketika praktikum (jika memakai kaos)
4. Bersikap sopan, santun dan tidak arogan
5. Konsekuensi keterlambatan maksimal 15 menit setelah praktikum dimulai dan tidak menerima penjelasan kembali ketika terlambat
6. Melebih batas keterlambatan diharap ikut kelas lain

2.2. Persiapan Alat

Untuk dapat mengikuti praktikum, para mahasiswa diwajibkan untuk menginstall beberapa aplikasi berikut di laptop atau komputer:

1. Web Server support PHP (XAMPP / WAMPP / LAMPP)
2. Text Editor (Visual Studio Code / PHP Storm / Notepad / Notepad++)
3. Browser (Google Chrome / Mozilla Firefox / Edge)

Kemudian jalankan web server (contoh XAMPP) di laptop dan klik tombol start pada baris Apache untuk menyalakan web server nya seperti pada gambar berikut

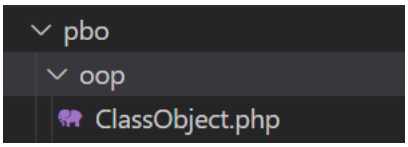


BAB III
MEMULAI PRAKTIKUM

3.1. Class dan Object

Seperti yang telah kita bahas pada bab 1, OOP hanya berkutat antara *class* dan *object*. Lalu bagaimana cara membuat *class* dan *object* di bahasa pemrograman PHP? Berikut penjelasannya.

1. Siapkan satu folder dimana kita akan menyimpan kode kita di folder tersebut. Sebagai contoh kita akan menyiapkan folder dengan nama “pbo” dan didalamnya terdapat folder lagi dengan nama “oop” seperti berikut dan simpan di folder htdocs hasil install web server:



2. Selanjutnya buat file dengan nama ClassObject.php didalam folder oop.
3. Lalu isi file ClassObject.php dengan kode seperti berikut

```
<?php
class Hewan {
    //isi dari class ada diantara kurung kurawal
}
```

pada kode diatas untuk membuat suatu class maka kita gunakan perintah `class`, diikuti dengan nama class tersebut yakni `Hewan`. Kemudian kita tambahkan tanda kurung kurawal/kurung keriting buka ({) dan tutup (}). Diantara kurung kurawal merupakan isi dari class yang akan kita buat.

Selanjutnya kita dapat mengakses class tersebut dengan membuat object, untuk membuat object dapat dilakukan dalam satu file yang sama ataupun berbeda. Untuk pembahasan berbeda file kita akan bahas di pembahasan berikutnya.

4. Buat sebuah object dengan nama ‘kelinci’ dan ‘kucing’ seperti pada kode berikut:

```
<?php
class Hewan {
    //isi disini
}
```

```
//buat object
$kelinci = new Hewan();
$kucing = new Hewan();
```

Untuk membuat object kita gunakan perintah `new` diikuti nama class, dalam kasus kasus ini seperti `$kelinci = new Hewan();`

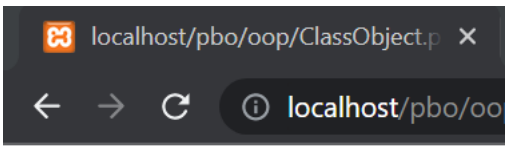
5. Selanjutnya gunakan perintah `print_r` untuk mencetak hasil dari object yang dibuat seperti berikut

```
<?php
class Hewan {
    //isi disini
}
//buat object
$kelinci = new Hewan();
$kucing = new Hewan();
print_r($kelinci);
print_r($kucing);
```

untuk melakukan debugging atau melihat hasil, maka kita dapat menggunakan perintah

- echo
- print_r, atau
- var_dump (lebih detail)

1. Terakhir akses file ClassObject.php di browser dengan mengetik localhost/pbo/oop/ClassObject.php



Hewan Object ( ) Hewan Object ( )

3.2. *Property* dan *Constant*

Mudahnya *property* merupakan variable biasa yang ada dalam suatu class, sedangkan *constant* merupakan variable namun tidak bisa di *assign* / diubah nilanya dan bersifat tetap tidak bisa diubah. Untuk membuat *property* dan *constant* ikuti langkah-langkah berikut:

1. Buat file dengan nama Property.php didalam folder oop.

2. Lalu isi file Property.php dengan kode seperti berikut

```
<?php
class Hewan {
    //ini property/variable
    public $mata;
    public $hidung;
    public $telinga;
}
```

Didalam `class Hewan` kita tambahkan property mata, hidung, telinga dengan disisipi simbol dolar ($) dan ditambahkan public di depan property tersebut. Jika kita hapus kode `public` maka akan muncul error seperti berikut.

Parse error: syntax error, unexpected '$mata' (T_VARIABLE), expecting function (T_FUNCTION) or const (T_CONST) in **C:\xampp\htdocs\pbo\oop\Property.php** on line **6**

Hal ini dikarenakan variable mata tidak diketahui apakah sebuah konstanta atau suatu method.

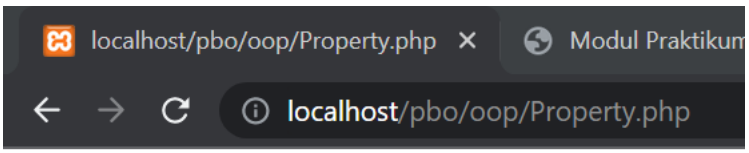
3. Selanjutnya kita buat object dengan nama ‘kelinci’ dan meng-assign tiap property

```
<?php
class Hewan {
    //ini property/variable
    public $mata;
    public $hidung;
    public $telinga;
}
//buat object
$kelinci = new Hewan();
//kita tambahkan isi dari tiap property (assign)
$kelinci->mata = 'bulat';
$kelinci->hidung = 'kecil';
```

```
$kelinci->telinga = 'panjang';
```

4. Terakhir kita tambahkan perintah `echo`, `print_r`, dan `var_dump` untuk melihat hasilnya dan tampilkan di browser (sama seperti menampilkan file ClassObject.php diatas)

```
<?php
class Hewan {
    //ini property/variable
    public $mata;
    public $hidung;
    public $telinga;
}
//buat object
$kelinci = new Hewan();

//kita tambahkan isi dari tiap property
$kelinci->mata = 'bulat';
$kelinci->hidung = 'kecil';
$kelinci->telinga = 'panjang';

//pakai echo
echo $kelinci->mata . PHP_EOL;

//pakai print_r
print_r($kelinci->hidung) . PHP_EOL;

//pakai var_dump
var_dump($kelinci->telinga) . PHP_EOL;
```

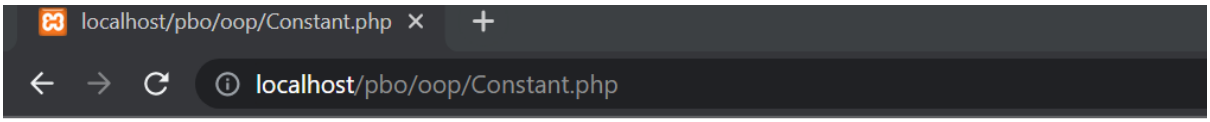


bulat kecilstring(7) "panjang"

Selanjutnya untuk *constant* kita akan buat seperti berikut

```
<?php
class Hewan {
    const BERNAFAS = 'semua hewan bisa bernafas';
}
```

```
echo Hewan::BERNAFAS . PHP_EOL;
print_r(Hewan::BERNAFAS) . PHP_EOL;
var_dump(Hewan::BERNAFAS) . PHP_EOL;
```



semua hewan bisa bernafas semua hewan bisa bernafasstring(25) "semua hewan bisa bernafas"

3.3. *Method* dan *Static Method*

Method merupakan salah satu bagian yang ada di dalam *class* selain *property* dan *constant*, *method* dapat diartikan sebagai hal apa saja yang bisa dilakukan *object* pada suatu *class*. Untuk lebih jelasnya ikuti kode berikut:

1. Buat file dengan nama Method.php didalam folder oop.
2. Lalu isi file Method.php dengan kode seperti berikut

```
<?php
class Hewan {
    function bergerak(){
        echo "aku dapat bergerak";
    }
}
```

Untuk membuat sebuah `method` maka kita harus menambahkan `function` kemudian diikuti dengan nama `method` dan beri tanda kurung buka-tutup (), selanjutnya tambahkan kurung kurawal buka-tutup {}. Isi dari `method` ada diantara kurung kurawal.

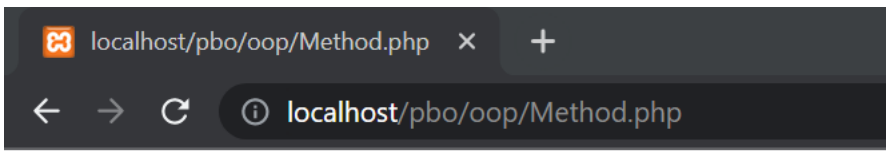
```
function bergerak(){
   //isi method
}
```

3. Selanjutnya kita buat object dengan nama ‘kucing’

```
<?php
class Hewan {
    function bergerak(){
        echo "aku dapat bergerak";
    }
}
$kucing = new Hewan();
```

4. Selanjutnya kita panggil method dari object yang kita buat dengan menggunakan anak panah seperti berikut dan tampilkan di browser

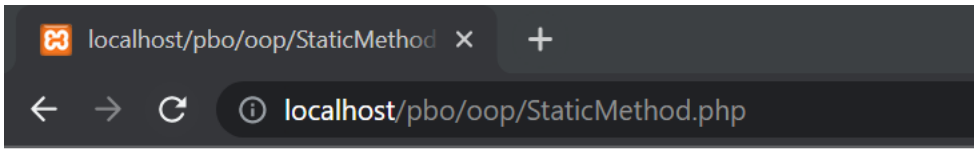
```
<?php
class Hewan {
    function bergerak(){
        echo "aku dapat bergerak";
    }
}
$kucing = new Hewan();
//panggil method
echo $kucing->bergerak() . PHP_EOL;
print_r($kucing->bergerak()) . PHP_EOL;
var_dump($kucing->bergerak()) . PHP_EOL;
```



aku dapat bergerak aku dapat bergerakaku dapat bergerakNULL

Untuk static method cukup tambahkan tanda titik dua sebanyak dua kali (::), tanpa harus membuat object

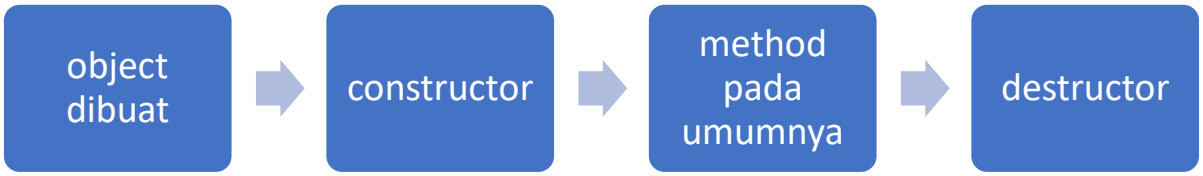
```php
<?php
class Hewan {
    static function bergerak(){
        echo "aku dapat bergerak";
    }
}
//panggil method langsung tanpa membuat object
echo Hewan::bergerak() . PHP_EOL;
print_r(Hewan::bergerak()) . PHP_EOL;
var_dump(Hewan::bergerak()) . PHP_EOL;
```



aku dapat bergerak aku dapat bergerakaku dapat bergerakNULL

3.4. *Constructor* dan *Destructor*

Dalam pemanggilan *object* dari suatu class, sebenarnya secara *default* kita sudah menjalankan *constructor*. *Constructor* merupakan suatu *method* yang akan otomatis berjalan ketika suatu *object* dibuat. Sedangkan *destructor* yakni sebaliknya, berjalan ketika suatu object sudah dibuat (opsional). Ilustrasinya yakni sebagai berikut.



Untuk lebih jelasnya mari ikuti instruksi berikut:

1. Buat file dengan nama ConstrutorDestructor.php didalam folder oop.
2. Lalu isi file ConstrutorDestructor.php dengan kode seperti berikut

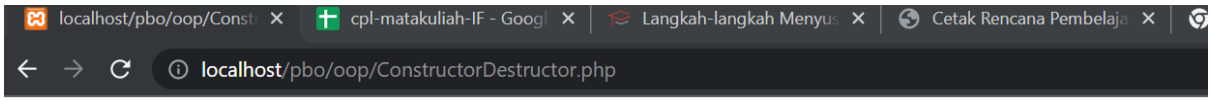
```
<?php
class Hewan {
    //ini constructor
    function __construct(){
        echo "aku dipanggil pertama kali lho" . PHP_EOL;
    }
    function bergerak(){
        echo "aku dapat bergerak";
    }
    //ini destructor
    function __destruct(){
        echo "aku dipanggil terakhir kali yaa" . PHP_EOL;
    }
}
```

Pada kode diatas kita tambahkan *method* dengan nama `__construct` dan `__destruct`. Untuk *constructor* dan *destructor* kita harus menggunakan nama *method* tersebut, jika tidak maka bukan dianggap *constructor* maupun *destructor*.

3. Selanjutnya kita buat *object* dan menampilkan *method* dengan nama bergerak(), serta tampilkan di browser.

```
<?php
class Hewan {
    //ini constructor
    function __construct(){
        echo "aku dipanggil pertama kali lho" . PHP_EOL;
    }
    function bergerak(){
        echo "aku dapat bergerak";
    }
```

```
    //ini destructor
    function __destruct(){
        echo "aku dipanggil terakhir kali yaa" . PHP_EOL;
    }
}

$kucing = new Hewan();
echo $kucing->bergerak() . PHP_EOL;
print_r($kucing->bergerak()) . PHP_EOL;
var_dump($kucing->bergerak()) . PHP_EOL;
```




```
aku dipanggil pertama kali lho aku dapat bergerak aku dapat bergerakaku dapat bergerakNULL aku dipanggil terakhir kali yaa
```

Seperti yang terlihat, walaupun kita memanggil method bergerak() namun isi dari constructor akan dipanggil pertama kali. Sedangkan isi dari destructor akan di tampilkan terakhir kali.

3.5. *Access* Modifier (Visibility)

Access modifier atau yang biasa dikenal dengan visibility merupakan salah satu cara yang bisa kita lakukan untuk memberikan hak akses pada setiap property ataupun method pada suatu class. Dalam OOP di PHP, access modifier dibagi menjadi 3 jenis antara lain:

No	Access Modifier	Ketersediaan Akses
1	Public	Dapat diakses oleh seluruh kode di projek kita, baik di satu file yang sama ataupun berbeda.
2	Protected	Hanya dapat diakses oleh class yang sama ataupun turunannya (inheritance)
3	Private	Hanya dapat diakses oleh kelas yang sama

Untuk lebih memudahkan pemahaman perhatikan diagram berikut

Public	Protected	Private
• Class yang sama • Class turunan • Class yang berbeda • Object	• Class yang sama • Class turunan	• Class yang sama

Berikut contoh kode dari access modifier

1. Buat file dengan nama Visibility.php didalam folder oop.
2. Lalu isi file Visibility.php dengan kode seperti berikut

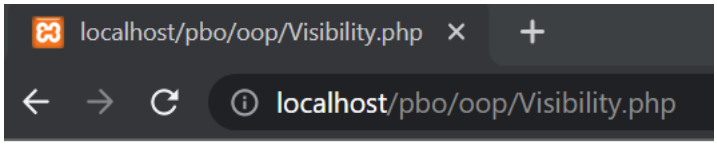
```
<?php
class Hewan {
    public $mata;
    protected $hidung;
    private $telinga;

    public function hidup(){
        return "aku hidup";
    }
    protected function berenang(){
        return 'aku berenang';
    }
    private function bertelur(){
        return 'aku bertelur';
    }
}
$kucing = new Hewan();
echo $kucing->mata = 'hitam';
echo $kucing->hidung = 'oval';
```

```
echo $kucing->telinga = 'runcing';


echo $kucing->hidup();
echo $kucing->berenang();
echo $kucing->bertelur();
```

Pada kode diatas kita tambahkan *access modifier* pada setiap *property* dan *method*. Pada isi method terdapat `return`, yang menjadi perbedaan antar `echo` dan `return` yakni jika `echo` tidak mengembalikan nilai. Sedangkan `return` mengembalikan nilai dari suatu *method* untuk diproses kembali pada tahap selanjutnya.

3. Selanjutnya kita akan membuat object ‘kucing’ dan menampilkan hasil dari masing-masing property dan method
   - Property mata

     ```
     $kucing = new Hewan();
     echo $kucing->mata = 'hitam';
     ```

     hitam

     Hasil: Berhasil karena memiliki hak akses public

   - Property hidung

     ```
     echo $kucing->hidung = 'oval';
     ```

     Fatal error: Uncaught Error: Cannot access protected property Hewan::$hidung in C:\xampp\htdocs\pbo\oop\Visibility.php:24 Stack trace: #0 {main} thrown in **C:\xampp\htdocs\pbo\oop\Visibility.php** on line **24**

     Hasil: Error karena property protected tidak bisa diakses setelah membuat object dari luar class

   - Property telinga

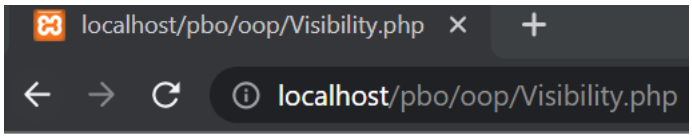
     ```
     echo $kucing->telinga = 'runcing';
     ```

Fatal error: Uncaught Error: Cannot access private property Hewan::$telinga in C:\xampp\htdocs\pbo\oop\Visibility.php:25 Stack trace: #0 {main} thrown in **C:\xampp\htdocs\pbo\oop\Visibility.php** on line **25**

Hasil: Error karena property private tidak bisa diakses setelah membuat object dari luar class

- Method hidup

```
echo $kucing->hidup();
```



aku hidup

- Method berenang

```
echo $kucing->berenang();
```

Fatal error: Uncaught Error: Call to protected method Hewan::berenang() from context '' in C:\xampp\htdocs\pbo\oop\Visibility.php:28 Stack trace: #0 {main} thrown in **C:\xampp\htdocs\pbo\oop\Visibility.php** on line **28**

- Method bertelur

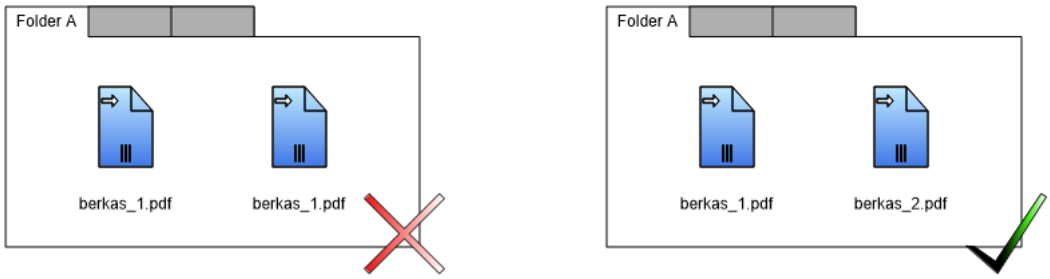
```
echo $kucing->bertelur();
```

Fatal error: Uncaught Error: Call to private method Hewan::bertelur() from context '' in C:\xampp\htdocs\pbo\oop\Visibility.php:29 Stack trace: #0 {main} thrown in **C:\xampp\htdocs\pbo\oop\Visibility.php** on line **29**

3.6. *Namespace* dan *Import*

Pada pengembangan aplikasi, kita tidak mungkin untuk membuat semua fitur dari aplikasi tersebut dalam satu file. Karena akan menyulitkan saat proses *debugging* dan kode menjadi tidak terstruktur sehingga sulit untuk *trace error* yang muncul. Sehingga hal yang harus dilakukan yakni terpaksa kita harus memecah kode menjadi beberapa file sesuai dengan fungsi dan fitur nya masing-masing. Pada OOP kita dapat memecah kode menjadi beberapa bagian yakni dengan menggunakan `use`. Adapun untuk kode dengan menggunakan teknik *procedural* kita bisa menggunakan `require` dan `require_once`.

Selanjutnya pada setiap kode yang kita buat terkadang memiliki nama *class* yang sama karena memiliki fitur yang mirip. Dimana ketika kita membuat *class* yang sama pada projek kita maka akan muncul error. Untuk lebih memudahkan pemahaman perhatikan gambar dibawah ini.



Konsep *namespace* pada OOP PHP mirip seperti nama file pada suatu folder, ketika kita memiliki nama file yang sama pada satu folder maka laptop kita akan memberi peringatan bahwa nama file sama. Kesimpulannya *namespace* berguna untuk melakukan grouping dari beberapa class. Selanjutnya apa korelasi nya dengan import? Import dengan `use` pada OOP PHP berguna untuk memanggil namespace pada suatu class. Sedangkan import dengan `require` atau `require_once` berguna untuk memanggil file yang terpisah. Untuk lebih jelasnya perhatikan kode berikut.

1. Buat folder baru dengan nama namespace sejajar dengan folder oop.
2. Buat file dengan nama Import.php dan Namespace.php di folder namespace
3. Buat dua class dengan nama yang sama di file Namespace.php

```
<?php

class Hewan {

}

class Hewan {

}
```

Didalam OOP PHP kita dapat membuat banyak class di satu file yang sama

4. Isikan kode berikut di file Import.php dan tampilkan hasilnya di browser

```
<?php
require_once('./Namespace.php');
```

`require_once` berfungsi untuk memanggil nama file, pada kode diatas yakni memanggil file Namespace.php.

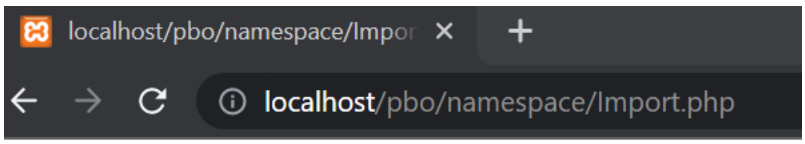
Fatal error: Cannot declare class Hewan, because the name is already in use in **C:\xampp\htdocs\pbo\namespace\Namespace.php** on line **7**

Info error diatas menyebutkan bahwa class dengan nama Hewan sudah di deklarasikan sebelumnya. Untuk memperbaikinya kita dapat menggunakan namespace seperti berikut.

5. Ubah kode di Namespace.php menjadi seperti berikut dan tampilkan di browser

```
<?php
namespace NamespaceSatu\Hewan {
    class Hewan {

    }
}
namespace NamespaceDua\Hewan {
    class Hewan {

    }
}
```

`namespace NamespaceSatu\Hewan` artinya kita membuat namespace dengan nama `NamespaceSatu` yang mana memiliki sub-namespace `Hewan`, begitupun `NamespaceDua\Hewan`



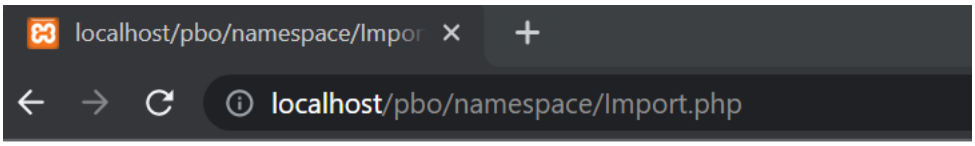
6. Browser tidak menampilkan apapun, untuk itu kita ubah sedikit kode di file Namespace.php seperti berikut

```
<?php
namespace NamespaceSatu\Hewan {
    class Hewan {
        function __construct(){
            echo "Hai saya dari NamespaceSatu\Hewan";
        }
    }
}
```

```
namespace NamespaceDua\Hewan {
    class Hewan {
        function __construct(){
            echo "Hai saya dari NamespaceDua\Hewan";
        }
    }
}
```

7. Selanjutnya ubah file Import.php seperti berikut dan tampilkan hasilnya di browser

```
<?php

require_once('./Namespace.php');

$kucing = new NamespaceSatu\Hewan\Hewan();
$kelinci = new NamespaceDua\Hewan\Hewan();

print_r($kucing,true);
print_r($kelinci,true);
```



Hai saya dari NamespaceSatu\HewanHai saya dari NamespaceDua\Hewan

Sedangkan untuk mengakses *namespace* antar *class* maka ikuti langkah berikut.

1. Buat file ClassNamespace.php di folder namespace dan isi dengan kode berikut

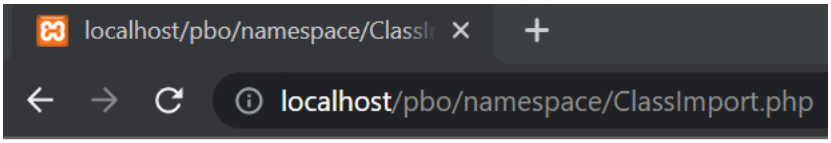
```
<?php
namespace Pbo\BelajarNamespace;
class ClassNamespace {
    public function show(){
        return "aku dari pbo\namespace\ClassNamespace";
    }
}
```

Pada kode diatas ditambahkan nama `namespace Pbo\BelajarNamespace;` untuk membuat grouping namespace dengan nama `BelajarNamespace;`

2. Selanjutnya buat file dengan nama ClassImport.php di folder namespace dan isi dengan kode berikut

```php
<?php

//import file ClassNamespace.php
require('./ClassNamespace.php');

//import namespace dari class ClassNamespace
use Pbo\BelajarNamespace\ClassNamespace;

class ClassImport {
    function showFromClassNamespace() {

        //buat object dari ClassNamespace
        $object = new ClassNamespace();
        return $object;
    }
}

$objectClassImport = new ClassImport();
print_r($objectClassImport->showFromClassNamespace());
```

maka hasilnya akan seperti berikut



Pbo\BelajarNamespace\ClassNamespace Object ( )

3.7. *Encapsulation (Setter – Getter)*

Encapsulation atau *setter-getter* merupakan salah satu konsep dimana kita memproteksi perubahan nilai dari property pada suatu class, yang mana property tersebut tidak bisa diakses secara langsung ketika kita telah membuat object. Karena menggunakan istilah "proteksi" maka kita akan membuat property tersebut memiliki access modifier private. Untuk memudahkan pemahaman ikuti kode berikut:

1. Buat file dengan nama Encapsulation.php di folder oop dan isi file dengan kode berikut

```php
<?php
class Hewan {
    // variabel kaki menggunakan tipe data string dan access modifier
private
    private string $kaki;
}
```

```
$ular = new Hewan();
$ular->kaki = 'tidak punya kaki';
print_r($ular->kaki);
```

pada kode diatas kita membuat *class* bernama `Hewan`, yang mana memiliki property "kaki" dengan tipe data *string* yang diberi *access modifier* nya yaitu *private*. Selanjutnya kita buat *object* dengan nama ular dan meng-*assign* / mengisi property kaki ular. Maka hasilnya yakni sebagai berikut:

Fatal error: Uncaught Error: Cannot access private property Hewan::$kaki in C:\xampp\htdocs\pbo\oop\Encapsulation.php:10 Stack trace: #0 {main} thrown in **C:\xampp\htdocs\pbo\oop\Encapsulation.php** on line **10**

Hal ini terjadi karena property kaki bersifat private, sehingga tidak bisa untuk mengisi property kaki secara langsung.

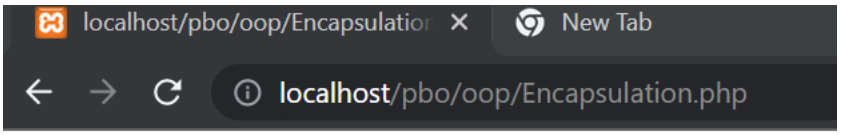
2. Untuk dapat mengakses nya kita akan membuat setter dan getter untuk property kaki. Perhatikan kode berikut:

```
<?php
class Hewan {
   //property kaki
    private string $kaki;

    //setter
    public function setKaki($kaki){
        $this->kaki = $kaki;
    }

    //getter
    public function getKaki(){
        return $this->kaki;
    }
}
```

Kemudian buat objek dengan nama 'ular' dan ayam', dan tampilkan dibrowser

```
<?php
class Hewan {
   //property kaki
```

```
    private string $kaki;

    //setter
    public function setKaki($kaki){
        $this->kaki = $kaki;
    }

    //getter
    public function getKaki(){
        return $this->kaki;
    }
}

$ular = new Hewan();
$ular->setKaki('ular tidak punya kaki'); //mengisi property kaki ular
print_r($ular->getKaki());

$ayam = new Hewan();
$ayam->setKaki('ayam berkaki dua');  //mengisi property kaki ayam
print_r($ayam->getKaki());
```



ular tidak punya kakiayam berkaki dua

3.8. *Inheritance*

Inheritance merupakan salah satu kemampuan dari OOP dimana kita dapat mewariskan sifat berupa *property* ataupun *method* kepada class turunan. Analogi *inheritance* seperti seorang anak yang mewarisi ciri-ciri dan sifat dari orang tua nya. Untuk mengaktifkan *inheritance* di class turunan, maka kita menambahkan `extends <nama kelas induk>` pada saat membuat class turunan. Untuk lebih jelasnya mari kita kombinasikan encapsulation dan inheritance seperti kode berikut:

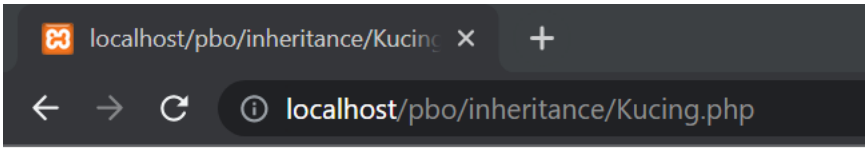
1. Buat folder dengan nama inheritance di sejajar dengan folder oop
2. Buat file dengan nama Hewan.php dan isi kode seperti berikut

```php
<?php
namespace MakhlukHidup;
class Hewan {

    private string $mata;
    private string $telinga;
    private string $kaki;

    public function setMata($mata) {
        $this->mata = $mata;
    }

    public function getMata(){
        return $this->mata;
    }

    public function setTelinga($telinga) {
        $this->telinga = $telinga;
    }

    public function getTelinga(){
        return $this->telinga;
    }

    public function setKaki($kaki) {
        $this->kaki = $kaki;
    }

    public function getKaki(){
        return $this->kaki;
    }
}
```

3. Selanjutnya buat Kucing.php dan isikan kode berikut

```php
<?php
require_once('./Hewan.php');
use MakhlukHidup\Hewan;

// pakai extends untuk memanggil sifat turunan dari class Hewan
class Kucing extends Hewan {

}

$siOren = new Kucing();

$siOren->setMata('mata aku hitam pekat');
$siOren->setTelinga('telinga aku runcing');
$siOren->setKaki('kaki aku kekar');

echo $siOren->getMata();
echo '<br/>';
echo $siOren->getTelinga();
echo '<br/>';
echo $siOren->getKaki();
echo '<br/>';
```

Kalau kita perhatikan *class* Kucing tidak memiliki *property* atau *method* apapun. Namun karena *class* Kucing merupakan turunan dari *class* Hewan, maka *class* Kucing dapat menggunakan *property* maupun *method* dari *class* Hewan. Berikut hasil nya di browser.



mata aku hitam pekat
telinga aku runcing
kaki aku kekar

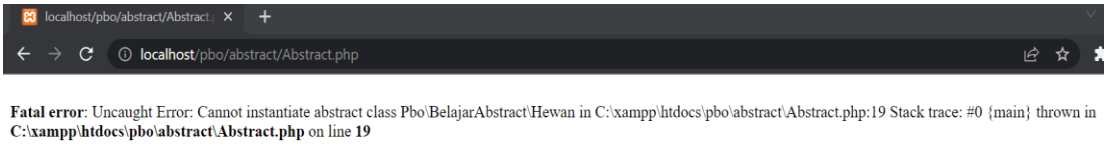
3.9. *Abstract Class*

Abstract class merupakan sebuah *class* "kontrak" yang mana untuk setiap *class* turunan nya harus mengimplementasikan *property* ataupun method pada *abstract class*. *Abstract class* tidak bisa untuk dibuat *object* nya, sehingga sebuah *abstract class* harus melakukan *inheritance* kepada *class* turunannya menggunakan `extends`. *Abstract class* hanya menerima *access modifier public* dan *protected*. Untuk lebih jelasnya perhatikan kode berikut:

6. Buat folder dengan nama abstract sejajar dengan folder oop
7. Buat file dengan nama Abstract.php didalam folder abstract dan isi dengan kode berikut:

```
<?php
namespace Pbo\BelajarAbstract;
//abstract class selalu diawali dengan keyword abstract
abstract class Hewan {
    public $nama;
    public function __construct($nama){
        $this->nama = $nama;
    }

    public function bergerak() {
        return 'namaku '.$this->nama. ' saya bisa bergerak bebas';
    }
}

$hewan = new Hewan('Si Oren');
echo $hewan->bergerak();
```

8. Jalankan di browser maka akan muncul error seperti berikut



Fatal error: Uncaught Error: Cannot instantiate abstract class Pbo\BelajarAbstract\Hewan in C:\xampp\htdocs\pbo\abstract\Abstract.php:19 Stack trace: #0 {main} thrown in **C:\xampp\htdocs\pbo\abstract\Abstract.php** on line **19**

Hal ini dikarenakan kita tidak bisa membuat *object* langsung dari *abstract class* tersebut, maka untuk dapat mengakses *property* dan *method* nya kita harus meng-*inheritance abstract class*.

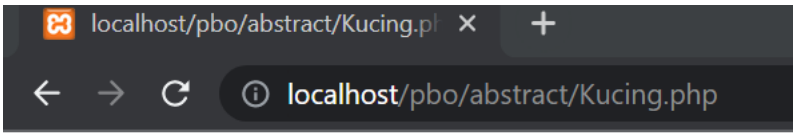
9. Selanjutnya hapus object dan method bergerak pada abstract class Hewan

```
<?php
namespace Pbo\BelajarAbstract;
//abstract class selalu diawali dengan keyword abstract
abstract class Hewan {
    public $nama;
    public function __construct($nama){
        $this->nama = $nama;
    }
```

```
    public function bergerak() {
        return 'nama saya '.$this->nama. ' saya bisa bergerak bebas';
    }
}
```

10. Kemudian buat file dengan nama Kucing.php didalam folder abstract dan isikan kode berikut, terakhir tampilkan di browser

```
<?php
require_once('./Abstract.php');
use Pbo\BelajarAbstract\Hewan;

class Kucing extends Hewan {
}

$kucing = new Kucing('Si Oren');
echo $kucing->bergerak();
```



nama saya Si Oren saya bisa bergerak bebas

3.10. *Interface*

Interface merupakan cara untuk membuat "kontrak" untuk setiap *class* yang mengimplementasikan dirinya sama seperti *abstract class*. Namun yang membedakan *interface* dengan *abstract class* yakni *interface* hanya menentukan nama suatu *method* saja tidak dengan isi dari *method* tersebut. Selain itu *interface* hanya boleh bersifat *public*, dan juga menggunakan keyword `implement` untuk menggunakan *interface* pada suatu *class*. Untuk lebih jelasnya ikuti kode berikut:

1. Buat folder dengan nama interface sejajar dengan folder oop
2. Buat file Interface.php didalam folder interface dan isikan kode berikut

```
<?php
interface Hewan {
    public function bergerak() : string;
    public function makan() : string;
}
```

Interface hanya membuat nama method saja namun tidak dengan isinya

3. Selanjutnya buat file Kuda.php di dalam folder interface dan isikan kode berikut, dan tampilkan hasilnya di browser

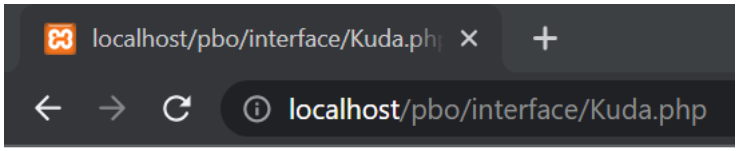
```
<?php
require_once('./Interface.php');

//gunakan keyword implements untuk menggunakan interface di suatu class
class Kuda implements Hewan {

}

$object = new Kuda();
```

Fatal error: Class Kuda contains 2 abstract methods and must therefore be declared abstract or implement the remaining methods (Hewan::bergerak, Hewan::makan) in **C:\xampp\htdocs\pbo\interface\Kuda.php** on line **5**

Error diatas menampilkan bahwa ada dua method yang terdapat di interface Hewan (method bergerak dan makan) yang belum diimplementasikan oleh class Kuda.

4. Terakhir kita ubah Kuda.php menjadi seperti berikut, dan tampilkan di browser

```
<?php
require_once('./Interface.php');
class Kuda implements Hewan {

    public function bergerak(){
        return 'kuda bisa bergerak';
    }
    public function makan(){
        return 'kuda memakan rerumputan';
    }
}

$object = new Kuda();
echo $object->bergerak();
echo "<br/>";
echo $object->makan();
echo "<br/>";
```



kuda bisa bergerak
kuda memakan rerumputan

3.11. *Polymorphism*

Polymorphism merupakan kemampuan dari sebuah *object* untuk berubah bentuk menjadi bentuk lain dan erat kaitannya dengan inheritance. Untuk lebih jelasnya kita pelajari kode berikut:

1. Buat folder dengan nama polymorphism sejajar dengan folder oop
2. Buat file dengan nama Hewan.php didalam folder polymorphism dan isikan kode sebagai berikut

```
<?php

class Hewan {

    public $nama;

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

class Harimau extends Hewan {

}

class Singa extends Hewan {

}

class Kucing {
    public Hewan $hewan;
}
```

Kode diatas terdapat class Harimau dan Singa, yang mana merupakan turunan dari class Hewan. Namun class Kucing hanya membuat object dengan nama $hewan dari class Hewan.

3. Buat file dengan nama Felis.php didalam folder polymorphism dan isikan kode sebagai berikut

```
<?php
require_once('./Hewan.php');

$kucing = new Kucing();
$kucing->hewan = new Harimau('kucing loreng');
print_r($kucing);

$kucing->hewan = new Singa('simba');
print_r($kucing);
```

pada kode diatas kita dapat membuat *object* harimau yang terhubung langsung dengan *class* kucing, yang mana *class* kucing tersebut membuat *object* di file Hewan.php pada *class* Hewan. Sehingga merubah bentuk dari *object-object class* turunan nya.

Berikut hasil dari kode diatas

Kucing Object ([hewan] => Harimau Object ([nama] => kucing loreng)) Kucing Object ([hewan] => Singa Object ([nama] => simba))