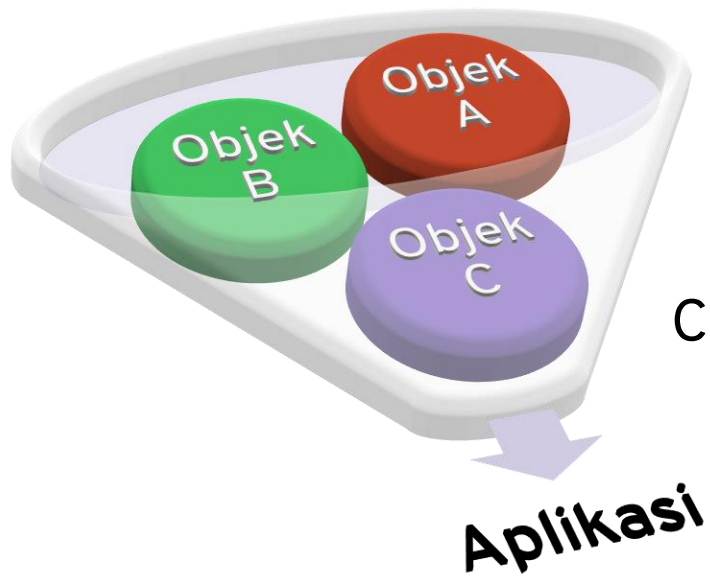


Konsep OOP

Object Oriented Programming

Apa itu OOP?

Paradigma dalam mengembangkan aplikasi dengan berorientasi
pada objek



Bagaimana Caranya?

Caranya dengan memecah suatu aplikasi menjadi beberapa bagian (yang kemudian disebut dengan objek) di mana masing-masingnya dapat berdiri sendiri

Contoh Objek Pada Apps?

- ❖ Objek Tampilan (menangani user interface),
- ❖ Objek Helper (menangani fungsi-fungsi umum),
- ❖ Objek Database (menangani interaksi dengan database),
- ❖ Objek Security (menangani keamanan, otorisasi),
- ❖ Objek File (menangani operasi dengan file),
- ❖ Objek Error Handling (menangani error pada aplikasi)
- ❖ dsb

Mengapa OOP?

Keuntungan vs Kekurangan

Keuntungan vs Kekurangan

- ❖ Menyederhanakan problem yang kompleks
- ❖ Efisien karena suatu objek dapat digunakan kembali
- ❖ Memudahkan mengelola kode & menemukan bug
- ❖ Mempercepat pengembangan aplikasi

- ❖ Lebih kompleks untuk dipelajari (karena banyak konsep-konsep baru)
- ❖ Ukuran kode aplikasi relative lebih besar
- ❖ Waktu eksekusi program lebih lambat

Paradigma Sebelum OOP?

Procedural atau functional yaitu menggunakan fungsi

Coding Procedural

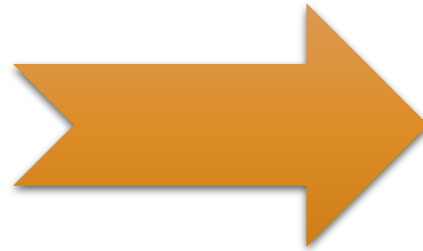
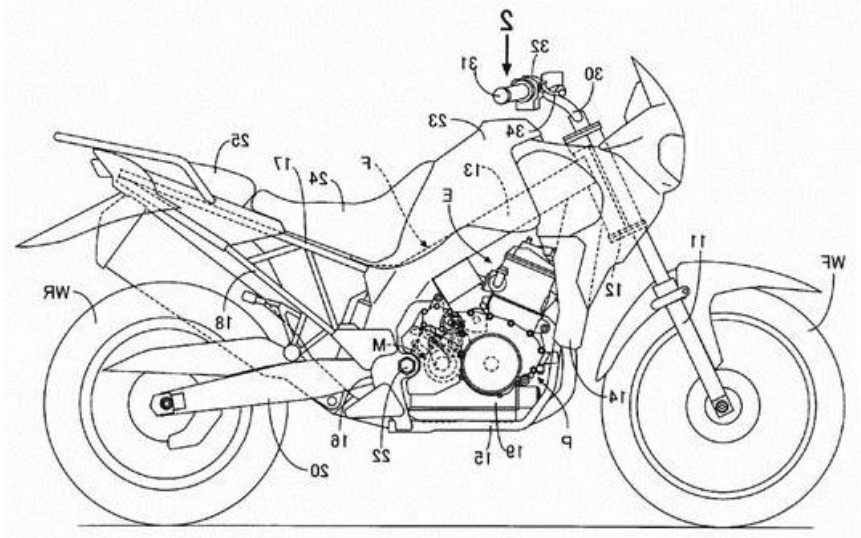
```
<?php
define("phi", 22/7);

function lebarLingkaran($radius){
    return 2*phi*$radius;
}

function luasLingkaran($radius){
    return phi * $radius * $radius;
}

echo "Lebar: ".lebarLingkaran(7); // hasilnya 44
echo "Luas: ".luasLingkaran(7); // hasilnya 154
```


Class vs Object



Class

Blue print atau desain atau rancangan dari suatu object

```
class NamaKelas
{
    // kode
}
```

Contoh Class (Static)

Member class yang dapat dipanggil tanpa meng-*instance* class menjadi object

❖ Method: money()

:: akses static property&method

```
Helper.php
<?php
class Helper
{
    public static function money($amount){
        return "Rp. " . number_format($amount, 2);
    }
}

echo Helper::money(7500); // Rp. 7,500.00
```

Contoh Class (Non Static)

Kumpulan dari function
(method) dan variable
(property)

- ❖ Property: phi, \$radius
- ❖ Method: lebar(), luas()

self:: akses constant (static)

\$this-> akses
property&method

```
<?php
class Lingkaran
{
    const phi = 22/7;
    public $radius;
    public function lebar(){
        return 2 * self::phi * $this->radius;
    }
    public function luas(){
        return self::phi * $this->radius * $this->radius;
    }
}
```

Lingkaran.php

Object

Instance dari suatu class

```
$kelas1 = new NamaKelas();  
$kelas2 = new NamaKelas();
```

Contoh Object

Include / require file class Lingkaran

```
<?php                                                                    object.php
require "../class/Lingkaran.php";

$lingkaran01 = new Lingkaran();
$lingkaran01->radius = 7;
echo "Lebar: " . $lingkaran01->lebar(); // hasilnya 44
echo "Luas: " . $lingkaran01->luas(); // hasilnya 154

$lingkaran02 = new Lingkaran();
$lingkaran02->radius = 14;
echo $lingkaran02->lebar(); // hasilnya 88
echo $lingkaran02->luas();
```

Class Constuctor

Method yang pertama kali dijalankan ketika class di-instance / dibuat, biasanya digunakan untuk menginisialisasi property

```
<?php                                     Test.php
class Test
{
    public function __construct(){
        echo "class test dibuat";
    }
}

$test1 = new Test(); // class test dibuat
```

Contoh Class Constructor

```
<?php                                     Lingkaran.php
class Lingkaran
{
    const phi = 22 / 7;
    public $radius;

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

    // method lain
}
```

```
<?php                                     object.php
require "Lingkaran.php";

$lingkaran01 = new Lingkaran(7);
echo "Lebar: ".$lingkaran01->lebar();
echo "Luas: ".$lingkaran01->luas();
```


Inheritance / Pewarisan

- ❖ Inheritance = class child memiliki sifat yang sama dengan parent (baik property maupun method)
- ❖ keyword = extends

```
class child extends parent  
{  
  
}
```

Contoh Class BaseAccount & Account

Parent Class

BaseAccount

- ❖ \$number;
- ❖ \$balance;
- ❖ __construct(\$number, \$balance)
- ❖ deposit(\$amount)
- ❖ withdraw(\$amount)
- ❖ print()

Child Class

Class Account

- ❖ transfer()



Semua property dan method yang dimiliki oleh parent class (BaseAccount) akan diwariskan atau akan juga dimiliki oleh child class (Account)

Contoh Class BaseAccount & Account

```
<?php                                     BaseAccount.php
class BaseAccount
{
    public $number;
    public $balance;
    public function __construct($number, $balance){
        $this->number = $number;
        $this->balance = $balance;
    }
    public function deposit($amount){
        $this->balance = $this->balance + $amount;
    }
    public function withdraw($amount){
        $this->balance = $this->balance - $amount;
    }
    public function print(){
        return $this->balance;
    }
}
```

```
<?php                                     Account.php
require "BaseAccount.php";
class Account extends BaseAccount{
    // method & property lain
}
```

```
<?php                                     object.php
require "Account.php";
$account1 = new Account("A001", "100000");
$account1->deposit(25000);
echo "Saldo1: ".$account1->print();

$account2 = new Account("A002", "50000");
$account2->withdraw(50000);
echo "Saldo2: ".$account2->print();
```

Polymorphisme / Method Overriding

Method yang sama pada class child bisa memiliki implementasi yang berbeda dengan method pada class parent

Contoh Class BaseAccount & Account

```
<?php                                     BaseAccount.php
class BaseAccount
{
    // method & property lain
    public function print(){
        return $this->balance;
    }
}
```

```
<?php                                     Account.php
require "BaseAccount.php";
class Account extends BaseAccount{
    // method & property lain
    public function print(){
        echo "balance: " . $this->balance;
    }
}
```

```
<?php                                     object.php
require "Account.php";

$account1 = new BaseAccount("A001", "100000");
$account1->deposit(25000);
echo "Saldo1: " . $account1->print();

$account2 = new Account("A002", "50000");
$account2->withdraw(5000);
$account2->print();
```

Visibility / Ruang Lingkup

Visibility = keteraksesan method dan property pada suatu class dan childnya

3 Jenis Visibility

- ❖ Public = method & property bisa diakses secara global baik dari dalam class, object, maupun dari class childnya (default)
- ❖ Private = method & property HANYA bisa diakses dari class tersebut
- ❖ Protected = method & property bisa diakses dari class tersebut dan child-nya, TIDAK BISA diakses dari object

Contoh Implementasi Visibility Public

```
class BaseAccount
{
    public $number;
    public $balance;
    // method & property lainnya
}
```

```
class Account extends BaseAccount
{
    // method & property lain
    public function print(){
        echo "balance: " . $this->balance;
    }
}
```

```
$account1 = new Account("A001", "100000");
$account1->balance = 5000000000;
$account1->print();
```

Catatan:

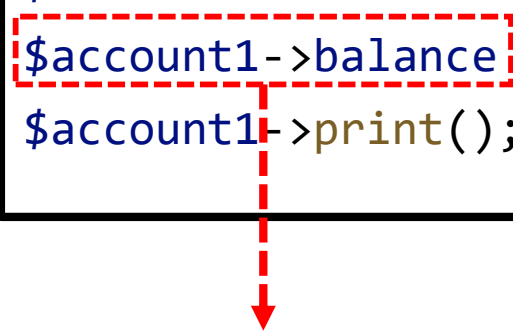
Property **balance** dan method **print()** dapat diakses baik dari class child maupun dari object, hal ini karena visibilitynya public

Contoh Implementasi Visibility Protected

```
class BaseAccount
{
    public $number;
    protected $balance;
    // method & property lainnya
}
```

```
class Account extends BaseAccount
{
    // method & property lain
    public function print(){
        echo "balance: " . $this->balance;
    }
}
```

```
$account1 = new Account("A001", "100000");
$account1->balance = 5000000000;
$account1->print();
```



PHP Fatal error: Uncaught Error: Cannot access protected property Account::\$balance in C:\xampp\htdocs\example\oop\visibility\protected.php:18

Catatan:

Akan terjadi error saat pemanggilan property balance pada object, hal ini dikarenakan visibility protected tidak mengizinkan property / method diakses dari object

Contoh Implementasi Visibility Private

```
class BaseAccount
{
    public $number;
    private $balance;
    // method & property lainnya
}
```

```
$account1 = new BaseAccount("A001", "100");
$account1->balance = 500;

$account2 = new Account("A002", "100");
$account2->print();
```

```
class Account extends BaseAccount
{
    // method & property lain
    public function print(){
        echo "balance: " . $this->balance;
    }
}
```

Undefined property:
Account::\$balance

Uncaught Error: Cannot
access private property
BaseAccount::\$balance

Catatan:

Akan terjadi error saat pemanggilan property balance pada class child, hal ini dikarenakan visibility private tidak akan mewariskan property atau method ke child serta tidak mengizinkan property / method diakses dari luar class tersebut

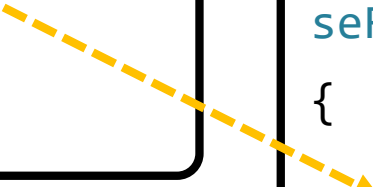
Interface

Interface = class yang semua method parent harus dideklarasikan atau diimplementasikan di class child

Keyword: implements

Contoh Interface

```
<?php                                     BaseFeature.php
interface BaseFunction
{
    public function transfer();
}
```



```
<?php                                     Account.php
require "BaseAccount.php";
require "BaseFeature.php";
class Account extends BaseAccount implements BaseFeature
{
    public function transfer(){
        echo "not yet";
    }
}
```

Catatan:

Jika tidak dideklarasikan pada class Account maka akan muncul "Fatal error: Class Account contains 1 abstract method and must therefore be declared abstract or implement the remaining methods (BaseFeature::transfer)"

Namespace

Namespace = identifier sebuah class yang umumnya diasosiasikan dengan lokasi file class tersebut berada, digunakan untuk membedakan class satu dengan yang lainnya meskipun namanya sama

Keyword: namespace & use

Contoh Penggunaan Namespace

Structure Directory App

index.php

+ base

BaseAccount.php

+ library

Account.php

```
<?php                                     index.php
require "library/Account.php";
use library\Account;

$account1 = new Account("A001", 50000);
echo $account1->print();
```

```
<?php                                     Account.php
namespace library;
require "base/BaseAccount.php";
use base\BaseAccount;
class Account extends BaseAccount
{
    public function print(){
        echo "balance: " . $this->balance;
    }
}
```

```
<?php                                     BaseAccount.php
namespace base;
class BaseAccount
{
    public $number;
    public $balance;
    // property atau method lain
}
```

Autoloading

Autoloading = Include class otomatis ketika class diciptakan

```
autoloader.php
<?php
spl_autoload_register(function ($class_name) {
    require $class_name . ".php";
    // require str_replace('\\', '/', $ class_name) . ".php";
});
```

Contoh Penggunaan Autoloading

Structure Directory App

index.php
autoloader.php
+ base
 BaseAccount.php
+ library
 Account.php

```
<?php                                     index.php
require "autoloader.php";
use library\Account;

$account1 = new Account("A01", 500);
echo $account1->print();
```

```
<?php                                     Account.php
namespace library;
use base\BaseAccount;
class Account extends BaseAccount
{
    public function print(){
        echo "balance: " . $this->balance;
    }
}
```

```
<?php                                     BaseAccount.php
namespace base;
class BaseAccount
{
    public $number;
    public $balance;
    // property atau method lain
}
```


Method Chaining

Mengakses beberapa method dalam satu baris perintah

```
$account2 = new BaseAccount('B001', 100000);  
$account2->deposit(25000)->withdraw(50000)->print();
```

Contoh Method Chaining

```
<?php                                     BaseAccount.php
class BaseAccount
{
    public $number;
    public $balance;
    public function __construct($number, $balance)
    {
        $this->number = $number;
        $this->balance = $balance;
    }
    public function deposit($amount){
        $this->balance = $this->balance + $amount;
        return $this;
    }
    public function withdraw($amount){
        $this->balance = $this->balance - $amount;
        return $this;
    }
    public function print(){
        echo "saldo : " . $this->balance;
        return $this;
    }
}
```

```
<?php                                     object.php
require "BaseAccount.php";

// nabung awal 100.000
$account1 = new BaseAccount("A001", 100000);
// nabung kemudian 25.000
$account1->deposit(25000);
// diambil 50.000
$account1->withdraw(50000);
// cetak buku tabungan
$account1->print();

$account2 = (new BaseAccount('B001', 100000))
->deposit(25000)->withdraw(50000)->print();
```

Terima Kasih