

# OOP PHP5

Erick Kurniawan, S.Kom, M.Kom

# Sejarah OOP di PHP

- OOP diperkenalkan sejak PHP3
- Masih simple (PHP3, PHP4)
- Karena perkembangan web application (ASP.NET, JSP) yang support full OOP
- Dikembangkan PHP5 yang support full OOP untuk memenuhi kebutuhan
- Pengembangan aplikasi yang besar (Enterprise Application)

# Deklarasi Class

```
<? //defineclass.php
class Person
{
    private $name;
    function setName($name)
    {
        $this->name = $name;
    }
    function getName()
    {
        return $this->name;
    }
};
?>
```

# Deklarasi Class

```
<? //defineclass.php
    $erick = new Person();
    $erick->setName("Erick");

    $rasmus = new Person();
    $rasmus->setName("Rasmus");

    echo $erick->getName()."\n";
    echo $rasmus->getName();
?>
```

# Objek Konstruktor

- Konstruktor adalah fungsi / method yang digunakan untuk inisialisasi awal variabel / method

# Objek Konstruktor

```
<? //konstruktor.php
class Person {
    private $name;
    function __construct($name)
    {
        $this->name = $name;
    }
    function getName()
    {
        return $this->name;
    }
};
?>
```

# Objek Konstruktor

```
<? //konstruktor.php
    $judy = new Person("Judy");
    $joe = new Person("Joe");

    echo $judy->getName() . "\n";
    echo $joe->getName();
?>
```

# Objek Destruktor

- Kebalikan dari objek konstruktor
- Dipanggil ketika objek di destroy
- Objek diberi nilai NULL

# Objek Destruktor

```
<? //destruktur.php
class MyClass {
    function __destruct()
    {
        print "Objek dengan tipe MyClass di
        destroyed\n";
    }
};

$objj = new MyClass();
$objj = NULL;
?>
```

# Static Properties / Method

- Method / Property dapat dideklarasikan secara statis
- Jika Method / Property dideklarasikan secara statik maka Method / Property tersebut dapat langsung diakses tanpa harus membuat instan class

# Static Properties

```
<? //staticproperty.php
class MyClass {
    static $myStaticVariable=15;
    static $myInitializedStaticVariable = 0;
};

echo MyClass::$myStaticVariable."\n";
MyClass::$myInitializedStaticVariable++;
echo MyClass::$myInitializedStaticVariable;
?>
```

# Static Method

```
<? //staticmethod.php
class PrettyPrinter {
    static function printHelloWorld()
    {
        print "Hello, World";
        self::printNewline();
    }
    static function printNewline()
    {
        print "\n";
    }
}
PrettyPrinter::printHelloWorld();
?>
```

# Class Constant

- Tipe data konstanta digunakan untuk menyimpan data yang nilainya selalu konstan
- Konstanta bersifat statis
- Dapat langsung diakses tanpa harus membuat instan class

# Class Constant

```
<? //konstanta.php
class MyColorEnumClass {
    const RED = "Red";
    const GREEN = "Green";
    const BLUE = "Blue";

    function printBlue()
    {
        print self::BLUE;
    }
}
echo MyColorEnumClass::RED."\n";
$obj = new MyColorEnumClass();
$obj->printBlue();
?>
```

# Cloning Object

```
<? //cloningobject.php
class MyClass {
    public $var = 1;
}

$obj1 = new MyClass();
$obj2 = $obj1;
$obj2->var = 2;
print $obj1->var;
?>
```

# Inheritance & Polymorphism

- Inheritance = Pewarisan
- Parent mewariskan sifat ke child
- Polymorphism = Banyak Bentuk
- Method dengan nama yang sama tapi beda parameternya (Overloading)
- Method dengan nama sama parameter sama tapi berada dalam kelas anak (Overriding)

# Inheritance & Polymorphism

```
//polymorphism.php
class Animal {
    function makeSound()
    {
        print "Error: This method should be re-
implemented in the children";
    }
}

class Cat extends Animal {
    function makeSound()
    {
        print "miau";
    }
}
```

# Inheritance & Polymorphism

```
class Dog extends Animal {
    function makeSound()
    {
        print "wuff";
    }
}
function printTheRightSound($obj)
{
    if ($obj instanceof Animal) {
        $obj->makeSound();
    } else {
        print "Error: Passed object yang salah";
    }
    print "\n";
}
```

# Inheritance & Polymorphism

```
<?  
printTheRightSound(new Cat());  
printTheRightSound(new Dog());  
?>
```

# Parent:: and Self::

```
//parentself.php  
class Ancestor {  
    const NAME = "Ancestor";  
    function __construct()  
    {  
        print "In " . self::NAME . "  
constructor\n";  
    }  
}
```

# Parent:: and Self::

```
class Child extends Ancestor {
    const NAME = "Child";
    function __construct()
    {
        parent::__construct();
        print "In " . self::NAME . "
constructor\n";
    }
}

$obj = new Child();
```

# Abstract Class

- Class bertipe abstrak
- Mempunyai method yang belum ada implementasinya (bertipe abstract)
- Digunakan dengan cara di extend / diturunkan

# Abstract Class

```
abstract class Shape {  
    function setCenter($x, $y) {  
        $this->x = $x;  
        $this->y = $y;  
    }  
  
    abstract function draw();  
    protected $x, $y;  
}
```

# Abstract Class

```
class Square extends Shape {  
    function draw()  
    {  
        //implementasinya  
    }  
}
```

```
class Circle extends Shape {  
    function draw()  
    {  
        //implementasinya  
    }  
}
```

# Interfaces

- Mendefinisikan method yang akan digunakan tapi belum ada implementasinya
- Keseluruhan method belum diimplementasikan
- Digunakan dengan cara di-implements

# Interfaces

```
interface Loggable {  
    function logString();  
}
```

```
class Person implements Loggable {  
    private $name, $address, $idNumber,  
    $age;  
    function logString() {  
        return "class Person: name = $this->name, ID = $this->idNumber\n";  
    }  
}
```

# Final Method

- Jika method dideklarasikan final pada class induk maka method tidak bisa digunakan lagi di class anaknya

# Final Method

```
//error karena method dideklarasikan final  
class MyBaseClass {  
    final function idGenerator()  
    {  
        return $this->id++;  
    }  
  
    protected $id = 0;  
}  
class MyConcreteClass extends MyBaseClass {  
    function idGenerator()  
    {  
        return $this->id += 2;  
    }  
}
```

# \_\_toString() Method

```
class Person {
    function __construct($name)
    {
        $this->name = $name;
    }
    function __toString()
    {
        return $this->name;
    }
    private $name;
}

$obj = new Person("Erick Kurniawan");
echo $obj;
```

# Exception

- Untuk menangkap dan handle kesalahan yang mungkin terjadi pada program
- Sama seperti pada c#, java, vb.net

# Exception

```
try {
    printObject(new MyName("Bill"));
    printObject(NULL);
    printObject(new MyName("Jane"));
} catch (NullHandleException $exception) {
    print $exception->getMessage();
    print " in file " . $exception->getFile();
    print " on line " . $exception->getLine() .
"\n";
} catch (Exception $exception) {
    // This won't be reached
}
```