Objected Oriented Programming in PHP

created by **Wai Yan Aung**

PHP, an open source language, allow programmers to quickly develop error-free programs using procedural and objected-oriented programming techniques.

Procedural programming

A procedural programming includes functions (methods) that can be called from the main flow of the program. The flow of the program jumps to the function (method), executes the code within the module, and then returns to the next statement in the main flow of the program.

Object-oriented programming (OOP)

```
<?php
        class User {
                private $name;
                public $gender;
                protected $age;
        }
```

OOP is a design philosophy that uses objects and methods rather than linear concepts of procedures and tasks (procedural programming) to accomplish programmatic goals. An object is a self-sustainable construct that enables re-usability of code. A method specifies one operation without providing any details to describe how the operation should be carried out.

Advantages of OOP

Object-oriented programming (OOP) has some advantages over procedural programming: modularity, code re-usability, information hiding, and debugging ease.

Modularity

OOP provides a clear modular structure for programs. Modularity refers to the concept of making multiple modules first and then linking and combining them to form a complete system. Modularity enables re-usability and minimizes duplication.

Reusability

Code can be re-use, without modification, to perform a specific service regardless of what application uses the code.

Information-hiding

The detail of internal implementation of a module (class) remain hidden from the outside world.

Debugging

Easier to fix problems because the module (class) is independent from other pieces of code. Modifying one piece of code doesn’t impact other pieces of code in the application.

Objected Oriented Programming in PHP

created by **Wai Yan Aung**

OOP fundamentals

Class

A class is a template for creating objects.

In PHP a class is a collection of variables and functions working with these variables.

To define a class, the class keyword is used, followed by a name and a code block delimited by curly braces.

```
class User {


}
```

Properties

Class member variables are called properties

In PHP, class member variables are called “properties” (also “attributes” or “fields”). They are defined by using one of the keywords public, protected or private, followed by a normal variable declaration.

```
class User{
        public $name;
}
```

Method

A method is a procedure associated with an object (similar to PHP functions)

A function (method) is a piece of code which may take some input in the form of parameter and does some processing and returns a value.

In OOP, a method is a part of a class and included in any object of that class. You can define method by using one of the keywords public, protected or private, followed by a normal name declaration.

```
class User{
        public $name;
        public function setName(){
        }
}
```

Properties and methods visibility

The property defined in above class were tagged as public. That means that it is accessible to anyone, or more precisely, from anywhere. This is called the visibility of the property or method.

private visibility

This type allows access only to members of the same class. Members declared as private may only be accessed by the class that defines the member.

Use this type if you want variable/function (method) to be visible in its own class only.

```
class User {

        //can be accessible within the same class

        private $name;

}
```

protected visibility

This type allows access to members of the same class and instances from classes that inherit from that one only.

Use this type if you want to make variable/function (method) to be visible in all classes that extend current class including the parent class.

```
class User {

        //accessible for same class and its child classes

        protected $name;

}
```

public visibility

This type refers to a property or method that is accessible from anywhere. Any classes or code in general from outside the class can access it.

Use this type if you want to make variable/function available from anywhere, other classes and instances of the object.

```
class User{

        //accessible from anywhere

        public $name;

}
```

Object

An instance of a class is called object.

Object refers to a particular instance of a class where the object can be a combination of variables, functions (methods), and data structures.

```
// Instance (or object) of User class
$user = new User();
```

Creating a PHP class : example

```
<?php
class User {
}
```

Set private properties

Private visibility: Fatal error if try to access from outside the class

```
<?php
class User {
        private $name;
        private $gender;
}
```

Set public properties

Public visibility: can be access and modify from anywhere

```
<?php
class User {
        private $name;
        private $gender;
        public $age;
}
```

Set initialized properties (or properties with default value)

```
<?php
class User {
        private $name;
        private $gender;
```

```
            public $age;
            private $department = 'Mechanical';
}
```

Set class methods

```
<?php
class User {
            private $name;
            private $gender;
            public $age;
            private $department = 'Mechanical';

            /* set value to 'private $name' property */
            public function setName($name){
                        $this->name = $name;
            }

            /* get value from 'private $name' property */
            public function getName(){
                        return $this->name;
            }
}
```

The complete class

```
<?php
class User {
            private $name;
            private $gender;
            public $age;
            private $department = 'Information Technology';

            public function setName($name){
                      $this->name = $name;
            }

            public function getName(){
                      return $this->name;
            }

            public function setGender($gender){
                      $this->gender = $gender;
            }
            public function getGender(){
                      return $this->gender;
            }

            public function setDepartment($dep){
                      $this->department = $dep;
            }
            public function getDepartment(){
                      return $this->department;
            }
}
```

Making instances of the class

```
$user = new User();
```

Assign value to instance variables

```
$user→setName('Doe Lone');
$user->setGender('M');

//public property : no error, accessible from anywhere
$user->age = 30;
```

```
//Fatal error: Cannot access private property

//Private and protected methods can not be access directly

$user->department = 'Computer';


//The right way to assign value for protected and private methods

$user→setDepartment('Computer');
```

Fetch and print instance variables value

```
echo $user->getName();

echo $user->getDepartment();

echo $user->age;


//Fatal error: Cannot access private property User::$department

echo $user->department;


/* You can make as many instance as you want */

$user2 = new User();


//prints Mechanical (default value for all instances)

echo $user2->getDepartment();


//changed the default department

$user2->setDepartment('Computer');
```

PHP provides features to implement an object oriented model. These features includes encapsulation, inheritance and polymorphism.

Encapsulation or information hiding

Encapsulation is just wrapping some data in an object. The term “encapsulation” is often used interchangeably with “information hiding”.

Hiding the internals of the object protects its integrity by preventing users from setting the internal data of the component into an invalid or inconsistent state.

You can use encapsulation if the properties of an object are private, and the only way to update them is through public methods.

For example in this code we can protect our User objects from setting an invalid gender parameter.

```
class User {
        private $name;
        private $gender;

        public function getName() {
                return $this->name;
        }

        public function setName($name) {
                $this->name = $name;
                return $this;
        }

        public function getGender() {
                return $this->gender;
        }

        public function setGender($gender) {
                if ('male' !== $gender and 'female' !== $gender) {
                        throw new Exception('Set male or female for gender');
                }
                $this->gender = $gender;
                return $this;
        }
}

        $user = new User();
        $user->setName('Michal');
        $user->setGender('male');
```

In above example we should assign male or female value for setGender method or it will print Fatal error: Uncaught exception 'Exception' with message 'Set male or female for gender'.

Polymorphism

Ability to process objects differently. An integral part of polymorphism is the common interface.

Polymorphism is a design pattern in which classes have different functionality while sharing a common **interface**. You can define an interface in PHP using **interface or abstract** classes.

This example will describe the general concept of polymorphism, and how it can easily be deployed in PHP.

In the example given below, the interface with the name of Shape commits all the classes that implement it to define an abstract method with the name of area().

Step 1 : declaring interface using interface keyword.

```
interface Shape {

}
```

Step 2: add method (methods can be defined in the interface without the body).

```
interface Shape {

 public function area();

}
```

Step 3: create Circle class that implements the Shape interface by putting into the area() method the formula that calculates the area of circles:

```
class Circle implements Shape {

		private $radius;


		public function __construct($radius) {

			$this -> radius = $radius;

		}


		public function area() {

			return $this -> radius * $this -> radius * pi();

		}

}
```

Step 4: The rectangle class also implements the Shape interface but defines the method area() with a calculation formula that is suitable for rectangles:

```
class Rectangle implements Shape {

		private $width;

		private $height;
```

```
        public function __construct($width,$height) {
                $this -> width  = $width;
                $this -> height = $height;
        }

        public function area() {
                return $this -> width * $this -> height;
        }
}
```

By implementing interface we can be sure that all of the objects calculate the area with the method that has the name of area(), whether it is a rectangle object or a circle object (or any other shape).

```
$rect = new Rectangle(3,3);
echo $rect -> area();

$circ = new Circle (5);
echo $circ -> area();
```

Inheritance

Take properties of existing objects

Since a class is simply a collection of related functions and variables, one way of adding new functionality is to amend the code. In the early stages of development, this is usually the correct approach, but a fundamental aim of OOP is reusability and reliability. There’s no need to code everything from scratch, you can base a new class on an existing one. OOP classes are extensible through inheritance.

A child class or subclass in OOP can inherit all the features of its parent or superclass, adapt some of them, override existing methods and properties, and add new ones of its own.

```
class Human {
        protected $name;

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

In order to declare that one class inherits the code from another class, we use the extends keyword.

```
class Person extends Human {
        /* no constructor, adapted the parent constructor */
        public function getName(){
                /*Will return name property defined in parent class*/
                return $this->name;
        }
}
```

The above code demonstrated that how Person class inherited the parent constructor and added new method as per our need.

```
//constructor function inherited from the parent
$person = new Person('Kelly');
echo $person->getName();
```

Both Composition and Aggregation are the form of association between two objects, but there is a subtle difference between composition and aggregation. In this tutorial we'll also discuss Abstract classes and Interface.

Abstract class

A class designed only as a parent from which sub-classes may be derived.

Abstract classes are classes that may contain abstract method(s).

An abstract method is a method that is declared, but contains no implementation.

Abstract classes can not be instantiated, and require subclasses to provide implementations for the abstract methods.

To make a class abstract, add the keyword abstract in front of class and the class name in the class definition.

```
abstract class Products {
        protected $name;
        protected $type;

        public function __construct($name, $type){
                $this->type = $type;
                $this->name = $name;
        }

        public function getType() {
                return $this->type;
        }

        public function getName() {
                return $this->name;
        }

        abstract public function getTypeName();
        }
```

Abstract classes can not be instantiated, so we'll extend it by making a child class product. An abstract function also declared in above example and we know, methods marked abstract in the parent's class declaration must be defined by the child class.

```
class Product extends Products {
        public function getTypeName() {
                return $this -> name .':'. $this -> type;
        }
}

$product = new Product('Paper','80g');
echo $product->getTypeName();
```

Interfaces

An interface allows the computer to enforce certain properties on an object (class). It is actually a concept of abstraction and encapsulation.

An interface is very similar to an abstract class, but it has no properties and cannot define how methods are to be implemented. Instead, it is simply a list of methods that must be implemented.

In its most common form, an interface is a group of related methods with empty bodies. A child class can implement multiple interfaces.

```
interface Animal {
        public function breath();
        public function eat();
}

class Dog implements Animal{
        //new method
        public function bark() {
                return 'wof';
        }

        //implemented methods
        public function breath() {
                return 'breathing';
        }

        public function eat() {
                return 'eating';
        }
}

$dog = new Dog();
echo $dog -> breath(); //breathing
echo $dog -> bark(); //wof
echo $dog -> eat(); //eating
```

Composition and Aggregation

relationships between objects.

Composition is an important concept in PHP. In this design pattern an object creates another object. Aggregation occurs when an object is composed of multiple objects.

Composition is effectively an ownership relationship, while aggregation is a “contains” relationship. In composition, the parts can not exist outside the thing that contains them, but individual things can exist on their own as unique entities in aggregation.

Composition example

```
class Author {
        private $name;
```

```
	private $email;

	public function __construct($name,$email){
		$this->name = $name;
		$this->email = $email;
	}

	public function getName(){
		return $this->name;
	}

	public function getEmail(){
		return $this->email;
	}
}

class Book {
	//using array to allow multiple authors
	private $authors = array();
	private $price;
	private $name;

	public function __construct($name,$price){
		$this->name = $name;
		$this->price = $price;
	}

	public function setPrice($price){
		$this->price = $price;
	}

	public function getPrice(){
		return $this->price;
```

```
        }

        public function getName(){
                return $this->name;
        }

        public function addAuthor($name,$email){
                $this->authors[] = new Author($name,$email);
        }

        public function getAuthors(){
                return $this->authors;
        }
}

$book = new Book('Book Name',10.0);
$book -> addAuthor('Kelly','kelly@brainbell.com');
$book -> addAuthor('Ken','ken@brainbell.com');

/* print book info */
echo $book -> getName();
echo $book -> getPrice();

/* print book's authors info */
$authors = $book -> getAuthors(); //Array of authors
foreach ($authors as $author) {
        echo $author -> getName();
        echo $author -> getEmail();
}
```

Aggregation exmaple

```
class Author {
        private $name;
```

```
	private $email;

	public function __construct($name,$email) {
		$this->name = $name;
		$this->email = $email;
	}

	public function getName() {
		return $this->name;
	}

	public function getEmail(){
		return $this->email;
	}
}

class Book {
	private $authors;
	private $price;
	private $name;

	public function __construct($name,$price,$author){
		$this->name = $name;
		$this->price = $price;
		$this->author = $author;
	}

	public function setPrice($price){
		$this->price = $price;
	}

	public function getPrice(){
		return $this->price;
		}


		public function getName(){

				return $this->name;

		}


		public function getAuthor(){

				return $this->author;

		}

}


$author = new Author('Kelly','kelly@brainbell.com');

$book = new Book('Book Name',10.0,$author);


echo $book->getName(); //Book Name

echo $book->getAuthor() -> getName(); //Kelly

echo $book->getAuthor() -> getEmail(); //kelly@brainbell.com
```