Object Oriented Programming in PHP

A full documentation by

Md Humayun Rashid

Class: Class is a user defined data type that has data and functions. It is a skeleton or structure of objects. It can have properties or methods. For example, assume we have a class named “car”. Then we can define$brandName, $colour, $numberofWheel, $YearOfLaunch e.t.c. are the properties of it.

When the individual objects (Premio, Sonata, etc.) are created, they inherit all the properties and behaviours from the class, but each object will have different values for the properties.

Class Code example:
```
<?php
class Car {
  // code goes here...
}

$obj=new Car();//to create object
?>
```

Object: It is an instance of a class. Each object has all the properties and methods defined in the class, but they will have different property values.

Object Code Example:
```
<?php
class Car {
  // code goes here...
}

$obj=new Car();//to create object
?>
```

Constructor: Constructor is a method that executes automatically after creating an object. In php the name of this method is __construct(). We can write any code inside it. After creating an object we don’t need to specify or invoke the method in order to execute. It’ll execute as soon as the objects are created.

Code Example:

```
<?php
class car{
    function __construct(){
        echo "This is constructor";
    }
}
$BMW=new car();
?>//Output: This is constructor.
```

Constructor Overloading: It means we can have more than one constructor with different arguments and working. But in PHP it doesn’t support it directly. But indirectly we can do that. But not as relevant to the constructor overloading.

Code Example:

```
<?php
class MyClass {
    public $property1;
    public $property2;

    public function __construct($param1, $param2 = null) {
        $this->property1 = $param1;
        $this->property2 = $param2;
    }
}
// Usage
$obj1 = new MyClass('value1');
$obj2 = new MyClass('value1', 'value2');
?>
```

Here we can see we created a constructor with two parameters param1 and param2. On second argument param2 we set a value null. For that we can create an object with 1 or two parameters. If we give 1 parameter then $param2 is set to null by default. For these two scenarios we are getting two different objects. This is how constructor overloading can happen in PHP.

Destructor: Destructor is a method that executes after all the code are executed. It is the last method to execute in a class. In PHP, we have to use __destruct() for destructor.

Code Example:

```
<?php
class car{
    function __construct(){
        echo "This is constructor";
     }

   function __destruct(){
    echo "\nThis is a Destructor and The code is ended.";
   }
}
$BMW=new car();
?>
//This is constructor.
//This is a Destructor and The code is ended.
```

Access Modifier: It controls which method or properties are accessed by whom. It gives the protection to necessary methods and properties. There are three access modifiers. They are below:

(1) **Public:** the property or method can be accessed from everywhere. This is the default.
(2) **Protected:** the property or method can be accessed within the class and by classes derived from that class.
(3) **Private**: the property or method can ONLY be accessed within the class.

Code Example:

```
<?php
class Car {
  public $name;
  protected $color;
  private $numberPlate;
}

$mango = new Car();
$mango->name = 'Sonata'; // OK
$mango->color = 'blue'; // ERROR
$mango->numberPlate = '4523'; // ERROR
?>
```

Four pillars of OOP: The four pillars of object-oriented programming (OOP) are a set of principles that guide the design and implementation of object-oriented systems. These pillars are:

Encapsulation: It means wrapping up the data and information in a single unit. For example, An It office has many team like development, deployment, sales e.t.c. Now think of a member of development team needs some sales information. But he/she don't have the access of data of sales. So he/she needs to communicate with the man who is responsible for the data and ask to give access of data. This is encapsulation. Here the data and members that can manipulate them are wrapped in a single team "sales team".

Code Example:

```php
<?php
class Car {
    private $model;
    private $color;

    public function __construct($model, $color) {
        $this->model = $model;
        $this->color = $color;
    }

    public function getModel() {
        return $this->model;
    }

    public function getColor() {
        return $this->color;
    }

    public function start() {
        echo "The car is starting.\n";
    }
}

// Usage
$myCar = new Car('Toyota', 'Blue');
echo 'Model: ' . $myCar->getModel() . "\n";
echo 'Color: ' . $myCar->getColor() . "\n";
$myCar->start();
?>
```

Abstraction: Abstraction means displaying only the necessary information and hiding all the details. For example, suppose a man is driving a car. He knows if he presses the accelerator then the car will move. He doesn't knows how it occurs or the functionality. Abstraction can be achieved through abstract classes and interfaces.

Abstract Class Code Example:

```
<?php
abstract class ParentClass {
  // Abstract method with an argument
  abstract protected function prefixName($name);
}

class ChildClass extends ParentClass {
  public function prefixName($name) {
    if ($name == "John Doe") {
      $prefix = "Mr.";
    } elseif ($name == "Jane Doe") {
      $prefix = "Mrs.";
    } else {
      $prefix = "";
    }
    return "{$prefix} {$name}";
  }
}

$class = new ChildClass;
echo $class->prefixName("John Doe");
echo "<br>";
echo $class->prefixName("Jane Doe");
?>
```

Interface Code Example:

```
<?php
interface car{
    public function makeSound();
}

class sound implements car{
    public function makeSound(){
        echo "broom broom";
    }
}
$sound=new sound();
$sound->makeSound();
```

```
?>
```

Inheritance: Inheritance means inheriting properties and methods from the parent class. The child class couldn't change the properties and methods of the parent class.There are some kinds of inheritance. They are below:

Multiple inheritance: PHP doesn't support multiple inheritance in the traditional sense (i.e., a class inheriting from more than one class directly). However, we can achieve similar behaviour using interfaces and traits. Interfaces allow a class to implement multiple interfaces, and traits provide a way to reuse methods in multiple classes.

Multiple Inheritance Code Example:

```
<?php
interface Engine {
    public function start();
}

trait HonkTrait {
    public function honk() {
        echo "Honk! Honk!\n";
    }
}

class Car implements Engine {
    use HonkTrait;

    public function start() {
        echo "Car engine started.\n";
    }
}

?>
```

Multilevel inheritance: PHP supports multilevel inheritance, where a class extends another class, and a third class extends the second class, creating a chain of inheritance.

Multilevel Inheritance Code Example:

```
<?php
class Vehicle {
    public function start() {
        echo "Vehicle engine started.\n";
    }
}

class Car extends Vehicle {
    public function honk() {
        echo "Car honk! Honk!\n";
    }
}

class SportsCar extends Car {
    public function accelerate() {
        echo "Sports car accelerating.\n";
    }
}
?>
```

Hierarchical Inheritance: PHP supports hierarchical inheritance, where multiple classes inherit from a common base class.

Hierarchical Inheritance Code Example:

```
<?php
class Animal {
    public function eat() {
        echo "Animal is eating.\n";
    }
}

class Dog extends Animal {
    public function bark() {
        echo "Dog is barking.\n";
    }
}

class Cat extends Animal {
    public function meow() {
        echo "Cat is meowing.\n";
    }
}
?>
```

Polymorphism: It means having many forms. For example we can say A man is a husband, brother, father at a same time. It differs based on the situation. There are two types of polymorphism. They are,

- **Compile Time Polymorphism (Function Overloading)**
- **Run Time Polymorphism (Function overriding)**

The compile time polymorphism happens because of function overloading and run time polymorphism happens because of function overriding. Now let’s discuss about function overloading and overriding.

Function Overloading: When there are multiple functions that have same name but has different parameters then the function is said to be overloaded and this is called function overloading. Compile-time polymorphism is achieved through function overloading. This is when multiple methods in the same class have the same name but differ in the number or types of their parameters. But PHP doesn't supports traditional function over loading. We have to do default value sysytem to achieve it.

Function overloading Code Example:

```
<?php
class MathOperations {
    public function add($a, $b) {
        return $a + $b;
    }

    public function addThree($a, $b, $c = 0) {
        return $a + $b + $c;
    }
}

$math = new MathOperations();
echo $math->add(2, 3) . "\n";          // Outputs: 5
echo $math->addThree(2, 3) . "\n";     // Outputs: 5
echo $math->addThree(2, 3, 4) . "\n"; // Outputs: 9
?>
```

Function Overriding: When a derived class has a definition for one of the member functions of the base class. Then it is said to be function overridden.

Function overriding Code Example:

```
<?php
class Shape {
    public function calculateArea() {
        return 0;
    }
}

class Circle extends Shape {
    private $radius;

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

    public function calculateArea() {
        return 3.14 * $this->radius * $this->radius;
    }
}

class Square extends Shape {
    private $side;

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

    public function calculateArea() {
        return $this->side * $this->side;
    }
}

$circle = new Circle(5);
$square = new Square(4);

echo $circle->calculateArea() . "\n"; // Outputs: 78.5
echo $square->calculateArea() . "\n"; // Outputs: 16


?>
```

Trait: PHP doesn't supports multiple inheritance that's why the traits are used. Traits are used to declare methods that can be used in multiple classes. Traits can have methods and abstract methods that can be used in multiple classes.

Trait Code Example:

```
<?php
trait message1 {
public function msg1() {
    echo "OOP is fun! ";
  }
}

class Welcome {
  use message1;
}

$obj = new Welcome();
$obj->msg1();
?>
```

Static Methods & Properties: In PHP these are the methods nad properties that we can call without creating an object. We can directly use these methods and properties. To access a static method & properties use the class name, double colon (::), and the method name.

Static Method Code Example:

```
<?php
class greeting {
  public static function welcome() {
    echo "Hello World!";
  }
}

// Call static method
greeting::welcome();
?>
```

Static Properties Code Example:

```
<?php
class pi {
  public static $value = 3.14159;
}

// Get static property
echo pi::$value;
?>
```

Namespaces: This is used to grouping classes that work together for today. They allow the same name for multiple classes. For example, we can have a class named Table that contains row,column,data. Also we can have a class named Table that has wood, color e.t.c. A namespace should be declared in the top of the PHP document.

Namespace declaration Example:

```
<?php
namespace Html;
?>
```