

PHP Classes and Objects

In this tutorial you will learn how to write code in object-oriented style in PHP.

What is Object Oriented Programming

Object-Oriented Programming (OOP) is a programming model that is based on the concept of classes and objects. As opposed to procedural programming where the focus is on writing procedures or functions that perform operations on the data, in object-oriented programming the focus is on the creations of objects which contain both data and functions together.

Object-oriented programming has several advantages over conventional or procedural style of programming. The most important ones are listed below:

- It provides a clear modular structure for the programs.
- It helps you adhere to the "don't repeat yourself" (DRY) principle, and thus make your code much easier to maintain, modify and debug.
- It makes it possible to create more complicated behavior with less code and shorter development time and high degree of reusability.

The following sections will describe how classes and objects work in PHP.

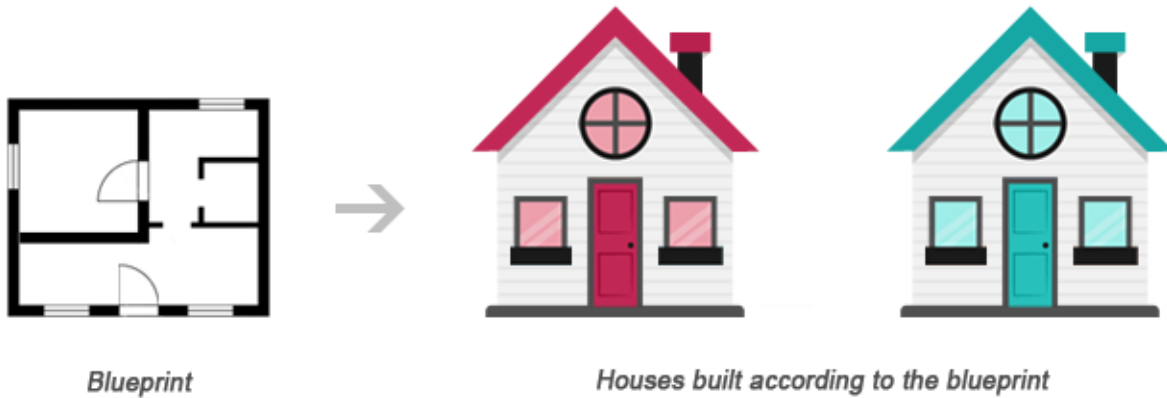
Tip: The idea behind Don't Repeat Yourself (DRY) principle is reducing the repetition of code by abstracting out the code that are common for the application and placing them at a single place and reuse them instead of repeating it.

Understanding Classes and Objects

Classes and objects are the two main aspects of object-oriented programming. A class is a self-contained, independent collection of variables and functions which work together to perform one or more specific tasks, while objects are individual instances of a class.

A class acts as a template or blueprint from which lots of individual objects can be created. When individual objects are created, they inherit the same generic properties and behaviors, although each object may have different values for certain properties.

For example, think of a class as a blueprint for a house. The blueprint itself is not a house, but is a detailed plan of the house. While, an object is like an actual house built according to that blueprint. We can build several identical houses from the same blueprint, but each house may have different paints, interiors and families inside, as shown in the illustration below.



A class can be declared using the `class` keyword, followed by the name of the class and a pair of curly braces (`{}`), as shown in the following example.

Let's create a PHP file named `Rectangle.php` and put the following example code inside it so that our class code should be separated from rest of the program. We can then use it wherever it's needed by simply including the `Rectangle.php` file.

Example	Download
<pre><?php class Rectangle { // Declare properties public \$length = 0; public \$width = 0; // Method to get the perimeter public function getPerimeter(){ return (2 * (\$this->length + \$this->width)); } // Method to get the area public function getArea(){ return (\$this->length * \$this->width); } }</pre>	

The `public` keyword before the properties and methods in the example above, is an [access modifier](#), which indicates that this property or method is accessible from anywhere. We will learn more about this a little later in this chapter.

Note: Syntactically, variables within a class are called *properties*, whereas functions are called *methods*. Also class names conventionally are written in PascalCase i.e. each

concatenated word starts with an uppercase letter (e.g. MyClass).

Once a class has been defined, objects can be created from the class with the `new` keyword. Class methods and properties can directly be accessed through this object instance.

Create another PHP file name test.php and put the following code inside it.

Example	Run this code »
<pre><?php // Include class definition require "Rectangle.php"; // Create a new object from Rectangle class \$obj = new Rectangle; // Get the object properties values echo \$obj->length; // Output: 0 echo \$obj->width; // Output: 0 // Set object properties values \$obj->length = 30; \$obj->width = 20; // Read the object properties values again to show the change echo \$obj->length; // Output: 30 echo \$obj->width; // Output: 20 // Call the object methods echo \$obj->getPerimeter(); // Output: 100 echo \$obj->getArea(); // Output: 600 ?></pre>	

The arrow symbol (`->`) is an OOP construct that is used to access contained properties and methods of a given object. Whereas, the pseudo-variable `$this` provides a reference to the calling object i.e. the object to which the method belongs.

The real power of object oriented programming becomes evident when using multiple instances of the same class, as shown in the following example:

Example	Run this code »
<pre><?php // Include class definition</pre>	

```
require "Rectangle.php";

// Create multiple objects from the Rectangle class
$obj1 = new Rectangle;
$obj2 = new Rectangle;

// Call the methods of both the objects
echo $obj1->getArea(); // Output: 0
echo $obj2->getArea(); // Output: 0

// Set $obj1 properties values
$obj1->length = 30;
$obj1->width = 20;

// Set $obj2 properties values
$obj2->length = 35;
$obj2->width = 50;

// Call the methods of both the objects again
echo $obj1->getArea(); // Output: 600
echo $obj2->getArea(); // Output: 1750
?>
```

As you can see in the above example, calling the `getArea()` method on different objects causes that method to operate on a different set of data. Each object instance is completely independent, with its own properties and methods, and thus can be manipulated independently, even if they're of the same class.

Using Constructors and Destructors

To make the object-oriented programming easier, PHP provides some magic methods that are executed automatically when certain actions occur within an object.

For example, the magic method `__construct()` (known as *constructor*) is executed automatically whenever a new object is created. Similarly, the magic method `__destruct()` (known as *destructor*) is executed automatically when the object is destroyed. A destructor function cleans up any resources allocated to an object once the object is destroyed.

Example

[Run this code »](#)

```
<?php
class MyClass
{
```

```

// Constructor
public function __construct(){
    echo 'The class "' . __CLASS__ . '" was initiated!<br>';
}

// Destructor
public function __destruct(){
    echo 'The class "' . __CLASS__ . '" was destroyed.<br>';
}
}

// Create a new object
$obj = new MyClass;

// Output a message at the end of the file
echo "The end of the file is reached.";
?>

```

The PHP code in the above example will produce the following output:

```

The class "MyClass" was initiated!
The end of the file is reached.
The class "MyClass" was destroyed.

```

A destructor is called automatically when a script ends. However, to explicitly trigger the destructor, you can destroy the object using the PHP `unset()` function, as follow:

Example

[Run this code »](#)

```

<?php
class MyClass
{
    // Constructor
    public function __construct(){
        echo 'The class "' . __CLASS__ . '" was initiated!<br>';
    }

    // Destructor
    public function __destruct(){
        echo 'The class "' . __CLASS__ . '" was destroyed.<br>';
    }
}

// Create a new object

```

```
$obj = new MyClass;

// Destroy the object
unset($obj);

// Output a message at the end of the file
echo "The end of the file is reached.";
?>
```

Now, the PHP code in the above example will produce the following output:

```
The class "MyClass" was initiated!
The class "MyClass" was destroyed.
The end of the file is reached.
```

Tip: PHP automatically clean up all resources allocated during execution when the script is finished, e.g. closing database connections, destroying objects, etc.

Note: The `__CLASS__` is a [magic constant](#) which contains the name of the class in which it is occur. It is empty, if it occurs outside of the class.

Extending Classes through Inheritance

Classes can inherit the properties and methods of another class using the `extends` keyword. This process of extensibility is called inheritance. It is probably the most powerful reason behind using the object-oriented programming model.

Example

[Run this code »](#)

```
<?php
// Include class definition
require "Rectangle.php";

// Define a new class based on an existing class
class Square extends Rectangle
{
    // Method to test if the rectangle is also a square
    public function isSquare(){
        if($this->length == $this->width){
            return true; // Square
        } else{
```

```

        return false; // Not a square
    }
}

// Create a new object from Square class
$obj = new Square;

// Set object properties values
$obj->length = 20;
$obj->width = 20;

// Call the object methods
if($obj->isSquare()){
    echo "The area of the square is ";
} else{
    echo "The area of the rectangle is ";
};
echo $obj->getArea();
?>

```

The PHP code in the above example will produce the following output:

```
The area of the square is 400
```

As you can see in the above example, even though the class definition of Square doesn't explicitly contain `getArea()` method nor the `$length` and `$width` property, instances of the Square class can use them, as they inherited from the parent Rectangle class.

Tip: Since a child class is derived from a parent class, it is also referred to as a derived class, and its parent is called the base class.

Controlling the Visibility of Properties and Methods

When working with classes, you can even restrict access to its properties and methods using the *visibility keywords* for greater control. There are three visibility keywords (from most visible to least visible): `public`, `protected`, `private`, which determines how and from where properties and methods can be accessed and modified.

- **public** — A public property or method can be accessed anywhere, from within the class and outside. This is the default visibility for all class members in PHP.
- **protected** — A protected property or method can only be accessed from within the class itself or in child or inherited classes i.e. classes that extends that class.

- **private** — A private property or method is accessible only from within the class that defines it. Even child or inherited classes cannot access private properties or methods.

The following example will show you how this visibility actually works:

Example	Download
<pre data-bbox="207 336 1388 1837"><?php // Class definition class Automobile { // Declare properties public \$fuel; protected \$engine; private \$transmission; } class Car extends Automobile { // Constructor public function __construct(){ echo 'The class "' . __CLASS__ . '" was initiated!
'; } } // Create an object from Automobile class \$automobile = new Automobile; // Attempt to set \$automobile object properties \$automobile->fuel = 'Petrol'; // ok \$automobile->engine = '1500 cc'; // fatal error \$automobile->transmission = 'Manual'; // fatal error // Create an object from Car class \$car = new Car; // Attempt to set \$car object properties \$car->fuel = 'Diesel'; // ok \$car->engine = '2200 cc'; // fatal error \$car->transmission = 'Automatic'; // undefined ?></pre>	

Static Properties and Methods

In addition to the visibility, properties and methods can also be declared as `static`, which makes them accessible without needing an instantiation of the class. Static properties and methods can be accessed using the scope resolution operator (`::`), like this: `ClassName::$property` and `ClassName::method()`.

A property declared as static cannot be accessed via the object of that class though a static method can be, as demonstrated in the following example:

Example	Download
<pre><?php // Class definition class HelloClass { // Declare a static property public static \$greeting = "Hello World!"; // Declare a static method public static function sayHello(){ echo self::\$greeting; } } // Attempt to access static property and method directly echo HelloClass::\$greeting; // Output: Hello World! HelloClass::sayHello(); // Output: Hello World! // Attempt to access static property and method via object \$hello = new HelloClass; echo \$hello->greeting; // Strict Warning \$hello->sayHello(); // Output: Hello World! ?></pre>	

The keyword `self` in the above example means "the current class". It is never preceded by a dollar sign (`$`) and always followed by the `::` operator (e.g. `self::$name`).

The `self` keyword is different from the `this` keyword which means "the current object" or "the current instance of a class". The `this` keyword is always preceded by a dollar sign (`$`) and followed by the `->` operator (e.g. `$this->name`).

Note: Since static methods can be called without an instance of a class (i.e. object), the pseudo-variable `$this` is not available inside the method declared as static.