CHAPTER 9

Class

A *class* is a template used to create objects. To define one, the `class` keyword is used, followed by a name and a code block. The naming convention for classes is mixed case, meaning that each word should be initially capitalized.

```
class MyRectangle {}
```

The class body can contain properties and methods. *Properties* are variables that hold the state of the object, whereas *methods* are functions that define what the object can do. Properties are also known as *fields* or *attributes* in other languages. In PHP, they need to have an explicit access level specified. In the following, the `public` access level is used, which gives unrestricted access to the property.

```
class MyRectangle
{
  public $x, $y;
  function newArea($a, $b) { return $a * $b; }
}
```

To access members from inside the class, the `$this` pseudo variable is used along with the single arrow operator (`->`). The `$this` variable is a reference to the current instance of the class and can only be used within an object context. Without it, `$x` and `$y` would just be seen as local variables.

```
class MyRectangle
{
  public $x, $y;

  function newArea($a, $b)
  {
    return $a * $b;
  }
  function getArea()
  {
    return $this->newArea($this->x, $this->y);
  }
}
```

Instantiating an Object

To use a class's members from outside the enclosing class, an object of the class must first be created. This is done using the new keyword, which creates a new object or instance.

```
$r = new MyRectangle(); // object instantiated
```

The object contains its own set of properties, which can hold values that are different from those of other instances of the class. As with functions, objects of a class may be created even if the class definition appears further down in the script file.

```
$r = new MyDummy(); // ok
class MyDummy {};
```

Accessing Object Members

To access members that belong to an object, the single arrow operator (->) is needed. It can be used to call methods or to assign values to properties.

```
$r->x = 5;
$r->y = 10;
$r->getArea(); // 50
```

Another way to initialize properties is to use initial property values.

Initial Property Values

If a property needs to have an initial value, a clean way is to assign the property at the same time that it is declared. This initial value is then set when the object is created. Assignments of this kind must be a constant expression. It cannot, for example, be a variable or a mathematical expression.

```
class MyRectangle
{
  public $x = 5, $y = 10;
}
```

Constructor

A class can have a constructor, which is a special method used to initialize (construct) the object. This method provides a way to initialize properties, which is not limited to constant expressions. In PHP, the constructor starts with two underscores followed by the word `construct`. Methods like these are known as *magic methods*.

```
class MyRectangle
{
  public $x, $y;

  function __construct()
  {
    $this->x = 5;
    $this->y = 10;
    echo "Constructed";
  }
}
```

When a new instance of this class is created, the constructor is called, which in this example sets the properties to the specified values. Note that any initial property values are set before the constructor is run.

```
$r = new MyRectangle(); // "Constructed"
```

Since this constructor takes no arguments, the parentheses may optionally be left out.

```
$r = new MyRectangle; // "Constructed"
```

Just as any other method, the constructor can have a parameter list. It can be used to set the property values to the arguments passed when the object is created.

```
class MyRectangle
{
  public $x, $y;

  function __construct($x, $y)
  {
    $this->x = $x;
    $this->y = $y;
  }
}

$r = new MyRectangle(5,10);
```

Destructor

In addition to the constructor, classes can also have a destructor. This magic method starts with two underscores followed by the word destruct. It is called as soon as there are no more references to the object, before the object is destroyed by the PHP garbage collector.

```
class MyRectangle
{
  // ...
  function __destruct() { echo "Destructed"; }
}
```

To test the destructor, the unset function can manually remove all references to the object.

```
unset($r); // "Destructed"
```

Bear in mind that the object model was completely rewritten in PHP 5. Therefore, many features of classes, such as destructors, do not work in earlier versions of the language.

Case Sensitivity

Whereas variable names are case sensitive, class names in PHP are case insensitive—as are function names, keywords, and built-in constructs such as echo. This means that a class named MyClass can also be referenced as myclass or MYCLASS.

```
class MyClass {}
$o1 = new myclass(); // ok
$o2 = new MYCLASS(); // ok
```

Object Comparison

When using the "equal to" operator (==) on objects, these objects are considered equal if the objects are instances of the same class and their properties have the same values and types. In contrast, the strict "equal to" operator (===) returns true only if the variables refer to the same instance of the same class.

```
class Flag
{
  public $flag = true;
}
```

```
$a = new Flag();
$b = new Flag();

$c = ($a == $b);  // true (same values)
$d = ($a === $b); // false (different instances)
```

Anonymous Classes

Support for anonymous classes were introduced in PHP 7. Such a class is useful in place of a named class when only a single, throwaway object is needed.

```
$obj = new class {};
```

The implementation of the anonymous class, and the object created from it, are no different from a named class; for instance, they can use constructors in the same way as any named class.

```
$o = new class('Hi')
{
  public $x;
  public function __construct($a)
  {
    $this->x = $a;
  }
};

echo $o->x; // "Hi";
```

Closure Object

Anonymous functions in PHP are also closures, as they have the ability to capture a context from outside of the function's scope. In addition to variables, this context can also be an object's scope. This creates a so-called *closure object*, which has access to the properties of that object. An object closure is made using the `bindTo` method. This method accepts two arguments: the object to which the closure is bound and the class scope that it is associated with. To access non-public members (private or protected), the name of the class or object must be specified as the second argument.

```
class C { private $x = 'Hi'; }

$getC = function() { return $this->x; };
$getX = $getC->bindTo(new C, 'C');
echo $getX(); // "Hi"
```

This example uses two closures. The first closure, $getC, defines the method for retrieving the property. The second closure, $getX, is a duplicate of $getC, to which the object and class scope has been bound. PHP 7 simplified this by providing a shorthand—a better-performing way of temporarily binding and then calling a closure in the same operation.

```
// PHP 7+ code
$getX = function() { return $this->x; };
echo $getX->call(new C); // "Hi"
```