Object Oriented Programming in PHP5

A WebApp Tutorial

Adrian Giurca
Chair of Internet Technology, Institute for Informatics

October 15, 2006

Revision History
Revision 1 Sept 20, 2005
Revision 2 Sept 13, 2006

Table of Contents

1. Basic PHP Constructs for OOP ...................................................................................................... 2
2. Advanced OOP Features .............................................................................................................. 4
   2.1. Public, Private, and Protected Members ................................................................................ 4
   2.2. Interfaces ........................................................................................................................ 5
   2.3. Constants ........................................................................................................................ 5
   2.4. Abstract Classes ............................................................................................................... 5
   2.5. Simulating class functions .................................................................................................. 5
   2.6. Calling parent functions ..................................................................................................... 6
      2.6.1. Calling parent constructors ...................................................................................... 6
      2.6.2. The special name parent ......................................................................................... 7
   2.7. Serialization .................................................................................................................... 7
   2.8. Introspection Functions ..................................................................................................... 8
3. OOP Style in PHP. The PEAR Coding Style ................................................................................... 12
   3.1. Indenting, whitespace, and line length ................................................................................ 13
   3.2. Formatting control structures ............................................................................................ 13
   3.3. Formatting functions and function calls ............................................................................... 14
   3.4. PHPDoc ....................................................................................................................... 14
Bibliography ............................................................................................................................... 15

How "object-oriented" is PHP?

Let's try an answer:

- **Single inheritance.** PHP allows a class definition to inherit from another class, using the `extends` clause. Both member variables and member functions are inherited.

- **Multiple inheritance.** PHP offers no support for multiple inheritance and no notion of interface inheritance as in Java. Each class inherits from, at most, one parent class (though a class may implement many interfaces).

- **Constructors.** Every class can have one constructor function, which in PHP is called `__construct()`. Note that there are two underscore characters at the front of that function name. Constructors of parent classes are not automatically called but must be invoked explicitly.

- **Destructors.** PHP supports explicit destructor functions as of version 5. The destructor function of a class is always called `__destruct()`.

- **Encapsulation/access control.** PHP supports public, private, and protected properties and methods as of version 5.
Polymorphism/overloading. PHP supports polymorphism in the sense of allowing instance of subclasses to be used in place of parent instances. The correct method will be dispatched to at runtime. There is no support for method overloading, where dispatch happens based on the method's signature-each class only has one method of a given name.

Static (or class) functions. PHP offers static properties and static methods as of version 5. It is also possible to call methods via the `Classname::function()` syntax.

Introspection. PHP offers a wide variety of functions here, including the capability to recover class names, methods names, and properties names from an instance.

In the next section, we cover the basic PHP syntax for OOP from the ground up, with some simple examples.

1. Basic PHP Constructs for OOP

The general form for defining a new class in PHP is as follows:

```php
class MyClass extends MyParent {
    var $var1;
    var $var2 = "constant string";
    function myfunc ($arg1, $arg2) {
        //...
    }
    //...
}
```

As an example, consider the simple class definition in the listing below, which prints out a box of text in HTML:

```php
class TextBoxSimple {
    var $body_text = "my text";
    function display() {
        print("<table><tr><td>$this->body_text</td></tr></table>");
    }
}
```

In general, the way to refer to a property from an object is to follow a variable containing the object with `->` and then the name of the property. So if we had a variable `$box` containing an object instance of the class `TextBox`, we could retrieve its `body_text` property with an expression like:

```php
$text_of_box = $box->body_text;
```

Notice that the syntax for this access does not put a `$` before the property name itself, only the `$this` variable.

After we have a class definition, the default way to make an instance of that class is by using the `new` operator.

```php
$box = new TextBoxSimple;
$box->display();
```

The correct way to arrange for data to be appropriately initialized is by writing a constructor function-a special function called `__construct()`, which will be called automatically whenever a new instance is created.

```php
class TextBox {
    var $bodyText = "my default text";
    // Constructor function
    function __construct($newText) {
        $this->bodyText = $newText;
    }
    function display() {
        print("<table><tr><td>$this->bodyText</td></tr></table>");
    }
}
```

// creating an instance
PHP class definitions can optionally inherit from a superclass definition by using the extends clause. The effect of inheritance is that the subclass has the following characteristics:

- Automatically has all the property declarations of the superclass.
- Automatically has all the same methods as the superclass, which (by default) will work the same way as those functions do in the superclass.

In addition, the subclass can add on any desired properties or methods simply by including them in the class definition in the usual way.

class TextBoxHeader extends TextBox{
    var $headerText;
    // CONSTRUCTOR
    function __construct($newHeaderText, $newBodyText) {
        $this->headerText = $newHeaderText;
        $this->bodyText = $newBodyText;
    }
    // MAIN DISPLAY FUNCTION
    function display() {
        $header_html = $this->make_header($this->headerText);
        $body_html = $this->make_body($this->bodyText);
        print("<table><tr><td>
        
        
        
        
        
        </td></tr><tr><td>
        
        
        
        
        
        </td></tr></table>
        
    }
    // HELPER FUNCTIONS
    function make_header ($text) {
        return($text);
    }
    function make_body ($text) {
        return($text);
    }
}

Function definitions in subclasses override definitions with the same name in superclasses. This just means that the overriding definition in the more specific class takes precedence and will be the one actually executed.

Before we move onto the more advanced features of PHP’s version of OOP, it’s important to discuss issues of scope—that is, which names are meaningful in what way to different parts of our code. It may seem as though the introduction of classes, instances, and methods have made questions of scope much more complicated. Actually, though, there are only a few basic rules we need to add to make OOP scope sensible within the rest of PHP:

- Names of properties and methods are never meaningful to calling code on their own—they must always be reached via the -> construct. This is true both outside the class definition and inside methods.
- The names visible within methods are exactly the same as the names visible within global functions—that is, methods can refer freely to other global functions, but can’t refer to normal global properties unless those properties have been declared global inside the method definition.

These rules, together with the usual rules about variable scope in PHP, are respected in the intentionally confusing example in the listing below. What number would you expect that code to print when executed?

```php
$myGlobal = 3;
function myFunction ($myInput) {
    global $myGlobal;
    return($myGlobal * $myInput);
}
class MyClass {
    var $myProperty;
```
End of ebook preview

Download the full PDF tutorial from the link below:

Click Here