Introduction to PHP5

By Luis Argerich – PHPBuilder.com

PHP5 is not yet official but the development versions are already usable (and unstable!) so we can start to learn and practice the new features of the upcoming version of PHP. In this article I will focus in three major new features of PHP5:.

- The new object model
- Exceptions
- Namespaces

First of all a couple of disclaimers:

- Some of the techniques described in this article can be done with PHP4, but are presented anyway to make the whole article more readable.
- Some of the features described in this article may change in the final release of PHP5.

PHP5 has not been released yet and I don't know when that will be but you can already try and investigate the new features of the language by downloading and installing a PHP5 development version from http://snaps.php.net. There you will find Linux and Windows versions of PHP5 ready to be used. Installation proceeds as in any normal PHP distribution so go there and grab a brand new toy.

The new object model

The PHP5 object model has been revamped adding a lot of features that will give PHP5 a Java flavour. The following section of this article will describe this new object model and some quick examples that you can use as a starting point for your experiments.

* Constructors and destructors
* Objects as references
* Cloning objects
* Private, Public and Protected keywords
* Interfaces
* Abstract classes
* __call
* __set and __get
* Static members

Constructors and destructors

In PHP4 constructors are named as the class and there're no destructors.
In PHP5 the constructor for a class is called __construct and the destructor is called __destruct.

Example 1: Constructors and destructors

```
<?php
class foo {
  var $x;

  function __construct($x) {
```

```
    $this->x = $x;
  }

  function display() {
    print($this->x);
  }

  function __destruct() {
    print("bye bye");
  }
}

$o1 = new foo(4);
$o1->display();
?>
```

As you will see the destructor will be called just before the object is eliminated.

Objects as references

In PHP4 as you may already know variables are passed to functions/methods by value (a copy is passed) unless you use the '&' symbol in the function declaration indicating that the variable will be passed as a reference. In PHP5 objects will be passed always as references. Object assignation is also done by reference.

Example 2: Objects as references

```
<?php
class foo {
  var $x;

  function setX($x) {
    $this->x = $x;
  }

  function getX() {
    return $this->x;
  }
}

$o1 = new foo;
$o1->setX(4);
$o2 = $o1;
$o1->setX(5);
if($o1->getX() == $o2->getX()) print("Oh my god!");
?>
```

Cloning objects

Since objects are passed and assigned as references you need some way to create a copy of an object. Enter the __clone method.

Example 3: Cloning objects

```
<?php
class foo {
  var $x;

  function setX($x) {
    $this->x = $x;
  }

  function getX() {
    return $this->x;
  }
}

$o1 = new foo;
$o1->setX(4);
$o2 = $o1->__clone();
$o1->setX(5);

if($o1->getX() != $o2->getX()) print("Copies are independant");
?>
```

Cloning is ok in programming languages, don't feel guilty ;-)

Private, Public and Protected keywords

In PHP4 all the methods and variables in an Object can be accessed from outside the object - this can be rephrased as methods and variables are always public. PHP5 introduces 3 modifiers to control the access to variables and methods: Public, Protected and Private.

Public: The method/variable can be accessed from outside the class.
Private: Only methods in the same class can access private methods or variables.
Protected: Only methods in the same class or derived classes can access proteted methods or variables.

Example 4: Public, protected and private

```
<?php
class foo {
  private $x;

  public function public_foo() {
    print("I'm public");
  }

  protected function protected_foo() {
    $this->private_foo(); //Ok because we are in the same class we
can call private methods
    print("I'm protected");
  }

  private function private_foo() {
```

```
    $this->x = 3;
    print("I'm private");
  }
}

class foo2 extends foo {
  public function display() {
    $this->protected_foo();
    $this->public_foo();
    // $this->private_foo();  // Invalid! the function is private in
the base class
  }
}

$x = new foo();
$x->public_foo();
//$x->protected_foo();  //Invalid cannot call protected methods
outside the class and derived classes
//$x->private_foo();    //Invalid private methods can only be used
inside the class

$x2 = new foo2();
$x2->display();
?>
```

Design tip: Variables should always be private, accessing variables is not a good OOP practice, it is always better to provide methods to get/set the variables.

Interfaces

As you know PHP4 supports inheritance using the "class foo extends parent" syntax. In PHP4 AND in PHP5 a class can only extend one class so multiple inheritance is not supported. An interface is a class that does not implement any methods it only defines the method names and the parameters the methods have. Classes can then 'implement' as many interfaces as needed indicating that the class will implement the methods defined in the interface.

Example 5: Interfaces

```
<?php
interface displayable {
  function display();
}

interface printable {
  function doprint();
}

class foo implements displayable,printable {
  function display() {
    // code
  }
```

```
  function doprint() {
    // code
  }
}
?>
```

This is very useful to make your code easier to read and understand, reading the class declaration you will know that the class implements the displayable and printable interfaces so the class must have a display() method and the class must have a print() method, no matter how they are implemented you know you can call the methods by just reading the class declaration.

Abstract classes

An abstract class is a class that cannot be instantiated.
An abstract class can, as any normal superclass, define methods and variables.
Abstract classes can also define asbtract methods, methods that are not implemented in the abstract class
but must be implemented in derived classes.

Example 6: Asbtract classes

```
<?php
abstract class foo {
  protected $x;

  abstract function display();

  function setX($x) {
    $this->x = $x;
  }
}


class foo2 extends foo {
  function display() {
    // Code
  }
}
?>
```

__call

In PHP5 the special __call() method can be implemented in a class as a "catch all" method for methods not implemented in the class. If you call a method not accessible or a method that doesn't exist the __call method (if defined) will be called.

Example 7: __call

```
<?php
class foo {
```

```
  function __call($name,$arguments) {
    print("Did you call me? I'm $name!");
  }
}

$x = new foo();
$x->doStuff();
$x->fancy_stuff();
?>
```

This special method can be used to implement method overloading because you can examine the arguments and call a private ad-hoc method depending on the arguments passed, example.

Exampe 8: Overloading methods with __call

```
<?php
class Magic {

  function __call($name,$arguments) {
    if($name=='foo') {
      if(is_int($arguments[0])) $this->foo_for_int($arguments[0]);
      if(is_string($arguments[0])) $this->foo_for_string($arguments
[0]);
    }
  }

  private function foo_for_int($x) {
    print("oh an int!");
  }

  private function foo_for_string($x) {
    print("oh a string!");
  }
}

$x = new Magic();
$x->foo(3);
$x->foo("3");
?>
```

__set and __get

And this gets even fancier, the __set and __get methods can be implemented as catch-all methods for accessing or setting variables not defined (or not accessible).

Example 9: __set and __get

```
<?php
class foo {

  function __set($name,$val) {
    print("Hello, you tried to put $val in $name");
```

```
  }

  function __get($name) {
    print("Hey you asked for $name");
  }
}

$x = new foo();
$x->bar = 3;
print($x->winky_winky);
?>
```

Type hinting

In PHP5 you will be able to indicate that a method must receive an object of some class as an argument.

Example 10: type hinting

```
<?php
class foo {
  // code ...
}

class bar {
  public function process_a_foo(foo $foo) {
   // Some code
  }
}

$b = new bar();
$f = new foo();
$b->process_a_foo($f);
?>
```

As you can see the class name can be indicated before the argument name to make PHP5 know that $foo should be an object of the class foo.

Static members

Static members and static methods can be used to implement terms known in OOP as "class methods" and "class variables".

A "class method" is a method that can be called without creating an instance of the class.
A "class variable" is a variable that can be accessed without creating an instance of the class (and without needing a get method)

Example 11: class methods and class variables

```
<?php
class calculator {
  static public $pi = 3.14151692;
```

```
  static public function add($x,$y) {
    return $x + $y;
  }
}

$s = calculator::$pi;
$result = calculator::add(3,7);
print("$result");
?>
```

Exceptions

Exceptions are an accepted way to handle errors and unexpected conditions in languages such as Java and C++, PHP5 incorporates exceptions implementing the "try" and "catch" hooks.

Example 12: Exceptions

```
<?php
class foo {

  function divide($x,$y) {
    if($y==0) throw new Exception("cannot divide by zero");
    return $x/$y;
  }
}

$x = new foo();

try {
  $x->divide(3,0);
} catch (Exception $e) {
    echo $e->getMessage();
    echo "\n<br />\n";
    // Some catastrophic measure here
}
?>
```

As you can see you use "try" to denote a block of code where exceptions will be handled by the "catch" clause at the end of the block. In "catch" you should implement whatever you need as your error handling policy. This leads to cleaner code with only one point for error handling.

Defining your own exceptions

You can define custom exceptions to handle unexpected problems in your programs. You only need to extend the Exception class implementing a constructor and the getMessage method.

Example 13: Custom exceptions

```
<?php
class WeirdProblem extends Exception {

   private $data;
```

```
    function WeirdProblem($data) {
         parent::exception();
         $this->data = $data;
     }

     function getMessage() {
         return $this->data . " caused a weird exception!";
     }
}
?>
```

Then use throw new WeirdProblem($foo) to throw your exception, if the exception is produced inside a try{} block then PHP5 will jump into the "catch" section for exception handling.

Namespaces

Namespaces can be used to group classes or functions for convenience.

Example 14: Namespaces

```
<?php
namespace Math {

  class Complex {
    //...code...
    function __construct() {
      print("hey");
    }
  }
}

$m = new Math::Complex();
?>
```

Note how namespaces can be used to qualify the class that should be created. As a practical example you may want to declare the same class name in different namespaces to do different things (but with the same interface).

Luis Argerich