Object Oriented Programming in PHP

CMPS 183 - Hypermedia and the Web
May 1, 2006

Mark Slater
mslater@soe.ucsc.edu

Object Oriented Basics

- OO programs consist of hierarchies and webs of Classes.
- Classes contain data and functions that act on that data.
- Classes can be extended through inheritance (hierarchy).
- Classes can contain other classes (web).

A Sample PHP Class

```php
<?php

class ImgTagGenerator
{
    function makeImgTag( $location )
    {
        if( isset( $location ) && ( strlen( location ) > 0 ) )
        {
            echo "<IMG src=\"" . $location . "\">\n";
        }
        else
        {
            echo " <!--Image location not specified!-->\n";
        }
    }
}

?>
```

$this

- PHP reserves the variable `$this` to refer to the object whose context the function was invoked from.
- Usually `$this` refers to the object the function belongs to.

$this

```php
<?php

class A
{
    public $num;

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

    function getNum()
    {
        return( $this->num );
    }
}
```

```php
class B
{
    public $num;

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

    function getNum()
    {
        return( $this->num );
    }

    function getANum()
    {
        return( A::getNum() );
    }
}

$a = new A( 5 );
$b = new B( "six" );

echo "A num = " . $a->getNum() . "<br>\n";
echo "B num = " . $b->getNum() . "<br>\n";
echo "A num from B = " . $b->getANum() . "<br>\n";
?>
```

$this

- The context in the `getANum()` function is B's.
- When A's `getNum()` function is called statically, the context does not change.

A num = 5
B num = six
A num from B = six

```
class B
{
    ...

    function getANum()
    {
        return( A::getNum() );
    }
}
```

Creating and Assigning Objects

- Just like Java, C++, and other OO languages, PHP has a `new` operator that creates instances of classes.
- Assigning a variable containing an object to another variable makes a copy.
- Assigning a reference to another variable does not.

Creating and Assigning Objects

```php
<?php

class A
{
    public $num;

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

    function getNum()
    {
        return( $this->num );
    }
}

$orig = new A( 5 );
$copy = $orig;
$ref =& $orig;

$orig = new A( 10 );

echo "<font size=18>";
echo "orig num = " . $orig->getNum() . "<br>\n";
echo "copy num = " . $copy->getNum() . "<br>\n";
echo "ref num = " . $ref->getNum() . "<br>\n";
echo "</font>";

?>
```

orig num = 10
copy num = 5
ref num = 10

Creating and Assigning Objects

```php
<?php

class A
{
    public $num;

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

    function getNum()
    {
        return( $this->num );
    }
}

$orig = new A( 5 );
$copy = $orig;
$ref =& $orig;

$orig->num = 10;

echo "orig num = " . $orig->getNum() . "<br>\n";
echo "copy num = " . $copy->getNum() . "<br>\n";
echo "ref num = " . $ref->getNum() . "<br>\n";

?>
```

orig num = 10
copy num = 10
ref num = 10

Assigning Objects: What Just Happened

- In the first example, the object that `$orig` pointed at changed.
- `$ref` was set to refer to whatever object or value that `$orig` points at.
- `$copy` still pointed to the original object.

```
$orig = new A( 5 );
$copy = $orig;
$ref =& $orig;

$orig = new A( 10 );
```

Assigning Objects: What Just Happened

- In the second example, a value was changed in the object that `$orig` points to.
- Since all three variables point to that object, they all got the new value.

```
$orig = new A( 5 );
$copy = $orig;
$ref =& $orig;

$orig->num = 10;
```

Assigning Objects: Summery

```
$orig = new A( 5 );
$copy = $orig;
$ref =& $orig;

$orig = new A( 10 );
```

orig num = 10
copy num = 5
ref num = 10

```
$orig = new A( 5 );
$copy = $orig;
$ref =& $orig;

$orig->num = 10;
```

orig num = 10
copy num = 10
ref num = 10

```
$orig = 5;
$copy = $orig;
$ref =& $orig;

$orig = 10;
```

orig = 10
copy = 5
ref = 10

Extending a class

- You can extend a class using the `extends` keyword.
- Sub-classes inherit the functions and member variables of their parent class.
- Multiple-inheritance is not allowed.
- Overriding methods and members is possible if they are not `final`.

Extending a class

```php
<?php

class A
{
    public $num;

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

    function getNum()
    {
        return( $this->num );
    }
}

class Double extends A
{
    function getNum()
    {
        return( parent::getNum() * 2 );
    }
}

$double = new Double( 5 );

echo "double getNum = " . $double->getNum() . "<br>\n";

?>
```

```
double getNum = 10
```

Constructors

- Just like Java and C++, you can create a constructor for your class.
- In PHP, you can only have one constructor.
- No methods can be overloaded.
- You must explicitly call the parent constructor.

Constructors

```php
<?php

class ImgTagGenerator
{
    public $location;

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

    function makeImgTag()
    {
        echo "<IMG src=\"" . $this->location . "\">\n";
    }
}
```

```php
class ImgWithAltTagGenerator
    extends ImgTagGenerator
{
    public $alt;

    function __construct( $location, $alt )
    {
        parent::__construct( $location );
        $this->alt = $alt;
    }

    function makeImgTag()
    {
        echo "<IMG alt=\"" . $this->alt .
            "\" src=\"" . $this->location . "\">\n";
    }
}

?>
```

Destructors

- PHP also has destructors for cleanup.
- They are automatically called when an object is explicitly destroyed or all references have been removed (like garbage collection).
- Again, you must explicitly call the parent destructor.

Destructors

```
function __destruct()
{
    // Do cleanup
}
```

Constructors and Destructors

- For objects created with `new`, the `__construct()` and `__destruct()` functions must be `public`.
- You cannot use an object until its constructor has finished executing.

Visibility: Public, Protected, and Private

- `public` variables can be accessed anywhere, by anyone.
- `protected` variables can only be accessed by the class and its sub-classes.
- `private` variables can only be accessed by the class that defines them.

Visibility: Public, Protected, and Private

- Just like other OO languages, you should default to `private` and use protected accessors for subclasses.
- Only constant values should ever be `public` or `protected`.
- If your class is more like a C `struct`, `public` is appropriate.

Visibility in Functions

- Functions can also be given visibilities of `public`, `protected`, or `private`.
- Functions without a visibility declaration are `public`.
- Explicitly declare the visibility.
- Default to `private` until a subclass needs the function.

Scope Resolution Operator

- Also known as the double colon — : :
- Allows you to access static, constant, and overridden variables or functions.
- Already seen it in constructors in destructors.

Scope Resolution Operator

```php
<?php

class ClassWithConst
{
    const SOME_CONST = 'some const';
}

class ClassWithStatic extends ClassWithConst
{
    private static $someStatic = "a static variable";

    public static function staticFunc()
    {
        echo "parent has ‘" . parent::SOME_CONST
            . "’ and I have ‘" . self::$someStatic
            . "’<br>\n";
    }
}

ClassWithStatic::staticFunc();

?>
```

parent has 'some const' and I have 'a static variable'

Static

- Variables and functions can be declared `static`.
- The `static` keyword comes after the visibility keyword.
- You must use the class name to access a static variable; using an object of that class's type will not work.

Static

```
class Foo
{
    public static $someStatic = "Some Static Variable";
    ...
}

$bar = new Foo();

echo Foo::$someStatic; // this works

echo $bar->someStatic; // this fails
echo $bar::someStatic; // this fails
```

Constants

- Constant variables are declared with the keyword `const`.
- Constants do not have visibility modifiers.
- Like `static` variables, `const` variables must be accessed using the class name, not objects of the class type.

Final

- The final keyword prevents subclasses from overriding a function.
- It can also be used to prevent a class from being extended at all.

Abstract Classes and Functions

- If the class is abstract, it can't be instantiated.
- If the class has one or more abstract functions, the class itself must be declared abstract.
- When overriding abstract methods, the visibility must be equal or weaker.

Interfaces

- PHP's interfaces work almost identically to Java's interfaces.
- PHP interfaces cannot declare variables.
  - They can declare constants.
- All interface functions must be `public`.

Interfaces

```php
<?php

interface HtmlTag
{
    public function getHtmlTag();
}

class ImgTag implements HtmlTag
{
    public function getHtmlTag()
    {
        ...
    }
}

?>
```

Iterating Object Variables

- You can use a `foreach` statement to iterate the visible variables of an object.
  - Inside an object, that includes all its `private` variables, its and its parent `protected` variables, and all `public` variables in the hierarchy.

Iterating Object Variables

```php
<?php

class Superclass
{
    public $all = 'parent pub';
    protected $hierarchy = 'parent protected';
    private $me = 'parent private';
}

class Subclass extends Superclass
{
    public $pub = 'child pub';
    protected $prot = 'child protected';
    private $priv = 'child private';

    public function iterateVariables()
    {
        echo "<p>child:<br>\n";
        foreach( $this as $key => $value )
        {
            echo $key . " => " . $value . "<br>";
        }
        echo "</p>\n";
    }
}

$obj = new Subclass();

$obj->iterateVariables();

echo "<p>outside:<br>\n";
foreach( $obj as $key => $value )
{
    echo $key . " => " . $value . "<br>";
}
echo "</p>\n";

?>
```

child:
pub => child pub
prot => child protected
priv => child private
all => parent pub
hierarchy => parent protected

outside:
pub => child pub
all => parent pub

Assigning Objects Redux: Cloning

- Objects can be cloned with the `clone` keyword.
- By default, PHP will do a shallow copy on the original object's variables.
- You can take control of the process by defining a public `__clone()` function to do deep copies, or update transient variables.

Assigning Objects Redux: Cloning

```
class ClassGettingCloned
{
    $private someAggregateObject;

    public function __clone()
    {
        $this->someAggregateObject = clone( $this->someAggregateObject );
    }
}
```

Object Comparison

- Comparison Operator (==)
  - Instances are equal when they have the same variables and values and are of the same type.
- Identity Operator (===)
  - Instances are equal when they are the same instance of the same class.

Specifying Variable Types

- The type of a PHP variable can change with each assignment (dynamic types).
- In functions (in or out of a class), you can supply a "type hint" that must be satisfied by any variable passed in that parameter location.
- Only class types and arrays can be used; primitive types aren't supported.

Using Classes to Standardize Your Site

- Most sites have a standard layout.
- Encapsulate that layout in a class.
- Each page creates an instance of the class, adds content to the instance, and the instance renders that page.

Using Classes to Standardize Your Site

- An example from my work

Questions?