International PHP Conference

Object-oriented programming in PHP

by Dr. Veikko Krypczyk & Elena Bochkor

Principle	Description
Encapsulation	Properties, methods and events of a real object are combined into classes. Concrete occurrences of a class are objects.
Information hiding	You can use just public methods or properties of a class without knowing how they are implemented. For example, objects of the class Car have the method brake. For a user, it is not of interest how the braking process is implemented.
Visibility	Classes, their properties and methods can have different visibility. This indicates which elements of an object are visible to other objects.
Inheritance	Using inheritance, properties and methods of a superclass can be inherited to subclasses. The assignments to upper and lower classes follow the principles of generalization and specialization.
Polymorphism	Objects of different classes sometimes must run the same functions. For example, all engine powered vehicles must be refueled. For this purpose, the refueling method is defined in the abstract vehicle class. Abstract objects can not be used to create objects. They serve only as placeholders of common properties. In the subclasses Cars and Trucks, the refueling method must be redefined concretely, because a car is fueled by petrol and a truck by diesel.

Object-Oriented Programming in PHP

The Basics

Member	Keyword/ Syntax	Example

Simple Class definition	class	class Car{
		  public $tradeMark="Toyota";
		  public function start(){
		   echo "The car is starting";
		  }
		}

Create an instance of a class	new	$myCar = new Car ();
Access operator	->	$myCar = new Car (); echo $myCar -> trademark

Constructor
 	
Parent constructors are not called implicitly if the child class defines a constructor. In order to run a parent constructor, a call to parent::__construct() within the child constructor is required. If the child does not define a constructor then it may be inherited from the parent class just like a normal class method (if it was not declared as private).
	function __construct()	function __construct() {
		  …
		}

Destructor
 	
Parent destructors will not be called implicitly by the engine. In order to run a parent destructor, one would have to explicitly call parent::__destruct() in the destructor body. A child class may inherit the parent's destructor if it does not implement one itself.
 	
The destructor will be called even if script execution is stopped using exit(). Calling exit() in a destructor will prevent the remaining shutdown routines from executing.
	function __destruct()	function __destruct() {
		  …
		}

Properties
 	
Class member variables are called "properties". They are defined by using one of the keywords public, protected, or private, followed by a normal variable declaration. This declaration may include an initialization.
 	
In order to maintain backward compatibility (PHP 4, PHP 5) will still accept the use of the keyword var in property declarations instead of public, protected, or private. However, var is no longer required. If you declare a property using var then PHP 5 will treat the property as if it had been declared as public.
	public $propertyName;
	
	protected $propertyName;
	
	private $propertyName;
	
	var $propertyName;
		class Foo
		{
		   public $bar = ‚property';
		}

Methods
 	
With a parameter list you can pass information to a function. The parameter list is a comma-separated list of expressions. The parameters are evaluated from left to right.
	function {
	    ....
	}
	
	function($par1, par2)
	    ....
	}
		public function bar() {
		        return ‘method’;
		    }
		
		public function sum() {$sum1, $sum2)
		        $result = $sum1 + $sum2
		        return $result;
		    }

Visibility

private:	That members can be accessed by the class where they are defined.
protected:	That members can be accessed only within the class itself and by inherited and parent classes.
public:	That members can be accessed everywhere.

Property Visibility

Description	Example	Result

Class properties may be defined as public, private, or protected. If declared using var, the property will be defined as public.
 	
Note: We can redeclare the public and protected properties, but not private.
	class MyClass
	{
	    public $public = ‚Public‘;
	    protected $protected = ‚Protected‘;
	    private $private = ‚Private‘;
	    function printHello()
	    {
	        echo $this->public;
	        echo $this->protected;
	        echo $this->private;
	    }
	}
		$obj = new MyClass();
		echo $obj->public; // Works
		echo $obj->protected; // Fatal Error
		echo $obj->private; // Fatal Error
		$obj->printHello(); // Shows Public, Protected and Private

Method Visibility

Class methods may be defined as public, private, or protected. Methods declared without any explicit visibility keyword are defined as public.
 	
Note: We can redeclare the public and protected methods, but not private.
	class MyClass
	{
	    public function __construct() { }
	    public function MyPublic() { }
	    protected function MyProtected() { }
	    private function MyPrivate() { }
	    function Foo() // This is public
	    {
	        $this->MyPublic();
	        $this->MyProtected();
	        $this->MyPrivate();
	    }
	}
		$myclass = new MyClass;
		$myclass->MyPublic(); // Works
		$myclass->MyProtected(); // Fatal Error
		$myclass->MyPrivate(); // Fatal Error
		$myclass->Foo(); // Public, Protected and Private work

Constant Visibility

Class constants may be defined as public, private, or protected. Constants declared without any explicit visibility keyword are defined as public.

Note: We can redeclare the public and protected constants, but not private.

```
class MyClass

{

    public const MY_PUBLIC = ‚public‘;

    protected const MY_PROTECTED = ‚protected‘;

    private const MY_PRIVATE = ‚private‘;

    public function foo()

    {

        echo self::MY_PUBLIC;

        echo self::MY_PROTECTED;

        echo self::MY_PRIVATE;

    }

}
```

```
$myclass = new MyClass();

MyClass::MY_PUBLIC; // Works

MyClass::MY_PROTECTED; // Fatal Error

MyClass::MY_PRIVATE; // Fatal Error

$myclass->foo(); // Public, Protected and Private work
```

Object Inheritance

When you extend a class, the subclass inherits all the public and protected methods from the parent class.

Unless a class overrides those methods, they will retain their original functionality.

```
class Foo

{

    public function printItem()

    {

    }

    public function printPHP()

    {

    }

}

class Bar extends Foo

{

    public function printItem()

    {

    }

}
```

```
$foo = new Foo();

$bar = new Bar();


$foo->printItem();  // use function from class Foo


$foo->printPHP(); // use function from class Foo


$bar->printPHP(); // use function from class Foo


$bar->printItem(); // use function from class Bar
```

Dr. Veikko Krypczyk is an enthusiastic developer and specialist author.

Elena Bochkor works primarily on the design and design of mobile applications and websites.

Further information on the topics of the authors of the Cheat Sheets can be found at http://larinet.com.