Apéndice A, (Programación orientada a objetos en PHP):

En éste apéndice explicaré que compone la programación orientada a objetos y como trabajar con objetos en PHP5, tendremos la teoría y por supuesto sus ejemplos.

La idea de la programación orientada a objetos es realizar algoritmos más ordenados y claros para un programador, la POO ayuda considerablemente a la reutilización de código, es cuestión de conocer las herramientas y características de la POO para aprovechar mejor el código, sin más doy paso a este breve tutorial (por decirlo así).

Lo que debemos saber:

- Clases: Definición, uso.
- Objetos: Definición, uso.
- Métodos: Definición, uso.
- Constructores y Destructores.
- Funciones: Definición, uso.
- Diferencias entre métodos y funciones
- El operador $this.
- La herencia.
- El encapsulamiento.
- El polimorfismo.
- Excepciones.
- Un ejemplo con todo esto

Clases

Una clase es una defición de unas características y funcionalidades, algo abstracto que se concretiza con la instanciación de un objeto de esa clase.

Definiendo una clase:

```
<?php
class clsMensajesDefault {
	var $intTipoMensaje = 1; //valor por defecto del objeto al instanciar la clase
	var $strNombre = 'Angel'; //valor por defecto del objeto al instanciar la clase
	var $strMensaje;
	function fctImprimirMensaje () {
		echo $this->strMensaje;
	}
	function fctGenerarMensaje() {
		switch ($this->intTipoMensaje) {
			case 1: $this->strMensaje = 'Hola '.$this->strNombre.' como estas?';break;
			case 2: $this->strMensaje = 'Adios '.$this->strNombre;break;
			case 3: $this->strMensaje = 'Salut! '.$this->strNombre.' ça va?';break;
			case 4: $this->strMensaje = 'Au revoir '.$this->strNombre;break;
			case 5: $this->strMensaje = 'Ciao! '.$this->strNombre.' come va?';break;
			case 6: $this->strMensaje = 'Ciao! '.$this->strNombre.' ci vediamo!';break;
		}
	}

}
```

```
?>
```

Usando la clase 'clsMensajesDefault':

Ahora veremos como usar la clase que acabamos de crear dando un par de ejemplos, en el ejemplo1 veremos como se imprime el mensaje SIN pasarle parámetros a los objetos intTipoMensaje y strNombre. En el segundo ejemplo le pasamos parámetros a strNombre y a intTipoMensaje.

Ejemplo 1:

```
<?php
$objMsj = new clsMensajesDefault(); //instanciamos la clase en un objeto llamado objMsj
$objMsj->fctGenerarMensaje(); // llamamos al método generar mensaje
$objMsj->fctImprimirMensaje(); // llamamos al método imprimir mensaje
?>
```

Ejemplo 2:

```
<?php
// asignamos valores a intTipoMensaje y strNombre
$objMsj->intTipoMensaje = rand(1,6); // asignamos a intTipoMensaje un valor random del 1 al 6
$objMsj->strNombre = 'Rafael'; //declaramos un valor al atributo strNombre
$objMsj->fctGenerarMensaje(); // llamamos al método generar mensaje
$objMsj->fctImprimirMensaje(); // llamamos al método imprimir mensaje
?>
```

Objetos

Un objeto puede ser cualquier cosa, principalemente está relacionado con una clase. Un objeto tiene propiedades y valores concretos, y pueden hacérsele llamadas a métodos que sean parte de ese objeto, para que así ejecuten acciones.

Si queremos trabajar con clases debemos instanciar un objeto que obtiene las propiedades, métodos y funciones de esa clase.

Ejemplo 1:

Hagamos la clase clsCarro y su implementación a través de el objeto $objCarro.

```
<?php
class clsCarro() {
	var $intNumeroRuedas = 4; //parametro por defecto
	var $blnMotor = TRUE; // parametro por defecto
	var $blnEncendido = FALSE; //parametro por defecto
	// Metodos de la clase clsCarro
	function fctRodar() {
		if ($this->blnEncendido) {
			echo 'soy un carro y voy a '.rand(20,100).'km/h <br />';
		}
		else {
			echo 'Enciéndeme primero!';
		}
	}
	function fctEncender() {
		if ($this->blnEncendido) {
			echo 'Vas a quemar el arranque ya yo estoy encendido!';
```

```
		}
		else {
			echo 'Carro encendido!';
			$this->blnEncendido = TRUE;
		}
	}
	function fctApagar() {
		if (!$this->blnEncendido) {
			echo 'No puedo apagarme más!';
		}
		else {
			echo 'Ok jefe, lo que usted diga :D';
		}
	}
	// Funciones de la funcion clsCarro
	function fctTieneMotor() {
		return ($this->blnMotor);
	}
}

/*
Ahora vamos con el script
Tenemos una clase llamada clsCarro la cual contiene lo siguiente:
Métodos: fctRodar, fctEncender, fctApagar
Valores: $intNumeroRuedas, $blnMotor, $blnEncendido
Funciones: fctTieneMotor
Realizaremos un pequeño algoritmo en un poco de pseudo-código y PHP
*/
$objCarro = new clsCarro();
if ($objCarro->fctTieneMotor() && $objCarro->intNumeroRuedas == 4) {
	$objCarro->fctEncender();
	$objCarro->fctRodar();
}
else {
	$objCarro->fctApagar();
}
?>
```

Como observamos en el programa a través del objeto $objCarro manejamos todos sus métodos, sin embargo en los siguientes ejemplos podremos observar que flexible puede ser el trabajar con clases.

Métodos

Los métodos son bloques de código a las que se le hace una llamada y ejecutan un procedimiento, colocaremos un ejemplo de un par de métodos dentro de una clase llamada clsPenalty.

```
<?php
class clsPenalty() {
	var $intDireccion = 0; // Direcciones posibles 0: Centro, 1: Izquierda, 2:Derecha,3:Fuera
	var $blnGol = FALSE;
```

```
	function fctCobrar() {
		switch ($this->intDireccion):
			case 0: $strDir = 'El Centro';break;
			case 1:$strDir =  'La Izquierda';break;
			case 2:$strDir =  'La Derecha';break;
		endswitch;
		if (rand(0,1) == 1) {
			echo 'GOOOOL un buen disparo hecho por '.$strDir;
			$this->blnGol = TRUE;
		}
		else {
			echo 'Lamentablemente el cancerbero se lanzo por '.$strDir;
			$this->blnGol = FALSE;
		}
	}
	function fctUltimoPenalty() {
		$strGol = ($this->blnGol) ? 'y fue un golazo!' : 'y por mala suerte fallaste';
		echo 'El ultimo penalty lo cobraron en la siguiente direccion '.$this->intDireccion;
		echo '<br />'.$strGol;
	}
}
// Ahora cobremos un penalty virtual!
$objPenal = new clsPenalty();
$objPenal->intDireccion = rand(0,2); //
$objPenal->fctCobrar(); //Prueba tu suerte!
$objPenal->fctUltimoPenalty(); // ¡Mira la repetición!
?>
```

En el ejemplo anterior vimos los métodos fctCobrar, y fctUltimoPenalty, estos métodos no se les pasó ningún parámetro, sin embargo pueden recibirlos sin ningún problema.

Los métodos, como son 'funciones que no devuelven resultados' (por llamarlo asi en PHP), no necesariamente tienen que estar dentro de una clase, sin embargo ya que se está explicando POO, agregamos estos métodos dentro de una clase y los trabajamos como tal.

Constructor y Destructor

Se le dice 'método constructor' al método que al momento de instanciar una clase se ejecuta, es decir, cuando hacemos $objMiClase = new miClase(); en ese momento se ejecuta un método constructor.

Para definir el método constructor se debe usar en el nombre de la función '__construct()'. También podemos crear un constructor si lo llamamos como la clase, es decir si tenemos la clase clsMiClase y una función dentro de esa clase llamada igual que la clase, el método constructor será ese, a continuación el ejemplo:

```
<?php
	class clsMiClase {
		....
		function clsMiClase() {
			echo 'Hola, soy el constructor';
		}
	}
?>
```

Se le dice 'método destructor' al método que se ejecuta en el momento donde se cierra la instancia de la clase; es decir, cuando hacemos unset($objMiClase); en ese momento llamamos al método destructor y luego se termina la instancia.

Para definir el método destructor se debe usar el nombre de la función '__destruct()'.

Ahora veamos un ejemplo básico de una clase con un constructor y un destructor, cabe destacar que por recomendación, prefiero usar __construct() en vez de hacer una funcion interna con el nombre de la clase.

```
<?php
class clsPuente {
	function __construct() {
		echo 'Acabas de comenzar la obra! <br />';
		//Aqui llamamos a otro método perteneciente de la clase
		$this->fctPicoYpala;
	}
	function __destruct() {
		echo 'Fin de la obra!, lástima que destruyas el puente :( <br />';
	}
	function fctPicoYpala {
		echo 'Hora de trabajar! <br />';
	}
}
// Ahora veamos el código...
$objMiPuente = new clsPuente(); // Aquí llamamos al método constructor
unset($objMiPuente); //Aquí lo destruimos
?>
```

Excepciones

Las excepciones proveen una solución para el manejo de errores en los lenguajes de programación. Las excepciones son la mejor alternativa para manejar el uso especial de los errores devueltos por el fallo a la llamada de una función. Tenemos varios métodos para manejar excepciones, así que pondremos unos varios ejemplos. En la mayoría de los casos, el omitir el uso de las excepciones se considera una mala práctica. Para trabajar las excepciones usaremos try (código que va a correr que puede tener una excepción) y catch para capturar la excepción en caso de que suceda, veamos un ejemplo:

```
<?php
class clsEscribirDatos {
	var $handle;
	var $strTexto = "Estamos escribiendo desde PHP, cambio.\n";
	function fctAbrirArchivo() {
		$this->handle = @fopen('archivo.txt','a+'); // con el @ evitamos el error msj de PHP
		if (!$this->handle) { // si falla al abrir el archivo
			$strError = 'No pude abrir el archivo!'; // definimos un mensaje de error
			throw new Exception($strError); // creamos la excepcion
		}
	}
```

```
	function fctEscribirDatos() {
		try { //como sabemos que este bloque _puede_ presentar errores creamos el try
			$this->fctAbrirArchivo();
			$blnEscrito = @fwrite($this->handle,$this->strTexto);
			if (!blnEscrito) {
				throw new Exception('Error escribiendo el archivo');
				//Creamos otra excepcion por si no podemos escribir
			}
		}
		catch(Exception $strEx){ // capturamos si hubo una excepcion
			echo 'Error: Houston we got a problem.. '.$strEx->getMessage();
			//imprimos el mensaje y chau
		}
	}
	function __construct($strTexto2) { //metodo constructor
		$this->strTexto = (!is_null($strTexto2)) ? $strTexto2 : $this->strTexto;
		$this->fctEscribirDatos();
	}

}
$objDatos = new clsEscribirDatos(NULL);
$objDatos->__construct("Estamos escribiendo llamando directamente al constructor :D\n");
?>
```

Funciones

Las funciones son bloques de códigos a los que se le pasa parámetros y devuelven una salida, una función es la explicación práctica de la teoría de entrada -> proceso -> salida.

Haremos un par de funciones más para la explicación de las mismas vamos a usar la misma clase clsPenalty.

```
<?php
class clsPenalty() {
	var $intDireccion = 0; // Direcciones posibles 0: Centro, 1: Izquierda, 2:Derecha,3:Fuera
	var $blnGol = FALSE;
	function fctCobrar() {
		switch ($this->intDireccion):
			case 0: $strDir = 'El Centro';break;
			case 1:$strDir =  'La Izquierda';break;
			case 2:$strDir =  'La Derecha';break;
		endswitch;
		if (rand(0,1) == 1) {
			echo 'GOOOOL un buen disparo hecho por '.$strDir;
			$this->blnGol = TRUE;
		}
		else {
			echo 'Lamentablemente el cancerbero se lanzo por '.$strDir;
			$this->blnGol = FALSE;
		}
		return $this->blnGol;
	}
	function fctUltimoPenalty() {
```

```
                $strGol = ($this->blnGol) ? 'y fue un golazo!' : 'y por mala suerte fallaste';
                echo 'El ultimo penalty lo cobraron en la siguiente direccion '.$this->intDireccion;
                echo '<br />'.$strGol;
        }
}
// Ahora cobremos un penalty virtual!
$objPenal = new clsPenalty();
$objPenal->intDireccion = rand(0,2); //
$blnGol =  $objPenal->fctCobrar(); //Prueba tu suerte!
if ($blnGol) {
        // Si hace Gol le mostramos la repeticion, sino no seremos tan crueles como con Baggio
        $objPenal->fctUltimoPenalty(); // ¡Mira la repetición!
}
?>
```

Como vemos la variable local fuera de la clase blnGol (ojo existe un valor ya dentro del objeto $objPenal->blnGol), tiene un valor true or false, si es true, mostramos la repetición, sino no se muestra.

El operador '$this'

El operador $this sirve para llamar métodos y variables que están disponibles en nuestra clase, pero que no son locales. Ejemplo:

```
<?php
class clsPrueba {
        var $strVar1;
        var $intVar2;
        function fctImprimir() {
                echo $this->strVar1; // esto devuelve el valor de strVar1
                echo $strVar1; //esto imprimirá nada :)
                echo '<br />';
        }
        function fctImprimir2() {
                echo 'Hola voy a imprimir un valor entero'.$this->intVar2.' <br />';
                echo 'Ahora vamos a llamar al metodo fctImprimir';
                $this->fctImprimir();
        }
}
// Bien tenemos nuestra clase lista ahora daremos paso a su uso
$objClasePrueba = new clsPrueba(); // instanciamos la clase
$objClasePrueba->strVar1 = 'Angel';
$objClasePrueba->intVar2 = 1337;
$objClasePrueba->fctImprimir(); //llamamos al método fctImprimir
$objClasePrueba->fctImprimir2(); //llamamos al método fctImprimir2
?>
```

Diferencias entre funciones y métodos

En PHP5 las funciones y los métodos se inicializan con la instrucción function, para PHP al parecer no existen los métodos en 'teoría' sin embargo nos permite que las funciones trabajen como métodos. Ahora veremos un pequeño ejemplo donde tenemos un método y una función, para entender su diferencia (si es que hasta ésta altura del tutorial no ha sido tan clara la diferencia).

```
<?php
```

```
	class clsVariada {
		function fctEstoEsUnMetodo() {
			echo 'Saludos desde un metodo';
		}
		function fctEstoEsUnaFuncion() {
			return 'Saludos desde una funcion';
		}
	}
	clsVariada::fctEstoEsUnMetodo();
	echo clsVariada::fctEstoEsUnaFuncion();
?>
```

El encapsulamiento

El encapsulamiento no es más que darle seguridad a nuestros objetos, protegerlos, por decirlo así de que sean llamados por otros objetos que no controlan esa clase. Cuando se habló de las clases, se habló de que podían ser públicas y privadas, el encapsulamiento es eso, el nombre teórico que se le da a la práctica de proteger métodos y objetos.

Veamos un ejemplo:

```
<?php
	class clsCapsula {
		private $strMiVariable = 'Hola, no puedes modificarme';
		function __construct() {
			echo $this->strMiVariable;
		}
	}
	$objCapsula = new clsCapsula;
	$objCapsula->strMiVariable = 'Si puedo modificarte!'; //Aqui sale el fatal error
	$objCapsula->__construct();

?>
```

Quizás en una variable no se ve tanto el ejemplo de encapsulamiento, a continuación mostraré un ejemplo más amplio.

```
<?php
	class clsCapsulaDos {
		private function fctQueFechaEs($intTiempo) {
			echo 'Hoy es '.date('d/m/Y',$intTiempo);
		}
		public function fctQueDiaEsHoy() {
			this::fctQueFechaEs(time());
		}
	}
	$objCapsula2 = new clsCapsulaDos();
	$objCapsula2->fctQueDiaEsHoy();
	// Si tratas de hacer esto fallaras
	$objCapsula2->fctQueFechaEs(1165179068); // no pasarás!
?>
```

La herencia

Como hemos mencionado anteriormente, la programación orientada a objetos es flexible, y una de las características que la hace flexible es la herencia.

Al defininir una clase 'extendida de otra' podemos acceder a sus métodos publicos, (la parte de los tipos de métodos y funciones se puede ver en el capítulo de encapsulamiento).

A continuación, un ejemplo básico de un arbol de clases pequeño.

```
<?php
	class clsPadres {
		function fctPadre() {
			echo 'Jose';
		}
		function fctMadre() {
			echo 'Maria';
		}
	}
	class clsHijos extends clsPadres {
		function fctQuienSoy() {
			echo 'Soy Jesus';
		}
		function fctMisPadres() {
			echo parent::fctPadre().' y '.parent::fctMadre();
		}
	}
	$objYo = new clsHijos();
	echo “Hola, ¿quien eres?<br />”;
	echo “Yo soy “.$objYo->fctQuienSoy.”<br />”;
	echo “¿Y quienes son tus padres?<br />”;
	echo “Mis padres son “.$objYo->fctMisPadres();
?>
```

Escrito por: Angel 'angvp' Velásquez <angvp@archlinux.com.ve>