Capitulo 7: Programación Orientada a Objetos en PHP

Para comenzar a hablar de programación orientada a objetos (OOP – Object Oriented Programming) es necesario recordar los conceptos de la programación orientada a objetos. Estos conceptos varían entre los distintos autores, pero podemos mencionar algunos que son básicos y necesarios para cualquier lenguaje del cual pueda decirse que es orientado a objetos:

- Tipos de datos abstractos e información encapsulada
- Herencia
- Polimorfismo

La encapsulación en PHP se codifica utilizando clases:

```
<?php

class Algo {
// En OOP se utilizan generalmente nombres comenzados con mayúscula.
var $x;

function setX($v) {
// Para los métodos se utilizan generalmente nombres en minúscula y sólo
// se utiliza mayúscula para separar las palabras, por ej. getValueOfArea()
$this->x=$v;
}

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

?>
```

Obviamente está nomenclatura es sólo a valor de recomendación para mantener un standard entre el código de los distintos programadores, y puede no ser respetado. Lo importante es acordar una nomenclatura standard que todos respeten.

Las propiedades de los objetos son definidas en PHP utilizando la declaración “var” dentro de la clase. Cuando se declara una propiedad la misma no tiene tipo alguno asignado, hasta que nosotros la asignemos algún valor en particular. Una propiedad puede ser un entero, un vector, un vector asociativo, e inclusive puede ser otro objeto.

Los métodos son definidos como funciones, también dentro de la clase,. Y para acceder a las propiedades de la instancia de esa clase es necesario referirse a las propiedades como $this->name. En caso de no utilizar el “$this->” la variable será local al método y una vez terminada la ejecución del mismo se perderá su valor.

Para crear una instancia de un objeto debemos ejecutar el operador “new”, que nos devuelve en una variable un objeto de la clase que le estamos indicando.

$obj = new Something;

Una vez instanciado el objeto podemos utilizar sus métodos:

$obj->setX(5);
$see=$obj->getX();

El método setX ejecutado en el objeto $obj hizo que se asigne un 5 a la propiedad “x” de dicha instancia. Notemos en este punto que podríamos haber seteado la propiedad “x” con cualquier tipo de variables, por ejemplo un string.

Para asignarle 5 a la propiedad “x” de nuestro objeto $obj podríamos haber puesto en nuestro código directamente “$obj->x=5;”, sin la necesidad de llamar a ningún método, pero el problema radicaría en que en que estaríamos violando la regla de encapsulamiento de los objetos. Una buena práctica de la programación orientada a objetos es acceder a las propiedades solamente mediante métodos propios de la clase y jamás acceder a ellos de otra forma. Lamentablemente PHP no ofrece la posibilidad de declarar las propiedades privadas, por lo que el programar en forma “encapsulada” se torna más una filosofia de programación que una obligación.

La herencia en PHP se realiza utilizando la sentencia “extends”:

```
<?php

class Another extends Something {
var $y;
function setY($v) {
// Para los métodos se utilizan generalmente nombres en minúscula y sólo
// se utiliza mayúscula para separar las palabras, por ej. getValueOfArea()
$this->y=$v;
}

function getY() {
return $this->y;
}
}

?>
```

Los objetos de la clase “Another” poseen todas las propiedades y métodos de su clase padre “Something”, más las propiedades y métodos propios. Ahora podemos ejecutar por ejemplo los siguientes comandos:

$obj2=new Another;
$obj2->setX(6);
$obj2->setY(7);

En PHP una única clase de objetos no puede ser “hija” de más de un “padre”, lo que es conocido como múltiple herencia.

En PHP se pueden reescribir métodos de la clase padre en la clase hijo (overriding). Para esto sólo hace falta volver a definir la función en el objeto hijo. Por ejemplo si queremos redefinir el método getX para la clase “Another” simplemente definimos la función en la clase “Another”. Una vez hecho esto no es posible para los objetos de la clase “Another” acceder al método getX de “Something”.

En caso de declararse una propiedad en la clase “hija” con el mismo nombre que en la clase padre, la propiedad de la clase padre se encontraría “oculta”.

En las clases se pueden definir constructores. Los constructores son métodos que se ejecutan al momento de instanciar la clase (cuando se ejecuta la sentencia new). La característica para que un método sea el constructor de una clase es que debe tener el mismo nombre que la clase.

```
<?php

class Something {
```

```
var $x;

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

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

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

?>
```

Entonces, se puede crear el objeto de la siguiente manera:

$obj=new Something(6);

Automáticamente el constructor asigna 6 a la propiedad “x”.

Todos los métodos, incluyendo los constructores, son funciones normales de php, por lo que se le pueden asignar valores por omisión.
Supongamos que tenemos el constructor definido de la siguiente manera:

function Something($x="3",$y="5")

En este caso podemos crear el objeto de la siguiente manera:

$obj = new Something(); // x=3 y y=5
$obj = new Something(8); // x=8 y y=5
$obj = new Something(8,9); // x=8 y y=9

Los argumentos por omisión son utilizados en C++ y son una vía para cuando no hay valores para pasar por parámetro a las funciones. Cuando el parámetro no es encontrado por la función que es llamada, toma el valor por omisión que le es especificada en su definición.

Cuando es creado un objeto de una clase que deriva de otra, se ejecuta sólo el constructor de la misma clase, y no la del padre. Este es un defecto del PHP, ya que es clásico en los lenguajes orientados a objetos que exista lo que se llama encadenamiento de constructores. Para hacer esto en PHP es necesario llamar explícitamente el constructor de la clase padre dentro del constructor de la clase heredera. Esto funciona porque todos los métodos de la clase padre se encuentran disponibles en la clase heredera.

```
<?php

function Another() {
$this->y=5;
$this->Something(); //Llamada explícita al constructor de la clase padre.
}

?>
```

Un buen mecanismo en OOP es usar clases abstractas. Clases abstractas son aquellas clases que no son instanciables y están hechas para el único propósito de definir una interface para otras clases derivadas. Los diseñadores

usualmente utilizan estas clases para forzar a los programadores a derivar clases desde ciertas bases clases de forma tal de asegurarse que cada clase tiene una cierta funcionalidad predefinida. No hay una forma standard de hacer esto en PHP pero: Si se necesita esta metodologia puede definirse una clase base y poner una sentencia "die" en el constructor, de esta forma nos aseguramos que la clase no es instanciable luego definimos los metodos interface poniendo una sentencia "die" en cada uno de ellos, de esta forma los programadores deben sobreescribir estos metodos para poder utilizarlos.

No hay destructores en PHP.

La sobrecarga (overloading) de metodos que es diferente a la redefinicion (overriding) no esta soportada en PHP. En OOP se dice que un metodo esta sobrecargado cuando hay mas de un metodo con el miusmo nombre pero diferentes tipos o cantidad de parametros. PHP es un lenguaje debilmente tipado por lo que la sobrecarga por tipos no funcionaria, por consistencia la sobrecarga por cantidad de parametros tampoco funciona.

En OOP es agradable muchas veces sobrecargar constructores de forma tal que una clase permita construir un objeto de muchas formas diferentes, podemos simular esto en PHP de la forma:

```
<?php

class Myclass {
function Myclass() {
$name= "Myclass".func_num_args();
$this->$name();
//Notar que $this->$name() es susualmente erroneo porque aquí
//$name es un string con el nombre del método a llamar.
}

function Myclass1($x) {
code;
}
function Myclass2($x,$y) {
code;
}
}

?>
```

Con éste trabajo extra podemos trabajar con la clase de un modo transparente para el usuario:

$obj1=new Myclass('1'); //Will call Myclass1
$obj2=new Myclass('1','2'); //Will call Myclass2

Polimorfismo

Polimorfismo se define como la habilidad de un objeto de determinar que método debe invocar para un objeto pasado como argumento en tiempo de ejecución. Por ejemplo si tenemos una clase figura que tiene un método "dibujar" y sus clases derivadas circulo y rectángulo, cuando reemplazamos el método "dibujar" podemos tener una función que cuente con un argumento "x" y después llamar a $x->dibujar(). Si tenemos polimorfismo el método "dibujar" llamado dependerá del tipo de objeto que pasemos a la función. El polimorfismo es muy fácil de aplicar y utilizar en lenguajes interpretados como PHP, pero no es tan secillo en lenguajes compilados, ya que no sabemos que método deberemos llamar de antemano.

```
<?php
```

```php
function niceDrawing($x) {
//Supongamos que este es un método de la clase Board.
$x->draw();
}

$obj=new Circle(3,187);
$obj2=new Rectangle(4,5);

$board->niceDrawing($obj); //Podemos llamar al método draw de circulo.
$board->niceDrawing($obj2); //Podemos llamar al método draw de rectángulo.

?>
```

Programación Orientada a Objetos en PHP

Algunos puristas pueden decir que PHP no es un verdadero lenguaje orientado a objetos. PHP es un lenguaje hibrido donde se puede utilizar OOP o procedimientos de programación procedural tradicional. Para largos proyectos usted pude querer o necesitar usar OOP "puro" en PHP declarando clases y utilizando sólo objetos y clases para su proyecto. Mientras más largo es el proyecto más puede ayudar la programación orientada a objetos, ya que código es más fácil de mantener, entender y reutilizar. Estos son los principios básicos de la Ingeniería de Software- Aplicar estos conceptos a proyectos basados en la web será la llave de acceso para exitosos sites en el futuro.

Técnicas avanzadas de programación orientada a objetos.

Luego de haber visto los conceptos básicos de la OOP, vamos a ver algunas técnicas más avanzadas:

Serialización

PHP no soporta la persistencia de objetos. En OOP los objetos persistentes son objetos que mantienen su estado y funcionalidad a través de múltiples invocaciones de la aplicación. Esto puede resolverse salvando el objeto en un archivo o en una base de datos y restaurando los datos cada vez que se ejecuta dicha aplicación. El mecanismo es conocido como serialización. PHP provee un método de serialización que puede ser llamado por los objetos . El método de serialización devuelve un string representando el objeto. La serialización guarda las propiedades del objeto pero no sus métodos.

En PHP4 si se serializa un objeto al string $s, se destruye el objeto, y entonces utilizando la desserialización de objeto en $obj se puede mantener el acceso a las propiedades del objeto. Esto no es recomendado por dos razones: la primera es porque no se garantiza que en futuras versiones esto siga funcionando. La segunda es porque si se serializa un objeto, se guarda a disco el string y se sale del script, al correr en el futuro dicho script no nos aseguramos que los métodos del objeto sean los mismos, ya que en la serialización sólo se guardaron las propiedades. Concluyendo, serialización sólo sirve en PHP para guardar las propiedades de un objeto nada más (se puede serializar un vector asociativo para salvarlos a disco por ejemplo).

NOTA: La version 4.0.1 de PHP4 soporta serializacion de objetos en forma completa!!!

Ejemplo: <?php

```php
$obj=new Classfoo();
$str=serialize($obj);
// Se salva $str al disco
```

```
//...algunos meses más tarde

//Se recupera str del disco
$obj2=unserialize($str)

?>
```

En este caso tenemos las propiedades recuperadas pero no podemos utilizar los métodos. Esto hace que la forma de recuperar $obj2->x el valor de “x” sea la única forma posible (no tenemos métodos!), por lo que esta práctica no es recomendada.

Estas son algunas maneras para arreglar el problema, las que veremos muy por encima, ya que son formas muy enmarañadas de resolverlo. Full serialización será uno de los agregados más esperados en PHP.

Nota: Enhorabuena, PHP 4.0.1 soporta “full serialization” de objetos.

Usando clases para manipular información almacenada

Una de las cosas útiles de PHP y OOP que se pueden definir fácilmente clases para manejar ciertas cosas y llamar a la clase apropiada cuando queramos. Supongamos que tenemos un formulario html donde un usuario selecciona un producto por medio de su producto ID. Supongamos también que tenemos los datos del producto en una base de datos y queremos mostrar el producto en sí, el precio y demás características. Si tenemos productos de diferentes tipos, y en la misma acción tenemos diferentes maneras para diferentes familias de productos. Por ejemplo para algunos porductos debemos reproducir un sonido mientras que para otros productos debemos mostrar una foto del mismo guardada en la base de datos. En este caso podemos utilizar OOP y PHP para realizar menos y mejor código fuente:

Podemos definir una clase Producto, definimos que métodos la clase debe tener (por ejemplo “display”), y entonces definimos clases para cada tipo de producto como una extensión (extends) la clase producto (por ejemplo clase ItemSonoro,. ItemVisible, etc.) redefiniendo los métodos que ya definimos en Producto en cada una de las clases que necesitemos. Lo que resulta conveniente en este caso es nombrara las clases de acuerdo con los tipos de “columna” que guardamos en la base de datos por cada tipo de producto que tenemos (id, tipo, precio, etc.). Entonces cuando procesamos el script podemos pedir el tipo desde la base de datos e instanciar un objeto de la clase del tipo de producto:

```
<?php

$obj=new $type();  // type has the name of the class to instanciate!
$obj->action();

?>
```

Esto es una buena propiedad de PHP que permite llamar al método “display” por ejemplo del tipo de objeto que tenemos. Con esta técnica no necesitamos tocar el script de proceso (el formulario) cuando agregamos un nuevo tipo de objetos, solamente agregamos la clase y listo. Esto es poderoso, lo único qu debemos definir son los métodos que todos los objetso deben tener de acuerdo a los tipos que tenemos, generando las diferentes maneras para los diferentes tipos.

Si ud. Lidera un grupo de programandores es fácil dividir tareas, cada uno responsable por cada tipo de objetos y las clases que dependen de él. Con estos métodos la internacionalización de un site puede ser muy fácil aplicando los métodos correspondientes de acuerdo al lenguaje que seleccionó el usuario.

Copiando y Clonando

Cuando creamos un objeto $obj se puede copiar un objeto usando $obj2=$obj. El nuevo objeto es una copia del objeto $obj, es decir que tiene el estado que $obj tenia en el momento que se realizó la asignación. Algunas veces no necesitamos esto, sólo queremos crear un nuevo objeto de la misma clase que $obj, llamando al contructor en el momento de la creación del nuevo objeto. Esto es posible utilizando la serialización y una clase que todas las otras clases sean extensión de la misma.

Entrando a una zona difícil

Cuando serializamos un objeto obtenemos un string de un formato propio. Podemos investigas (siendo curiosos) cada una de las cosas que tiene dicho string. Una cosa curiosa es que esta guardado en el el nombre de la clase que debemos utilizar para desserializar el objeto:

```
<?php

$herring=serialize($obj);
$vec=explode( ':',$herring);
$nam=str_replace( "\"", '',$vec[2]);

?>
```

Entonces supongamos que creamos una clase “Universe” y forzamos que todas las clases sean extensión de universo, podemos definir un método que clone Universe como:

```
<?php

class Universe {
function clone() {
$herring=serialize($this);
$vec=explode( ':',$herring);
$nam=str_replace( "\"", '',$vec[2]);
$ret=new $nam;
return $ret;
}
}

//Entonces:

$obj=new Something();
//Algo extensión de universo !!
$other=$obj->clone();

?>
```

Se obtiene un objeto nuevo de la clase “Something” creado de la misma forma que llamando a New, se invoca al constructor etc. Distintas aplicaciones del uso de la clase universal permiten varios manejos curiosos de objetos en PHP, el unico limite es la imaginacion!