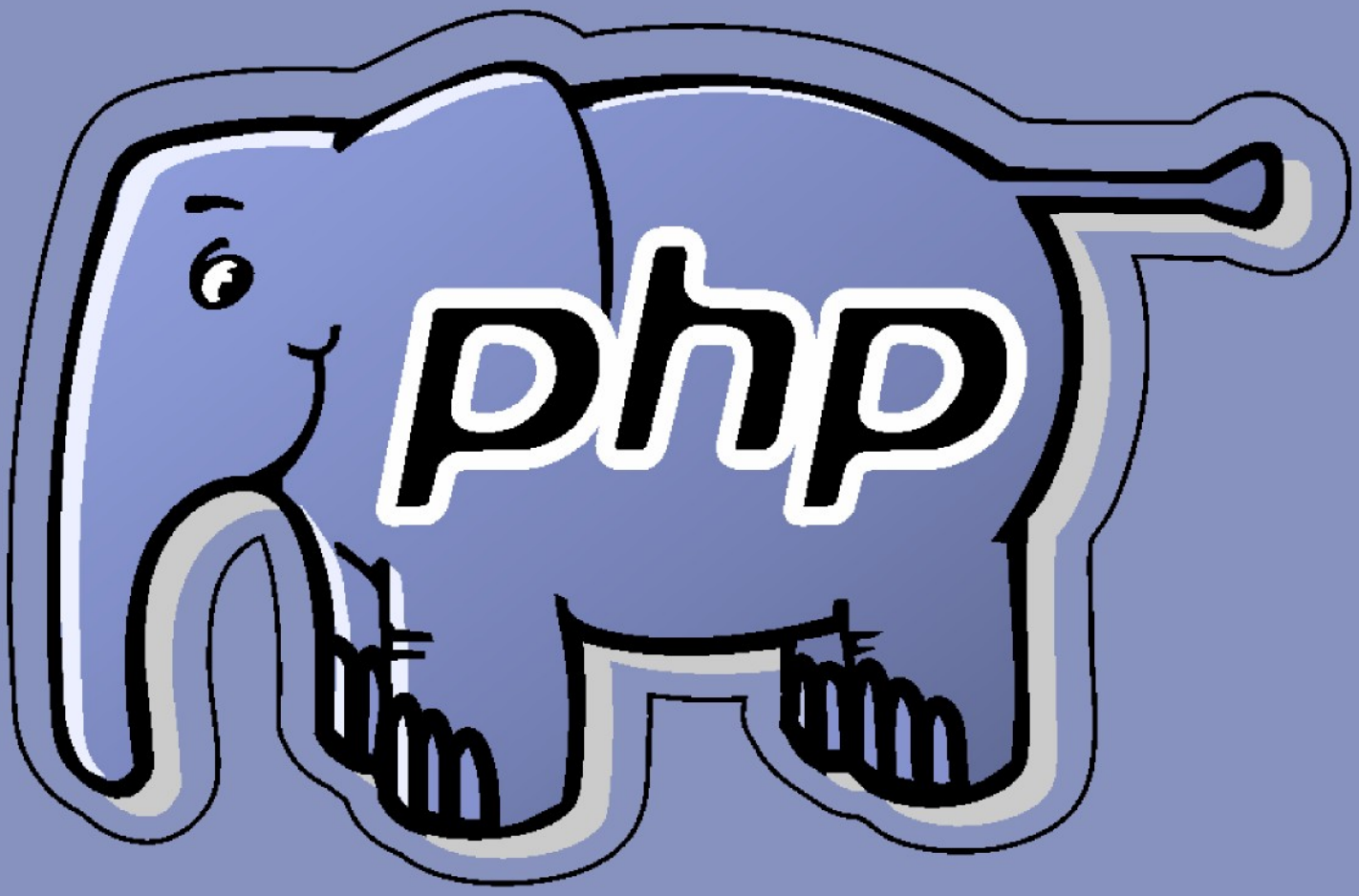
Programación en



Versión 2.0

php

Gugler
Laboratorio de Investigación Gugler

FCyT
Facultad de Ciencia y Tecnología

UADER
Universidad Autónoma de Entre Ríos

CONTENIDO

Capítulo 3

POO en PHP

Introducción

Como introducción al capítulo repasaremos algunos conceptos y fundamentos para poder abarcar la Programación Orientada a Objetos en PHP.

El concepto de la POO gira en torno a la idea de agrupar código y datos en unidades lógicas llamadas *clases*. Este proceso se conoce generalmente como encapsulación puesto que su objetivo es el de dividir una aplicación en entidades separadas cuyos componentes internos pueden cambiar sin alterar sus interfaces externas.

Las *clases* son en esencia la representación de un conjunto objetos por medio de funciones (*métodos*) y variables (*propiedades*) diseñados para trabajar juntos y para proporcionar la interfaz al mundo exterior. Es importante comprender que *las clases* son planos que *no se pueden utilizar directamente*, sino que deben ser *instanciadas en objetos*, que pueden interactuar con el resto de la aplicación. Podemos pensar en clases como los planos para la construcción de un auto, mientras que los objetos son los propios autos cuando están fuera de la línea de producción. Al igual que un único plano se puede utilizar para construir varios autos, una clase puede ser utilizada para instanciar varios objetos.

Conceptos fundamentales de POO

¿ Qué es POO?

La programación orientada a objetos o POO (OOP según sus siglas en inglés) es un paradigma de programación que usa objetos y sus interacciones, para diseñar aplicaciones y programas de ordenador. En un programa OO, los objetos se envían mensajes entre si y esperan a cambio determinados comportamientos o mensajes.

Un ejemplo de orientación a objetos es una PC. Cada PC está formada por componentes fabricados por distintas empresas que no se encuentran relacionadas entre si. Cada componente está fabricado bajo una especificación que incluye información y comportamientos estandarizados. De esta forma, una lectora de CD deberá devolver datos cada vez que un sistema operativo lo requiera. Gracias a ésto, el fabricante de la PC no se deberá preocupar por el funcionamiento interno de la unidad de CD, sino de que esta responda las solicitudes adecuadamente.

Cuando programamos OO, realizamos llamadas a objetos sin conocer todas las características de los mismos. El programa solamente espera obtener información o cierto comportamiento por parte de los objetos sin necesidad de conocerlos internamente.

Este conjunto de características que hacen a la POO se logra en base a una serie de técnicas que nos permiten la adaptabilidad de los mismos a lo largo del desarrollo. Entre estas técnicas podemos nombrar: *herencia, abstracción, polimorfismo y encapsulamiento*. Todos ésto temas los iremos tratando a lo largo de este capítulo.

¿ Qué es un Objeto ?

Es cualquier entidad que posee un conjunto de propiedades o atributos (datos) y de comportamiento o funcionalidades (métodos) las cuales pueden interactuar entre si y reaccionar ante distintos eventos.

Los programas OO están compuestos por varios objetos que se comunican entre si mediante el envío de mensajes uno a otro. Un programa OO verdadero constará de un grupo lógico de objetos los que se comunican ente sí para dar la funcionalidad total de dicho programa.

Ejemplo: en un sistema informático un Cliente, un Video o una Factura son ejemplos de objetos. Un Cliente tendrá un nombre, una dirección y un teléfono. Además podrá alquilar un video, devolverlo o pagarlo.

¿ Qué es una clase ?

Una clase es un modelo de abstracción confeccionado a partir de objetos del mundo real. Una clase define las propiedades y el comportamiento de un conjunto de objetos. Una clase constituye una categoría de objetos y sirve para la creación de este tipo de objetos.

Ejemplo: si definiésemos la clase Cliente, diríamos que el conjunto de Clientes que podemos definir con ella posee nombre, dirección y teléfono (atributos), pero además puede alquilar un video, devolverlo o pagarlo (métodos).

¿ Qué es atributo ?

Es el conjunto de valores o datos que definirán en un momento en particular un objeto de otro. Es el conjunto de características que diferencian a cada uno de los objeto de nuestra aplicación.

Ejemplo: siguiendo con el ejemplo anterior vemos que dos clientes que se encuentren en el mismo momento dentro del video club podrían ser representados mediante dos objetos de la clase Cliente. Como ambos están por devolver un video que han alquilado, gracias a los atributos podremos diferenciar a cada uno de ellos sin inconvenientes.

¿ Qué es método ?

Es el conjunto de operaciones que un objeto puede realizar y mediante la cual se comunicará con otros objetos. Para cada uno de los objetos debemos preguntarnos ¿qué es lo que hace este objeto? y de esta manera podremos conocer de que forma interacciona con los demás objetos dentro del sistema.

Ejemplo: el objeto Cliente podrá alquilar, devolver, pagar, etc. Cada uno de estos serán métodos de nuestro objeto que permitirán relacionar al objeto Cliente con, por ejemplo, el objeto Video.

¿ Qué es un mensaje ?

Es la acción de efectuar una llamada a un método. Las acciones en POO se producen como respuesta a las peticiones de acciones, que se llaman mensajes. Un objeto puede aceptar un mensaje y como respuesta realizará una acción y devolverá un valor.

Ejemplo: para el caso del videoclub, el Cliente enviará un mensaje de Alquilar Video al dueño

para que este le Alquile el Video.

¿ Qué es herencia ?

Es la posibilidad que nos brindan los objetos de poder reutilizar los atributos y métodos definidos en otro objeto.

Ejemplo: siguiendo el ejemplo, si tomásemos el objeto Cliente y el objeto Dueño, ambos son dos personas que tienen roles (métodos) diferentes pero que además tienen cosas en común (nombre, dirección, teléfono, etc). Podríamos entonces definir una clase Persona que tuviese atributos nombre, dirección y teléfono y luego declarar que cada una de las clases Cliente y Dueño *hereden* estos atributos.

Características de POO

Abstracción

El término abstracción se refiere al hecho de aislar un elemento de su contexto o del resto de los elementos que lo acompañan y poner énfasis en el "¿qué hace?" más que en el "¿cómo lo hace?".

La abstracción desde el punto de vista de la POO expresa las características de un objeto que lo distinguen de los demás. Además de distinguir entre los objetos provee límites conceptuales. Se puede decir que la abstracción separa las características esenciales de las no esenciales dentro de un objeto. Si un objeto tiene más características de las necesarias los mismos resultarán difíciles de usar, modificar, construir y comprender.
Ejemplo: siguiendo con el caso del video club, un cliente tranquilamente posee un método *comer*. Si bien es sabido que es un método existente, sería innecesario para el desarrollo del sistema el hecho de contemplar a dicho método.

Encapsulamiento

La encapsulación significa agrupar a todos los componentes de un objeto en uno solo, por medio de algo, para ocultar de la simple vista los componentes internos del objeto.

Ejemplo: la televisión esta formada internamente por circuitos, chips, cinescopio y demás, la encapsulación de la televisión es el cajón donde están metidos todos estos componentes. Este

cajón los oculta de la simple vista, esto es la encapsulación.

Cada objeto es una estructura compleja en cuyo interior hay datos y programas, todos ellos relacionados entre sí, como si estuvieran encerrados conjuntamente en una cápsula. Esta propiedad (encapsulamiento), es una de las características fundamentales en la OOP.

Nota: la abstracción y la encapsulación no son lo mismo, pero están relacionadas porque sin encapsulación no hay abstracción, ya que si no se encapsulan los componentes no podríamos dar una abstracción alta del objetos al cual nos estamos refiriendo .

Ocultación

El principio de ocultación fomenta el hecho de que los objetos son inaccesibles e impiden que otros objetos, los usuarios, o incluso los programadores conozcan cómo está distribuida la información o qué información hay disponible.
Si el objeto ha sido bien construido, sus métodos seguirán funcionando en el nuevo entorno sin problemas. Esta cualidad hace que la OOP sea muy apta para la reutilización de programas.

Ejemplo: en el caso de la clase Cliente, si sus atributos y métodos estuviesen bien definidos, tranquilamente podríamos adaptarlo a otra aplicación para ser reutilizado ya que, por ejemplo, ese cliente del video club podría representar a otro cliente de un local de ropa.

Polimorfismo

Denota la posibilidad de construir varios métodos con el mismo nombre, pero con relación a la clase a la que pertenece cada uno, con comportamientos diferentes. Esto conlleva la habilidad de enviar un mismo mensaje a objetos de clases diferentes. Estos objetos recibirían el mismo mensaje global pero responderían a él de formas diferentes.
Ejemplo: un mensaje "+" a un objeto ENTERO significaría suma, mientras que para un objeto STRING significaría concatenación.

POO en PHP

Declarando una Clase

La declaración de una clase en PHP es muy sencilla, solamente bastará:

```
<?php
```

```
<?php
        class MiClase
        {
                // declaración de métodos y propiedades
        }
?>
```

En base a esta declaración el intérprete entenderá que se está declarando una clase de nombre *MiClase* la cual estará compuesta por métodos y propiedades.

Ejemplo:

```
<?php
        class Cliente
        {
                // declaración de métodos y propiedades
        }
?>
```

Instanciando un Objeto

Una vez definida nuestra clase, deberemos crear una instancia de ella para poder hacer uso de sus métodos y atributos. A dicha instancia se la llama *objeto* y para crearla haremos uso de la construcción *new*.

```
<?php
        class MiClase
        {
                // declaración de métodos y propiedades
        }

        $objeto= new MiClase();
?>
```

Ejemplo:

```
<?php
        class Cliente
        {
                // declaración de métodos y propiedades
        }

        $cliente= new Cliente();
?>
```

Modificadores de acceso

Los modificadores de acceso son palabras reservadas que especifican el nivel de accesibilidad que tendrán los métodos y propiedades de un objeto. PHP5 incluye cuatro modificadores de acceso:

- **public:** el recurso se puede acceder desde cualquier ámbito.
- **protected:** el recurso puede ser accedido desde la clase en que fue definido y desde sus descendientes.
- **private:** el recurso solo puede ser accedido desde la clase en que fue definido.
- **final:** el recurso es accesible desde cualquier ámbito pero no puede ser reescrito por clases descendientes.

Normalmente se declararán como públicos aquellos métodos y propiedades que deban ser accesibles desde fuera del objeto, mientras que deseará mantener como protegidos o privados aquellos que que sean internos a la llamada de los mimos.

Propiedades

Las propiedades de una clase se declaran indicando el modificador de acceso y luego su nombre.

```
class MiClase
{
        public $var1;
        protected $var2;
        private $_var3;
        public variable1 = "Primer Valor";  // String
        public variable2 = 1.23;  // Valor numérico
        public variable3 = array (1, 2, 3);
}
```

Al igual que una variable normal en PHP, la propiedad de una clase se puede inicializar mientras se está declarando. La inicialización se limita solamente a la asignación de valores, pero sería recomendable que dicha tarea se realice utilizando algún método (recomendablemente el constructor que veremos mas adelante).

Ejemplo:

```
<?php
        class Cliente
        {
                public $nombre,$dirección;
                private $_telefono=”33554488”;
```

```
        }

        $cliente= new Cliente();
                $cliente->nombre=”Antonio”;
                echo $cliente->nombre;
?>
```

Como podemos ver, al definir algunos atributos como *public* o otros *privated* tendremos acceso o no a cada uno de ellos. En el caso anterior obtendríamos el siguiente resultado:

```
Antonio
```

Pero si ahora declarásemos el siguiente objeto y quisiésemos ver la propiedad privada teléfono:

```
<?php
        $cliente= new Cliente();
                $cliente->nombre=”Antonio”;
                echo $cliente->_telefono;
?>

Fatal error: Cannot access private property Cliente::$_telefono in
C:\xampp\htdocs\programacion\clase7\index.php on line 10
```

De esta forma veremos que solamente podremos acceder a aquellos atributos declarados público y no a los declarados como privados.

Métodos

Como mencionamos anteriormente las clases pueden contener métodos y propiedades. Los métodos son declarados como funciones tradicionales y podremos hacer uso como lo hicimos en el apartado anterior de los modificadores de acceso.

```
<?php
        class MiClase
        {
                public  function saludar()
                {
                        echo “Hola a todos!!!!”;
                }
        }
        $objeto = new MiClase();
                $objeto ->saludar();
?>
```

Ejemplo:

```
<?php
        class Cliente
        {
                public  function saludar()
                {
                        echo “Hola a todos”;
                }
        }

        $cliente= new Cliente();
                $cliente->saludar();     // Hola a todos.
?>
```

Como se puede ver, luego de instanciar el objeto hacemos uso de → para llamar a sus métodos y atributos.
Si quisiésemos referenciar al mismo objeto *dentro de la declaración de la clase*, PHP nos provee de la variable *$this*. Esto nos será de utilidad para referenciar métodos o propiedades dentro del objeto a manera de interactuar con los mismos. Del ejemplo anterior:

```
<?php
        class Cliente
        {
                private function saludar()
                {
                        echo “Hola a todos”;
                }
                public  function saludo()
                {
                        $this->saludar();
                }
        }

        $cliente= new Cliente();
                $cliente->saludo();      // Hola a todos.
?>
```

De esta manera vemos que obtendremos el mismo resultado, pero esta vez la llamada al método *saludar()* la hacemos mediante *saludo()*.

Constructores

El *constructor* y el *destructor* son métodos especiales que se llaman, como su nombre indica, para la creación y destrucción de objetos, respectivamente.
Los constructores son útiles para inicializar las propiedades de un objeto, o para realizar los

procedimientos de inicio, como, por ejemplo, la conexión a una base de datos, o abrir un archivo remoto.
Los destructores en cambio liberarán los recursos utilizados por el objeto instanciado.

A partir de PHP5 podemos hacer uso del método *__construct()* para realizar dicha operación.

```
<?php
        class MiClase
        {
                function __construct()
                {
                        echo __METHOD__.PHP_EOL;
                }
        }

        $obj=new MiClase();
?>
```

Ejemplo:

```
<?php
        class Cliente
        {
                public $_textosaludo;
                function  __construct()
                {
                        $this->_textosaludo="Hola a todos!!!";
                }
                function saludo()
                {
                        echo $this->_textosaludo;
                }
        }

        $cliente= new Cliente();
                $cliente->saludo();      // Hola a todos!!!.
?>
```

En el ejemplo vemos que mediante __construct() inicializamos un atributo llamado $textosaludo (mas adelante en este capítulo veremos los atributos de un objeto) con el texto del saludo. Esta acción ocurrirá en el preciso momento en que se este declarando el objeto ($cliente=new Cliente()). Luego podremos hacer uso de la misma desde cualquier método.

Destructores

Así como existe el método *__construct()*, tenemos el método *__destruct()* el cual será llamado siempre antes de que un objeto sea eliminado o destruido. Es útil para realizar limpieza en procedimientos, cerrar conexiones a base de datos, borrar archivos temporales, etc.

```
<?php
        class MiClase
        {

                function __construct()
                {
                        echo __METHOD__.PHP_EOL;
                }
                function __destruct()
                {
                        echo __METHOD__;
                }
        }
        $obj=new MiClase();
?>
```

La salida de este código será:

```
MiClase::__construct
MiClase::__destruct
```

Ejemplo:

```
<?php
        class Cliente
        {
                public $_textosaludo;

                public function  __construct()
                {
                        $this->_textosaludo="Hola a todos!!!";
                }
                public function  __destruct()
                {
                        $this->_textosaludo="";
                }

                public function saludo()
                {
                        echo $this->_textosaludo;
                }
        }

        $cliente= new Cliente();
```

```
                $cliente->saludo();      // Hola a todos!!!.
        ?>
```

En el ejemplo veremos que justo antes de destruirse el objeto (luego de la llamada al método saludo()), se llamará al __destruct() y éste pondrá en cero el atributo $_textosaludo.

toString()

De la misma manera que encontramos el *__construct()* y el *__destruct(),* tenemos el método toString(). Básicamente éste método nos permitirá definir un string con la estructura que nosotros queramos para que en cualquier momento de la aplicación podamos obtener el estado (conjunto de valores) de un objeto.
Éste método es llamado de manera predeterminada en PHP ante un *echo* o un *print* de un objeto en si. Esto quiere decir que al encontrar una instrucción *echo $miObjeto* o print (*$miObjeto)* ejecutará automáticamente dicho método.

```
<?php
        class MiClase
        {

                function  getNombre()
                {
                        return “Pedro”;
                }

                function __toString()
                {
                        $desc  = “(“.$this->getNombre();
                        $desc .= “)”:
                    return $desc;
                }
        }
        $obj=new MiClase();
?>
```

La salida de este código será:

```
$miClase = new MiClase();
echo $miClase;          // Resultado (Pedro);
```

Ejemplo:

```
<?php
        class Cliente
```

```
{
        private $_apellido,$_nombre,$_direccion;

        public function __construct()
        {
                $this_apellido=”Sosa”;
                $this_nombre=”Juan”;
                $this_direccion=”Los Espinillos”;
        }


        public function toString()
        {
                $desc=”(“.$this->_apellido;
                $desc.=”-“.$this->_nombre;
                $desc=”-“.$this->_direccion.”)”;
        }
}

$cliente= new Cliente();
        echo $cliente; // (Sosa-Juan-Los Espinillos)
?>
```

Herencia

Uno de los conceptos fundamentales de la programación orientada a objetos es la *herencia*. Esto permite a una clase ampliar sus métodos y atributos a partir de otra, en base a la adición de nuevos métodos y propiedades. Por ejemplo

```
<?php
    class A
     {
        function test()
        {
            echo "a::llamada a la función test";
        }

        function func()
        {
            echo "a::llamada a la función func ";
        }
    }

    class B extends A
    {
        function test()
        {
            echo "b::llamada a la función test";
        }
```

```
        }

        class C extends B
        {
                function test()
                {
                        parent::test();
                }
        }

        class D extends C
        {
                function test()
                {
                        B::test();
                }
        }

        $a = new A();
        $b = new B();
        $c = new C();
        $d = new D();
        $a->test();     // Resultado  "llamada a la función test"
        $b->test();     // Resultado  "b::llamada a la función test"
        $b->func();     // Resultado  "a::llamada a la función func "
        $c->test();     // Resultado  "b::llamada a la función test"
        $d->test();     // Resultado  "b::llamada a la función test"
?>
```

En este script, empezamos por declarar una clase llamada *A*. A continuación, declaramos la clase *B*, que se extiende *A*. Como puede ver, esta clase también tiene un método *test()*, que reemplaza al declarado en *A*, por lo que la salida de éste daría: *b::llamada a la función test*. Además se puede tener acceso a todos los métodos definidos en *A* a través de *B,* es decir, podemos llamar *$ b->func()*, siendo que *func()* fue definido en la clase *A.*

Como podemos ver esto nos da una gran flexibilidad ya que podemos reutilizar los métodos y propiedades declarados en los objeto padres. Para reverenciarlos podremos hacer uso de *parent:método()* como lo hicimos en la declaración de la clase *C*, o directamente como lo hicimos en *D*.