Manual de PHP 5

Manual por: DesarrolloWeb.com [http://www.desarrolloweb.com/] **Versión on-line:**
"Tu mejor ayuda para aprender a hacer webs" http://www.desarrolloweb.com/manuales/58

Introducción a PHP 5

Vamos a comenzar con el manual de la última versión de PHP, lanzada recientemente al mercado: PHP 5. Una esperada evolución del, tal vez, más popular de los lenguajes de programación de páginas y aplicaciones web del lado del servidor.

Este manual no va a tratar de explicar desde cero la programación de aplicaciones del lado del servidor con PHP, pues ese asunto lo tenemos detallado en el manual de PHP [http://www.desarrolloweb.com/manuales/12/] y pensamos que merece la pena su lectura para empezar los primeros pasos en el lenguaje.

Si una persona no sabe lo que son las páginas dinámicas de servidor le recomendamos que comience aclarando esos conceptos. Para ello, tenemos dos manuales que explican las nociones de programación del lado del cliente y servidor, más bien teóricas, desde dos puntos de vista: Manual de páginas dinámicas [http://www.desarrolloweb.com/manuales/7/] y la Introducción a los lenguajes del web [http://www.desarrolloweb.com/manuales/27/].

Asimismo, queremos presentarte la sección monotemática de PHP [http://www.desarrolloweb.com/php], donde se concentran todos los contenidos sobre PHP que dispone DesarrolloWeb.com. También sería interesante el Manual del lenguaje SQL [http://www.desarrolloweb.com/manuales/9/] y el Taller de MySQL [http://www.desarrolloweb.com/manuales/34/], que serán muy útiles referencias para aclarar los conceptos de acceso a bases de datos.

Introducción a PHP 5

Con las primeras 2 versiones de PHP, PHP 3 y PHP 4, se había conseguido una plataforma potente y estable para la programación de páginas del lado del servidor. Estas versiones han servido de mucha ayuda para la comunidad de desarrolladores, haciendo posible que PHP sea el lenguaje más utilizado en la web para la realización de páginas avanzadas.

Sin embargo, todavía existían puntos negros en el desarrollo PHP que se han tratado de solucionar con la versión 5, aspectos que se echaron en falta en la versión 4, casi desde el día de su lanzamiento. Nos referimos principalmente a la programación orientada a objetos (POO) que, a pesar de que estaba soportada a partir de PHP3, sólo implementaba una parte muy pequeña de las características de este tipo de programación.

> **Nota:** la orientación a objetos es una manera de programar que trata de modelar los procesos de programación de una manera cercana a la realidad: tratando a cada componente de un programa como un objeto con sus características y funcionalidades. Podemos ver una pequeña introducción en el artículo Qué es la programacion orientada a objetos [http://www.desarrolloweb.com/articulos/499.php].

El principal objetivo de PHP5 ha sido mejorar los mecanismos de POO para solucionar las carencias de las anteriores versiones. Un paso necesario para conseguir que PHP sea un lenguaje apto para todo tipo de aplicaciones y entornos, incluso los más exigentes.

Instalación de PHP5 con WAMP5

Existe una manera de comenzar a utilizar PHP5 en Windows sin tener que sufrir las complicaciones típicas de la instalación de los servidores necesarios para programar en PHP. Se trata de instalar un paquete llamado WAMP, que permite instalar y configurar en un solo

proceso el servidor Apache, la base de datos MySQL y el módulo de programación en PHP versión 5.

WAMP es un sistema indicado para los usuarios que no tienen instalado en el sistema ninguno de los programas necesarios para programar en PHP (Apache, PHP y MySQL), ya que realiza una instalación completa y desde cero. Pero también pueden utilizar este programa los usuarios que disponen de Apache, PHP y/o MySQL en su sistema. En cuyo caso, simplemente se realizará otra copia de las aplicaciones en un directorio distinto, que en principio, no tiene por qué interferir con las otras instalaciones alojadas en nuestro equipo.

Programas que contiene WAMP5

El software que se instala con WAMP5 contiene los siguientes servidores y programas:

- Apache 1.3.31. El servidor de páginas web más extendido del mercado. Aunque la última versión de este servidor es Apache 2, se instala una versión anterior que resulta más estable. Existe un Add-on que permite sustituir la versión 1.3.31 de Apache por la última versión.
- PHP5. El motor renovado del lenguaje.
- MySQL. La base de datos más extendida para utilizar con PHP.
- PHPmyadmin. Un software que permite administrar una base de datos a través de una interfaz web.
- SQLitemanager. Un sistema para administrar una base de datos a partir de sentencias SQL.

Instalación de WAMP

La instalación se realiza a través de un ejecutable Windows donde se pueden introducir muy pocas configuraciones, apenas el directorio donde deseamos que se instalen los programas. Después del proceso de instalación se habrán creado un par de servicios con el servidor web y el de bases de datos:

- Servicio wampapache: Relacionado con el servidor Apache.
- Servicio wampmysql: Relacionado con la base de datos MySQL.

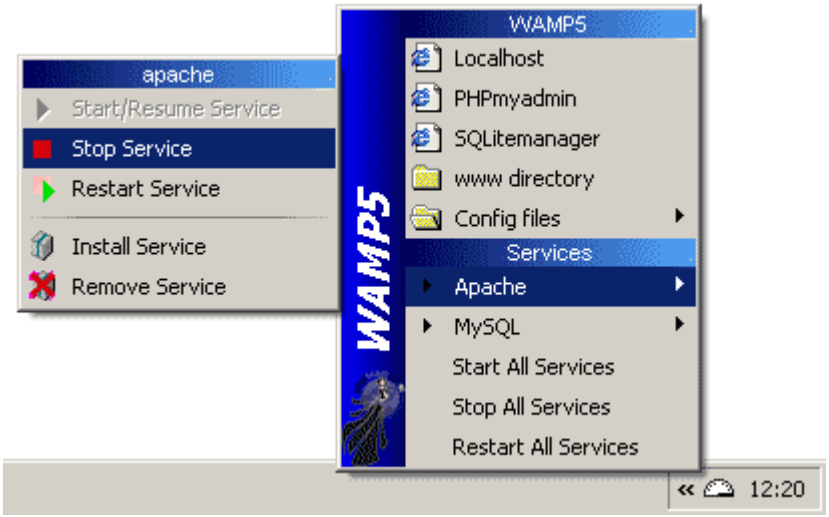
Además, dentro del directorio donde hayamos instalado WAMP5 se habrá creado una carpeta llamada "www", que corresponde con el directorio de publicación, es decir, el lugar donde se deben colocar las páginas web.

Durante la instalación también debemos decidir si deseamos que WAMP5 se inicie automáticamente al arrancar el equipo o si deseamos que su puesta en marcha se realice manualmente.

Puesta en marcha de los servidores

Cuando instalamos WAMP5 se crea un grupo de programas llamado WampServer, donde podremos encontrar una opción que pone "Start Wampserver", que será necesario ejecutar si no hemos seleccionado que el servidor se inicie automáticamente.

Una vez arrancado WampServer dispondremos de un icono en la barra de tareas con una forma similar a la de un marcador de velocidad. Si pulsamos sobre ese icono se abrirá un menú con opciones variadas para gestionar los servicios relacionados con el paquete. La imagen de abajo muestra ese menú.



Podemos probar si los servicios están corriendo perfectamente accediendo a la página de inicio del servidor, escribiendo en la barra de direcciones de nuestro navegador algo como http://localhost/.

Entonces nos debería de aparecer una página con varios enlaces a las distintas herramientas instaladas con WAMP5, además de algunas páginas de prueba de PHP.

Add-ons

Existen varios añadidos que se pueden instalar con WAMP, para ampliar las posibilidades del paquete. Por ejemplo, podemos instalar un add-on para permitir que WAMP trabaje con PHP5 o bien PHP4, creando una nueva opción en el menú de WAMP5 que permite cambiar de una versión a otra de PHP.

Hay otros añadidos disponibles:

- Instalar ActiveState Perl en nuestro sistema, para permitir la ejecución de CGI.
- Actualizar a la versión de Apache 2.
- Instalar Zend Optimizer, para mejorar el comportamiento en tiempo de ejecución de PHP.
- Por último, el add-on que instala Webalizer, un sistema para obtener estadísticas de uso del servidor web.

Se puede obtener más información de este sistema y opciones para la descarga en la página http://www.en.wampserver.com/

Modelo de orientación a objetos de PHP 3 y 4

La versión 3 de PHP ya soportaba la programación orientada a objetos (POO), aunque es verdad que la mayoría de las características de este tipo de programación no estaban implementadas todavía. En concreto, con PHP3 podíamos crear clases e instanciar objetos. Las clases permitían agrupar tanto métodos como propiedades o atributos, pero la cosa se quedaba ahí.

En PHP4, se reescribió el motor de PHP para hacerlo mucho más rápido y estable, pero la POO, que había introducido la anterior versión del lenguaje, no se llegó a modificar prácticamente. Aun así, durante la vigencia de PHP 4, la programación orientada a objetos fue utilizada habitualmente, a menudo en aplicaciones de gran tamaño. Entornos donde se puso de manifiesto la falta de potencia de la POO en PHP 4 y la necesidad de mejorarla en una nueva versión.

El mayor problema de la POO en las versiones 3 y 4 de PHP se basaba en que, cada vez que se asignaba una variable que contenía un objeto a otra variable, o se pasaba un objeto por parámetro en una función, se realizaba una copia (un clon) de ese objeto y quedaba a disposición del programa en la nueva variable o parámetro.

```
$pepe = new persona("pepe");
$pepe2 = $pepe;
```

En un código como el anterior, se tiene un objeto persona alojado en la variable $pepe y en la segunda línea de código, se crea un clon de $pepe y se asigna a la variable $pepe2. En este caso y siempre siguiendo el anterior modo de trabajo de PHP, aunque $pepe y $pepe2 contienen un objeto idéntico, no se trata del mismo objeto sino de una copia. Todo esto implica que el espacio en memoria para guardar los dos objetos es el doble que si fuera un mismo objeto con dos nombres distintos.

Esta situación ocurría porque los objetos eran tratados del mismo modo que las variables normales, que se pasan por valor en las funciones y en caso de asignarse, se realiza una copia de la variable antes de asignarse al nuevo espacio.

Ejemplo del modo de trabajo con objetos de PHP 3 y 4

Vamos a realizar un ejemplo para ilustrar el modo de trabajo de PHP 3 y 4 con los objetos. En este ejemplo podrá quedar patente el proceso de clonación de los objetos al ser pasados en una función o al asignarse a otra variable.

Primero veamos una declaración de un objeto muy simple. Se trata de una "caja" que tiene un atributo que es el contenido y dos métodos, uno para introducir nuevos contenidos en la caja y otro para mostrar el contenido actual de la caja.

```
class Caja{
var $contenido;

function introduce($cosa){
$this->contenido = $cosa;
}

function muestra_contenido(){
echo $this->contenido;
}
}
```

Ahora vamos a ver unas pocas líneas de código que hacen uso de la clase Caja para ilustrar el modo de trabajo de los objetos en PHP 4. Vamos a instanciar el objeto, luego lo asignamos a otra variable, con lo que se creará un clon de ese objeto, continuamos modificando el clon y veremos que pasa.

```
$micaja = new Caja();
$micaja->introduce("algo");
$micaja->muestra_contenido();

echo "<br>";

$segunda_caja = $micaja;
$segunda_caja->introduce("contenido en segunda caja");
$segunda_caja->muestra_contenido();

echo "<br>";

$micaja->muestra_contenido();
```

En la primera línea de código se instancia la caja y se aloja el objeto en la variable $micaja. En la segunda línea se introduce el string "algo" en el contenido de la caja. Luego se muestra el contenido, con lo que saldrá el string "algo" en la página web.

En el segundo bloque de código se asigna el objeto $micaja a la variable $segunda_caja, con lo que se crea el mencionado clon del objeto $micaja y se asigna a la nueva variable. Luego se introduce un nuevo contenido a la instancia alojada en la variable $segunda_caja. Atención aquí, porque se ha modificado el clon alojado en la variable $segunda_caja, dejando inalterable el objeto original $micaja.

Para comprobarlo, se muestra el contenido del objeto $segunda_caja, con lo que aparece en la página web el string "contenido en segunda caja". También se muestra el contenido de $micaja, que no se ha modificado a pesar de actualizar el contenido de su clon, con lo que se muestra el string "algo".

Espero que no sea demasiado difícil de entender. Podéis hacer la prueba por vosotros mismos para comprender bien el ejercicio. De todos modos, vamos a hacer otro ejemplo en el que se utiliza la clase Caja, que esperamos sirva para aclarar mejor el trabajo con objetos en PHP 3 y 4.

```
$micaja = new Caja();
$micaja->introduce("algo");
$micaja->muestra_contenido();

echo "<br>";

function vacia_caja($caja_vaciar){
$caja_vaciar->introduce("polvo");
}

vacia_caja($micaja);

$micaja->muestra_contenido();
```

En este ejemplo hemos creado una función que recibe por parámetro un objeto de la clase caja. Como los parámetros en las funciones se reciben por valor en lugar de referencia, cuando se pasa el parámetro del objeto caja, en el fondo lo que se está realizando es una copia de ese objeto, de modo que dentro de la función se trabaja con un clon del objeto, en lugar del objeto mismo.

En el código se instancia el objeto caja y se introduce "algo" en su contenido. Luego, se declara una función que recibe el objeto y modifica su contenido, introduciendo el string "polvo" en el contenido de la caja. En las siguientes líneas de código, se llama a la función declarada anteriormente, pasando por parámetro el objeto $micaja. Dentro de la función, como decía, se modifica el contenido de la caja, aunque realmente se está modificando el contenido de un clon.

Por último, se muestra el contenido del objeto $micaja. En este caso aparece "algo", a pesar de que en la función ese "algo" se modificó por "polvo". A pesar de poder parecer pesado, vuelvo a repetir que en la función se modificó un clon del objeto y no el objeto original.

Los comportamientos descritos anteriormente no son muy habituales en otros lenguajes de programación orientada a objetos, como Java, donde el objeto no se duplica cada vez que se realiza una asignación o paso por parámetro.

Para evitar el comportamiento que hemos descrito, PHP dispone de la opción de paso de parámetros por referencia, que se realiza con el carácter "&". Por ejemplo, para asignar el propio objeto y no un clon podríamos haber utilizado este código:

```
$segunda_caja = &$micaja;
```

Para recibir un parámetro por referencia en lugar de por valor en una función utilizaríamos esta declaración de función:

```
function vacia_caja(&$caja_vaciar){
```

La posibilidad de utilizar el carácter "&" para forzar un paso por referencia no deja de ser un problema, puesto que nos obliga a utilizar ese mecanismo en múltiples lugares y es muy fácil olvidarse del "&" en algún sitio, con lo que nuestro programa ya no realizará los resultados esperados. Muchos programadores han gastado horas en encontrar el problema y en cualquier caso, es una molestia tener que estar pendientes de incluir constantemente el signo "&" en el código para hacer que funcione como ellos desean.

Modelo de orientación a objetos en PHP 5

En el artículo anterior comentamos las carencias del modelo de orientación a objetos en PHP 3 y 4, que afortunadamente han quedado solventadas en la versión PHP 5.

Como decíamos, uno de los problemas más básicos de las versiones anteriores de PHP era la clonación de objetos, que se realizaba al asignar un objeto a otra variable o al pasar un objeto por parámetro en una función. Para solventar este problema PHP5 hace uso de los manejadores de objetos (Object handles), que son una especie de punteros que apuntan hacia los espacios en memoria donde residen los objetos. Cuando se asigna un manejador de objetos o se pasa como parámetro en una función, se duplica el propio object handle y no el objeto en si.

> **Nota:** También se puede realizar una clonación de un objeto, para obtener una copia exacta, pero que no es el propio objeto. Para ello utilizamos una nueva instrucción llamada "clone", que veremos más adelante.

Algunas características del trabajo con POO en PHP 5

Veamos a continuación una pequeña lista de las nuevas características de la programación orientada a objetos (POO) en PHP5. No vamos a describir exhaustivamente cada característica. Ya lo haremos más adelante en este mismo manual.

1.- Nombres fijos para los constructores y destructores
En PHP 5 hay que utilizar unos nombres predefinidos para los métodos constructores y destructores (Los que se encargan de resumir las tareas de inicialización y destrucción de los objetos. Ahora se han de llamar __construct() y __destruct().

2.- Acceso public, private y protected a propiedades y métodos
A partir de ahora podemos utilizar los modificadores de acceso habituales de la POO. Estos modificadores sirven para definir qué métodos y propiedades de las clases son accesibles desde cada entorno.

3.- Posibilidad de uso de interfaces
Las interfaces se utilizan en la POO para definir un conjunto de métodos que implementa una clase. Una clase puede implementar varias interfaces o conjuntos de métodos. En la práctica, el uso de interfaces es utilizado muy a menudo para suplir la falta de herencia múltiple de lenguajes como PHP o Java. Lo explicaremos con detalle más adelante.

4.- Métodos y clases final
En PHP 5 se puede indicar que un método es "final". Con ello no se permite sobrescribir ese método, en una nueva clase que lo herede. Si la clase es "final", lo que se indica es que esa clase no permite ser heredada por otra clase.

5.- Operador instanceof
Se utiliza para saber si un objeto es una instancia de una clase determinada.

6.- Atributos y métodos static
En PHP5 podemos hacer uso de atributos y métodos "static". Son las propiedades y funcionalidades a las que se puede acceder a partir del nombre de clase, sin necesidad de haber instanciado un objeto de dicha clase.

7.- Clases y métodos abstractos
También es posible crear clases y métodos abstractos. Las clases abstractas no se pueden instanciar, se suelen utilizar para heredarlas desde otras clases que no tienen porque ser abstractas. Los métodos abstractos no se pueden llamar, se utilizan más bien para ser heredados por otras clases, donde no tienen porque ser declarados abstractos.

8.- Constantes de clase
Se pueden definir constantes dentro de la clase. Luego se pueden acceder dichas constantes a través de la propia clase.

9.- Funciones que especifican la clase que reciben por parámetro
Ahora se pueden definir funciones y declarar que deben recibir un tipo específico de objeto. En caso que el objeto no sea de la clase correcta, se produce un error.

10.- Función __autoload()
Es habitual que los desarrolladores escriban un archivo por cada clase que realizan, como técnica para organizar el código de las aplicaciones. Por esa razón, a veces resulta tedioso realizar los incluyes de cada uno de los códigos de las clases que se utilizana en un script. La función __autoload() sirve para intentar incluir el código de una clase que se necesite, y que no haya sido declarada todavía en el código que se está ejecutando.

11.- Clonado de objetos
Si se desea, se puede realizar un objeto a partir de la copia exacta de otro objeto. Para ello se utiliza la instrucción "clone". También se puede definir el método __clone() para realizar tareas asociadas con la clonación de un objeto.

Clases en PHP 5

Clases en PHP 5

Las clases en Programación orientada a objetos (POO) son definiciones de los elementos que forman un sistema, en este caso, definiciones de los objetos que van a intervenir en nuestros programas.

Un objeto se define indicando qué propiedades y funcionalidades tiene. Justamente esas declaraciones son lo que es una clase. Cuando se hace una clase simplemente se especifica qué propiedades y funcionalidades tiene. Por ejemplo, un hombre podría tener como propiedades el nombre o la edad y como funcionalidades, comer, moverse o estudiar.

En la clase hombre declararíamos dos atributos: la edad o el nombre, que serían como dos variables. También deberíamos crear tres métodos, con los procedimientos a seguir para que el hombre pueda comer, moverse o estudiar. Estos métodos se definen declarando funciones dentro de la clase.

El código para definir una clase se puede ver a continuación:

```
class hombre{
var $nombre;
var $edad;

function comer($comida){
//aquí el código del método
}

function moverse($destino){
//aquí el código del método
}

function estudiar($asignatura){
//aquí el código del método
}
```

```
}
```

Podrá comprobarse que este código no difiere en nada del de las versiones anteriores de PHP, que ya soportaban ciertas características de la POO. Esta situación cambiará a poco que exploremos las características más avanzadas de PHP 5, que implicarán mejoras que no estaban presentes en las versiones anteriores

Instanciar objetos a partir de clases

Hemos visto que una clase es tan sólo una definición. Si queremos trabajar con las clases debemos instanciar objetos, proceso que consiste en generar un ejemplar de una clase.

Por ejemplo, tenemos la clase hombre anterior. Con la clase en si no podemos hacer nada, pero podemos crear objetos hombre a partir de esa clase. Cada objeto hombre tendrá unas características propias, como la edad o el nombre. Además podrá desempeñar unas funciones como comer o moverse, ahora bien, cada uno comerá o se moverá por su cuenta cuando le sea solicitado, sin interferir en principio con lo que pueda estar haciendo otro hombre.

Ya que estamos, vamos a ver cómo se generarían un par de hombres, es decir, cómo se instanciarían un par de objetos de la clase hombre. Para ello utilizamos el operador new.

```
$pepe = new hombre();
$juan = new hombre();
```

Conclusión

Es importante darse cuenta de la diferencia entre un objeto y una clase. La clase es una definición de unas características y funcionalidades, algo abstracto que se concreta con la instanciación de un objeto de dicha clase.
Un objeto ya tiene propiedades, con sus valores concretos, y se le pueden pasar mensajes (llamar a los métodos) para que hagan cosas.

Constructores en PHP 5

Los constructores se encargan de resumir las acciones de inicialización de los objetos. Cuando se instancia un objeto, se tienen que realizar varios pasos en su inicialización, por ejemplo dar valores a sus atributos y eso es de lo que se encarga el constructor. Los constructores pueden recibir unos datos para inicializar los objetos como se desee en cada caso.

La sintaxis para la creación de constructor varía con respecto a la de PHP 3 y 4, pues debe llamarse con un nombre fijo: __construct(). (Son dos guiones bajos antes de la palabra "construct")

A lo largo de los ejemplos de este manual vamos a ir creando un código para gestión de un videoclub. Vamos a empezar definiendo una clase cliente, que utilizaremos luego en nuestro programa.

```
class cliente{
   var $nombre;
   var $numero;
   var $peliculas_alquiladas;

   function __construct($nombre,$numero){
      $this->nombre=$nombre;
      $this->numero=$numero;
      $this->peliculas_alquiladas=array();
   }

   function dame_numero(){
      return $this->numero;
   }
}
```

El constructor en esta clase recibe el nombre y número que asignar al cliente, que introduce luego en sus correspondientes propiedades. Además inicializa el atributo películas_alquiladas como un array, en este caso vacío porque todavía no tiene ninguna película en su poder.

Nota: En programación orientada a objetos $this hace referencia al objeto sobre el que se está ejecutando el método. En este caso, como se trata de un constructor, $this hace referencia al objeto que se está construyendo. Con $this->nombre=$nombre; estamos asignando al atributo "nombre" del objeto que se está construyendo el valor que contiene la variable $nombre, que se ha recibido por parámetro.

Luego hemos creado un método muy sencillo para poder utilizar el objeto. Vamos a ver unas acciones simples para ilustrar el proceso de instanciación y utilización de los objetos.

```
//instanciamos un par de objetos cliente
$cliente1 = new cliente("Pepe", 1);
$cliente2 = new cliente("Roberto", 564);

//mostramos el numero de cada cliente creado
echo "El identificador del cliente 1 es: " . $cliente1->dame_numero();
echo "El identificador del cliente 2 es: " . $cliente2->dame_numero();
```

Este ejemplo obtendría esta salida como resultado de su ejecución:

El identificador del cliente 1 es: 1
El identificador del cliente 2 es: 564

Destructores en PHP 5

Los destructores son funciones que se encargan de realizar las tareas que se necesita ejecutar cuando un objeto deja de existir. Cuando un objeto ya no está referenciado por ninguna variable, deja de tener sentido que esté almacenado en la memoria, por tanto, el objeto se debe destruir para liberar su espacio. En el momento de su destrucción se llama a la función destructor, que puede realizar las tareas que el programador estime oportuno realizar.

La creación del destructor es opcional. Sólo debemos crearlo si deseamos hacer alguna cosa cuando un objeto se elimine de la memoria.

El destructor es como cualquier otro método de la clase, aunque debe declararse con un nombre fijo: __destruct().

En el código siguiente vamos a ver un destructor en funcionamiento. Aunque la acción que realiza al destruirse el objeto no es muy útil, nos puede servir bien para ver cómo trabaja.

```
class cliente{
   var $nombre;
   var $numero;
   var $peliculas_alquiladas;

   function __construct($nombre,$numero){
      $this->nombre=$nombre;
      $this->numero=$numero;
      $this->peliculas_alquiladas=array();
   }

   function __destruct(){
      echo "<br>destruido: " . $this->nombre;
   }

   function dame_numero(){
      return $this->numero;
   }
}

//instanciamos un par de objetos cliente
$cliente1 = new cliente("Pepe", 1);
```

```
$cliente2 = new cliente("Roberto", 564);

//mostramos el numero de cada cliente creado
echo "El identificador del cliente 1 es: " . $cliente1->dame_numero();
echo "<br>El identificador del cliente 2 es: " . $cliente2->dame_numero();
```

Este código es igual que el anterior. Sólo se ha añadido el destructor, que imprime un mensaje en pantalla con el nombre del cliente que se ha destruido. Tras su ejecución obtendríamos la siguiente salida.

```
El identificador del cliente 1 es: 1
El identificador del cliente 2 es: 564
destruido: Pepe
destruido: Roberto
```

Como vemos, antes de acabar el script, se libera el espacio en memoria de los objetos, con lo que se ejecuta el destructor y aparece el correspondiente mensaje en la página.

Un objeto puede quedar sin referencias y por lo tanto ser destruido, por muchas razones. Por ejemplo, el objeto puede ser una variable local de una función y al finalizarse la ejecución de esa función la variable local dejaría de tener validez, con lo que debe destruirse.

El código siguiente ilustra cómo una variable local a cualquier ámbito (por ejemplo, local a una función), se destruye cuando ese ámbito ha finalizado.

```
function crea_cliente_local(){
   $cliente_local = new cliente("soy local", 5);
}
crea_cliente_local()
```

La función simplemente crea una variable local que contiene la instanciación de un cliente. Cuando la función se acaba, la variable local deja de existir y por lo tanto se llama al destructor definido para ese objeto.

> **Nota:** También podemos deshacernos de un objeto sin necesidad que acabe el ámbito donde fue creado. Para ello tenemos la función unset() que recibe una variable y la elimina de la memoria. Cuando se pierde una variable que contiene un objeto y ese objeto deja de tener referencias, se elimina al objeto y se llama al destructor.

Modificadores de acceso a métodos y propiedades en PHP5

Veremos en este capítulo los nuevos modificadores de acceso a los métodos y atributos de los objetos que se han incorporado en PHP 5. Estos modificadores de acceso no son otros que los conocidos public, protected y private, que ya disponen otros lenguajes como Java.

Uno de los principios de la programación orientada a objetos es la encapsulación, que es un proceso por el que se ocultan las características internas de un objeto a aquellos elementos que no tienen porque conocerla. Los modificadores de acceso sirven para indicar los permisos que tendrán otros objetos para acceder a sus métodos y propiedades.

Modificador public

Es el nivel de acceso más permisivo. Sirve para indicar que el método o atributo de la clase es público. En este caso se puede acceder a ese atributo, para visualizarlo o editarlo, por cualquier otro elemento de nuestro programa. Es el modificador que se aplica si no se indica otra cosa.

Veamos un ejemplo de clase donde hemos declarado como public sus elementos, un método y una propiedad. Se trata de la clase "dado", que tiene un atributo con su puntuación y un método para tirar el dado y obtener una nueva puntuación aleatoria.

```
class dado{
   public $puntos;

   function __construct(){
     srand((double)microtime()*1000000);
   }

   public function tirate(){
     $this->puntos=$randval = rand(1,6);
   }
}

$mi_dado = new dado();

for ($i=0;$i<30;$i++){
   $mi_dado->tirate();
   echo "<br>Han salido " . $mi_dado->puntos . " puntos";
}
```

Vemos la declaración de la clase dado y luego unas líneas de código para ilustrar su funcionamiento. En el ejemplo se realiza un bucle 30 veces, en las cuales se tira el dado y se muestra la puntuación que se ha obtenido.

Como el atributo $puntos y el método tirate() son públicos, se puede acceder a ellos desde fuera del objeto, o lo que es lo mismo, desde fuera del código de la clase.

Modificador private

Es el nivel de acceso más restrictivo. Sirve para indicar que esa variable sólo se va a poder acceder desde el propio objeto, nunca desde fuera. Si intentamos acceder a un método o atributo declarado private desde fuera del propio objeto, obtendremos un mensaje de error indicando que no es posible a ese elemento.

Si en el ejemplo anterior hubiéramos declarado private el método y la propiedad de la clase dado, hubiéramos recibido un mensaje de error.

Aquí tenemos otra posible implementación de la clase dado, declarando como private el atributo puntos y el método tirate().

```
class dado{
   private $puntos;

   function __construct(){
     srand((double)microtime()*1000000);
   }

   private function tirate(){
     $this->puntos=$randval = rand(1,6);
   }

   public function dame_nueva_puntuacion(){
     $this->tirate();
     return $this->puntos;
   }
}

$mi_dado = new dado();

for ($i=0;$i<30;$i++){
   echo "<br>Han salido " . $mi_dado->dame_nueva_puntuacion() . " puntos";
}
```

Hemos tenido que crear un nuevo método público para operar con el dado, porque si es todo privado no hay manera de hacer uso de él. El mencionado método es dame_nueva_puntuación (), que realiza la acción de tirar el dado y devolver el valor que ha salido.

Modificador protected

Este indica un nivel de acceso medio y un poco más especial que los anteriores. Sirve para que el método o atributo sea público dentro del código de la propia clase y de cualquier clase que herede de aquella donde está el método o propiedad protected. Es privado y no accesible desde cualquier otra parte. Es decir, un elemento protected es público dentro de la propia clase y en sus heredadas.

Más adelante explicaremos la herencia y podremos ofrecer ejemplos con el modificador protected.

Conclusión

Muchas veces el propio desarrollador es el que fija su criterio a la hora de aplicar los distintos modificadores de acceso a atributos y métodos. Poca protección implica que los objetos pierdan su encapsulación y con ello una de las ventajas de la POO. Una protección mayor puede hacer más laborioso de generar el código del programa, pero en general es aconsejable.

La herencia en PHP5

La herencia es uno de los mecanismos fundamentales de la programación orientada a objetos. Por medio de la herencia, se pueden definir clases a partir de la declaración de otras clases. Las clases que heredan incluyen tanto los métodos como las propiedades de la clase a partir de la que están definidos.

Por ejemplo, pensemos en la clase "vehículo". Esta clase general puede incluir las características generales de todos los vehículos (atributos de la clase), como la matrícula, año de fabricación y potencia. Además, incluirá algunas funcionalidades (métodos de la clase) como podrían ser, arrancar() o moverse().

Ahora bien, en la práctica existen varios tipos de vehículos, como los coches, los autobuses y los camiones. Todos ellos tienen unas características comunes, que han sido definidas en la clase vehículo. Además, tendrán una serie de características propias del tipo de vehículo, que, en principio, no tienen otros tipos de vehículos. Por ejemplo, los camiones pueden tener una carga máxima permitida o los autobuses un número de plazas disponibles. Del mismo modo, las clases más específicas pueden tener unas funcionalidades propias, como los camiones cargar() y descargar(), o los autobuses aceptar_pasajeros() o vender_billete().

Lo normal en sistemas de herencia es que las clases que heredan de otras incluyan nuevas características y funcionalidades, aparte de los atributos y métodos heredados. Pero esto no es imprescindible, de modo que se pueden crear objetos que hereden de otros y no incluyan nada nuevo.

Sintaxis de herencia en PHP 5

La programación orientada a objetos nos ofrece una serie de mecanismos para definir este tipo de estructuras, de modo que se puedan crear jerarquías de objetos que heredan unos de otros. Veremos ahora cómo definir estas estructuras de herencia en PHP 5. Para ello, continuando con nuestro ejemplo de video club, vamos a crear los distintos tipos de elementos que se ofrecen en alquiler.

Como todo el mundo conoce, los video clubs ofrecen distintos tipos de elementos para alquiler, como pueden ser las películas (cintas de vídeo o DVD) y los juegos. Cada elemento tiene unas características propias y algunas comunes. Hemos llamado "soporte" a la clase general, que incluye las características comunes para todos los tipos de elementos en alquiler. Luego hemos creado tres tipos de soportes distintos, que heredan de la clase soporte, pero que incluyen algunas características y funcionalidades nuevas. Estos tipos de soporte serán "cinta_video", "dvd" y "juego".

El esquema de herencia que vamos a realizar en este ejemplo se puede ver en la siguiente imagen.

Empezamos por la clase soporte. Su código será el siguiente:

```
class soporte{
  public $titulo;
  protected $numero;
  private $precio;

  function __construct($tit,$num,$precio){
    $this->titulo = $tit;
    $this->numero = $num;
    $this->precio = $precio;
  }

  public function dame_precio_sin_iva(){
    return $this->precio;
  }

  public function dame_precio_con_iva(){
    return $this->precio * 1.16;
  }

  public function dame_numero_identificacion(){
    return $this->numero;
  }

  public function imprime_caracteristicas(){
     echo $this->titulo;
     echo "<br>" . $this->precio . " (IVA no incluido)";
  }
}
```

Los atributos que hemos definido son, título, numero (un identificador del soporte) y precio. Hemos aplicado a cada uno un modificador de acceso distinto, para poder practicar los distintos tipos de acceso.

Hemos definido un constructor, que recibe los valores para la inicialización del objeto. Un método dame_precio_sin_iva(), que devuelve el precio del soporte, sin aplicarle el IVA. Otro método dame_precio_con_iva(), que devuelve el precio una vez aplicado el 16% de IVA. El método dame_numero_identificacion(), que devuelve el número de identificador y imprime_caracteristicas(), que muestra en la página las características de este soporte.

> **Nota:** Como se ha definido como private el atributo precio, este atributo sólo se podrá acceder dentro del código de la clase, es decir, en la propia definición del objeto. Si queremos acceder al precio desde fuera de la clase (algo muy normal si tenemos en cuenta que vamos a necesitar el precio de un soporte desde otras partes de la aplicación) será necesario crear un método que nos devuelva el valor del precio. Este método debería definirse como public, para que se pueda acceder desde cualquier sitio que se necesite.

En todos los métodos se hace uso de la variable $this. Esta variable no es más que una

referencia al objeto sobre el que se está ejecutando el método. En programación orientada a objetos, para ejecutar cualquiera de estos métodos, primero tenemos que haber creado un objeto a partir de una clase. Luego podremos llamar los métodos de un objeto. Esto se hace con $mi_objeto->metodo_a_llamar(). Dentro de método, cuando se utiliza la variable $this, se está haciendo referencia al objeto sobre el que se ha llamado al método, en este caso, el objeto $mi_objeto. Con $this->titulo estamos haciendo referencia al atributo "titulo" que tiene el objeto $mi_objeto.

Si queremos probar la clase soporte, para confirmar que se ejecuta correctamente y que ofrece resultados coherentes, podemos utilizar un código como el siguiente.

```
$soporte1 = new soporte("Los Intocables",22,3);
echo "<b>" . $soporte1->titulo . "</b>";
echo "<br>Precio: " . $soporte1->dame_precio_sin_iva() . " euros";
echo "<br>Precio IVA incluido: " . $soporte1->dame_precio_con_iva() . " euros";
```

En este caso hemos creado una instancia de la clase soporte, en un objeto que hemos llamado $soporte1. Luego imprimimos su atributo titulo (como el título ha sido definido como public, podemos acceder a él desde fuera del código de la clase.
Luego se llaman a los métodos dame_precio_sin_iva() y dame_precio_con_iva() para el objeto creado.

Nos daría como resultado esto:

Los Intocables
Precio: 3 euros
Precio IVA incluido: 3.48 euros

En siguientes capítulos vamos a ver cómo definir clases que hereden de la clase soporte.

La herencia en PHP 5, Segunda parte

Como estamos viendo, los mecanismos de herencia en PHP5 son similares a los existentes en otros lenguajes de programación. Ahora vamos a relatar cómo construir una clase que hereda de otra.

Continuando con nuestro ejemplo de videoclub, vamos a construir una clase para los soportes de tipo cinta de video. Las cintas de vídeo tienen un atributo nuevo que es la duración de la cinta. No tienen ninguna clase nueva, aunque debemos aprender a sobrescribir métodos creados para el soporte, dado que ahora tienen que hacer tareas más específicas.

Sobrescribir métodos

Antes de mostrar el código de la clase cinta_video, vamos a hablar sobre la sobrescritura o sustitución de métodos, que es un mecanismo por el cual una clase que hereda puede redefinir los métodos que está heredando.

Pensemos en una cafetera. Sabemos que existen muchos tipos de cafeteras y todas hacen café, pero el mecanismo para hacer el café es distinto dependiendo del tipo de cafetera. Existen cafeteras express, cafeteras por goteo y hasta se puede hacer café con un calcetín. Nuestra cafetera "padre" (de la que va a heredar todas las cafeteras) puede tener definido un método hacer_cafe(), pero no necesariamente todas las cafeteras que puedan heredar de esta hacen el café siguiendo el mismo proceso.

Entonces podemos definir un método para hacer café estándar, que tendría la clase cafetera. Pero al definir las clases cafetera_express y cafetera_goteo, deberíamos sobrescribir el método hacer_cafe() para que se ajuste al procedimiento propio de estas.

La sobrescritura de métodos es algo bastante común en mecanismos de herencia, puesto que

los métodos que fueron creados para una clase "padre" no tienen por qué ser los mismos que los definidos en las clases que heredan.

Veremos cómo sobrescribir o sustituir métodos en un ejemplo de herencia, siguiendo nuestro ejemplo de videoclub.

Sintaxis para heredar en PHP 5

Habíamos comentado que el videoclub dispone de distintos elementos para alquilar, como cintas de vídeo, DVD o juegos. Habíamos creado una clase soporte, que vamos a heredar en cada uno de los elementos disponibles para alquiler. Vamos a empezar por la clase cinta_video, cuyo código será el siguiente.

```
class cinta_video extends soporte{
  private $duracion;

  function __construct($tit,$num,$precio,$duracion){
    parent::__construct($tit,$num,$precio);
    $this->duracion = $duracion;
  }

  public function imprime_caracteristicas(){
    echo "Película en VHS:<br>";
    parent::imprime_caracteristicas();
    echo "<br>Duración: " . $this->duracion;
  }
}
```

Con la primera línea class cinta_video extends soporte estamos indicando que se está definiendo la clase cinta_video y que va a heredar de la clase soporte.

> **Nota:** Como se está heredando de una clase, PHP tiene que conocer el código de la clase "padre", en este caso la clase soporte. De modo que el código de la clase soporte debe estar incluido dentro del archivo de la clase cinta_video. Podemos colocar los dos códigos en el mismo fichero, o si están en ficheros independientes, debemos incluir el código de la clase soporte con la instrucción include o require de PHP.

En la clase cinta_video hemos definido un nuevo atributo llamado $duracion, que almacena el tiempo que dura la película.

Aunque la clase sobre la que heredamos (la clase soporte) tenía definido un constructor, la cinta de vídeo debe inicializar la nueva propiedad $duracion, que es específica de las cintas de video. Por ello, vamos a sobrescribir o sustituir el método constructor, lo que se hace simplemente volviendo a escribir el método. La gracia aquí consiste en que el sistema puede basar la nueva declaración del constructor en la declaración que existía para la clase de la que hereda.

Es decir, ya se había definido un constructor para la clase soporte, que inicializaba los atributos de esta clase. Ahora, para la clase cinta_video, hay que inicializar los atributos definidos en la clase soporte, más el atributo $duracion, que es propio de cinta_video.

El código del constructor es el siguiente:

```
function __construct($tit,$num,$precio,$duracion){
  parent::__construct($tit,$num,$precio);
  $this->duracion = $duracion;
}
```

En la primera línea del constructor se llama al constructor creado para la clase "soporte". Para ello utilizamos parent:: y luego el nombre del método de la clase padre al que se quiere llamar, en este caso __constructor(). Al constructor de la clase padre le enviamos las variables que se deben inicializar con la clase padre.

En la segunda línea del constructor se inicializa el atributo duracion, con lo que hayamos

recibido por parámetro.

Nos pasa lo mismo con el método imprime_caracteristicas(), que ahora debe mostrar también el nuevo atributo, propio de la clase cinta_video. Como se puede observar en el código de la función, se hace uso también de parent::imprime_caracteristicas() para utilizar el método definido en la clase padre.

Si queremos probar la clase cinta_video, podriamos utilizar un código como este:

```
$micinta = new cinta_video("Los Otros", 22, 4.5, "115 minutos");
echo "<b>" . $micinta->titulo . "</b>";
echo "<br>Precio: " . $micinta->dame_precio_sin_iva() . " euros";
echo "<br>Precio IVA incluido: " . $micinta->dame_precio_con_iva() . " euros";
```

Lo que nos devolvería lo siguiente:

Los Otros
Precio: 4.5 euros
Precio IVA incluido: 5.22 euros
Película en VHS:
Los Otros
4.5 (IVA no incluido)
Duración: 115 minutos

La herencia en PHP5, Tercera parte

La clase soporte tiene otras clases que heredan de ella y que todavía no hemos definido.

Veamos primero el código de la clase "dvd", que es muy parecido al visto para la clase cinta_video. Lo único que cambia es que ahora vamos a definir otros atributos relacionados con los DVD, como son los idiomas disponibles en el DVD y el formato de pantalla que tiene la grabación.

```
class dvd extends soporte{
  public $idiomas_disponibles;
  private $formato_pantalla;

  function __construct($tit,$num,$precio,$idiomas,$pantalla){
    parent::__construct($tit,$num,$precio);
    $this->idiomas_disponibles = $idiomas;
    $this->formato_pantalla = $pantalla;
}

  public function imprime_caracteristicas(){
    echo "Película en DVD:<br>";
    parent::imprime_caracteristicas();
    echo "<br>" . $this->idiomas_disponibles;
  }
}
```

> **Nota:**Para una explicación detallada de este código os referimos al capítulo anterior, donde se explicaba la clase cinta_video y la sobrescritura de métodos.

Por su parte, la clase juego, tendrá 3 nuevos atributos. Estos son "consola", para especificar la consola para la que está creado este juego, "min_num_jugadores", para especificar el número de jugadores mínimo y "max_num_jugadores", para especificar el máximo número de jugadores que pueden participar en el juego.

Este será el código de la clase juego.

```
class juego extends soporte{
  public $consola; //nombre de la consola del juego ej: playstation
  private $min_num_jugadores;
```

```
  private $max_num_jugadores;

  function __construct($tit,$num,$precio,$consola,$min_j,$max_j){
    parent::__construct($tit,$num,$precio);
    $this->consola = $consola;
    $this->min_num_jugadores = $min_j;
    $this->max_num_jugadores = $max_j;
  }

  public function imprime_jugadores_posibles(){
    if ($this->min_num_jugadores == $this->max_num_jugadores){
      if ($this->min_num_jugadores==1)
      echo "<br>Para un jugador";
      else
      echo "<br>Para " . $this->min_num_jugadores . " jugadores";
    }else{
     echo "<br>De " . $this->min_num_jugadores . " a " . $this->max_num_jugadores . " Jugadores.";
    }
  }

  public function imprime_caracteristicas(){
    echo "Juego para: " . $this->consola . "<br>";
    parent::imprime_caracteristicas();
    echo "<br>" . $this->imprime_jugadores_posibles();
  }
}
```

Nos fijamos en el constructor, que llama al constructor de la clase padre para inicializar algunos atributos propios de los soportes en general.

Luego nos fijamos en el método imprime_jugadores_posibles(), que muestra los jugadores permitidos. Ha sido declarada como public, para que se pueda acceder a ella desde cualquier lugar. Nos da un mensaje como "Para un jugador" o "De 1 a 2 Jugadores", dependiendo de los valores min_num_jugadores y max_num_jugadores.

Por su parte, se sobrescribe la función imprime_caracteristicas(), para mostrar todos los datos de cada juego. Primero se muestra la consola para la que se ha creado el juego. Los datos generales (propios de la clase "soporte") se muestran llamando al mismo método de la clase "parent" y el número de jugadores disponibles se muestra con una llamada al método imprime_jugadores_posibles().

Podríamos utilizar un código como el que sigue, si es que queremos comprobar que la clase funciona correctamente y que nos ofrece la salida que estábamos pensando.

```
$mijuego = new juego("Final Fantasy", 21, 2.5, "Playstation",1,1);
$mijuego->imprime_caracteristicas();

//esta línea daría un error porque no se permite acceder a un atributo private del objeto
//echo "<br>Jugadores: " . $mijuego->min_num_jugadores;
//habria que crear un método para que acceda a los atributos private
$mijuego->imprime_jugadores_posibles();

echo "<p>";
$mijuego2 = new juego("GP Motoracer", 27, 3, "Playstation II",1,2);
echo "<b>" . $mijuego2->titulo . "</b>";
$mijuego2->imprime_jugadores_posibles();
```

Este código que utiliza la clase "juego" dará como salida:

Juego para: Playstation
Final Fantasy
2.5 (IVA no incluido)
Para un jugador

Para un jugador
GP Motoracer
De 1 a 2 Jugadores.

Los atributos de los objetos pueden ser otros objetos

Las características de los objetos, que se almacenan por medio de los llamados atributos o propiedades, pueden ser de diversa naturaleza. La clase hombre puede tener distintos tipos de atributos, como la edad (numérico), el nombre propio (tipo cadena de caracteres), color de piel (que puede ser un tipo cadena de caracteres o tipo enumerado, que es una especie de variable que sólo puede tomar unos pocos valores posibles). También puede tener una estatura o un peso (que podrían ser de tipo float o número en coma flotante).

En general, podemos utilizar cualquier tipo para los atributos de los objetos, incluso podemos utilizar otros objetos. Por ejemplo, podríamos definir como atributo de la clase hombre sus manos. Dada la complejidad de las manos, estas podrían definirse como otro objeto. Por ejemplo, las manos tendrían como características la longitud de los dedos, un coeficiente de elasticidad. Como funcionalidades o métodos, podríamos definir agarrar algo, soltarlo, pegar una bofetada, o cortarse las uñas. Así pues, uno de los atributos de la clase hombre podría ser un nuevo objeto, con su propias características y funcionalidades. La complejidad de las manos no le importa al desarrollador de la clase hombre, por el principio de encapsulación, dado que este conoce sus propiedades (o aquellas declaradas como public) y los métodos (también los que se hayan decidido declarar como públicos) y no necesita preocuparse sobre cómo se han codificado.

Clase cliente del videoclub

Para continuar el ejemplo del videoclub, hemos creado la clase cliente que vamos a explicar a continuación. En los clientes hemos definido como atributo, entre otros, las películas o juegos alquilados.

> **Nota:**Como vemos, los objetos pueden tener algunos atributos también de tipo objeto. En ese caso pueden haber distintos tipos de relaciones entre objetos. Por ejemplo, por pertenencia, como en el caso de la clase hombre y sus manos, pues a un hombre le pertenecen sus manos. También se pueden relacionar los objetos por asociación, como es el caso de los clientes y las películas que alquilan, pues en ese caso un cliente no tiene una película propiamente dicha, sino que se asocia con una película momentáneamente.

La clase cliente que hemos creado tiene cierta complejidad, esperamos que no sea demasiada para que se pueda entender fácilmente. La explicaremos poco a poco para facilitar las cosas.

Atributos de la clase cliente

Hemos definido una serie de atributos para trabajar con los clientes. Tenemos los siguientes:

```
public $nombre;
private $numero;
private $soportes_alquilados;
private $num_soportes_alquilados;
private $max_alquiler_concurrente;
```

Como se puede ver, se han definido casi todos los atributos como private, con lo que sólo se podrán acceder dentro del código de la clase.

El atributo nombre, que guarda el nombre propio del cliente, es el único que hemos dejado como público y que por tanto se podrá referenciar desde cualquier parte del programa, incluso desde otras clases. El atributo numero se utiliza para guardar el identificador numérico del cliente. Por su parte, soportes alquilados nos servirá para asociar al cliente las películas o juegos cuando este las alquile. El atributo num_soportes_alquilados almacenará el número de películas o juegos que un cliente tiene alquilados en todo momento. Por último, max_alquiler_concurrente indica el número máximo de soportes que puede tener alquilados un cliente en un mismo instante, no permitiéndose alquilar a ese cliente, a la vez, más que ese número de películas o juegos.

El único atributo sobre el que merece la pena llamar la atención es soportes_alquilados, que contendrá un array de soportes. En cada casilla del array se introducirán las películas o juegos que un cliente vaya alquilando, para asociar esos soportes al cliente que las alquiló. El array contendrá tantas casillas como el max_alquiler_concurrente, puesto que no tiene sentido asignar mayor espacio al array del que se va a utilizar. Para facilitar las cosas, cuando un cliente no tiene alquilado nada, tendrá el valor null en las casillas del array.

> **Nota:**Recordemos que los soportes fueron definidos en capítulos anteriores de este manual de PHP 5 [http://www.desarrolloweb.com/manuales/58/], mediante un mecanismo de herencia. Soporte era una clase de la que heredaban las películas en DVD, las cintas de vídeo y los juegos.

Constructor de la clase cliente

```
function __construct($nombre,$numero,$max_alquiler_concurrente=3){
  $this->nombre=$nombre;
  $this->numero=$numero;
  $this->soportes_alquilados=array();
  $this->num_soportes_alquilados=0;
  $this->max_alquiler_concurrente = $max_alquiler_concurrente;
  //inicializo las casillas del array de alquiler a "null"
  //un valor "null" quiere decir que el no hay alquiler en esa casilla
  for ($i=0;$i<$max_alquiler_concurrente;$i++){
    $this->soportes_alquilados[$i]=null;
  }
}
```

El constructor de la clase cliente recibe los datos para inicializar el objeto. Estos son $nombre, $numero y $max_alquiler_concurrente. Si nos fijamos, se ha definido por defecto a 3 el número máximo de alquileres que puede tener un cliente, de modo que, en el caso de que la llamada al constructor no envíe el parámetro $max_alquiler_concurrente se asumirá el valor 3.

El atributo soportes_alquilados, como habíamos adelantado, será un array que tendrá tantas casillas como el máximo de alquileres concurrentes. En las últimas líneas se inicializan a null el contenido de las casillas del array.

Los atributos de los objetos pueden ser otros objetos, 2º parte

Método dame_numero()

```
function dame_numero(){
  return $this->numero;
}
```

Este método simplemente devuelve el numero de identificación del cliente. Como se ha definido el atributo numero como private, desde fuera del código de la clase, sólo se podrá acceder a este a través del método dame_numero().

Método tiene_alquilado($soporte)

```
function tiene_alquilado($soporte){
 for ($i=0;$i<$this->max_alquiler_concurrente;$i++){
  if (!is_null($this->soportes_alquilados[$i])){
   if ($this->soportes_alquilados[$i]->dame_numero_identificacion() == $soporte->dame_numero_identificacion()){
    return true;
   }
  }
 }
 //si estoy aqui es que no tiene alquilado ese soporte
return false;
}
```

Este recibe un soporte y devuelve true si está entre los alquileres del cliente. Devuelve false en caso contrario.

A este método lo llamamos desde dentro de la clase cliente, en la codificación de otro método. Podríamos haber definido entonces el método como private, si pensásemos que sólo lo vamos a utilizar desde dentro de esta clase. En este caso lo hemos definido como public (el modificador por defecto) porque pensamos que puede ser útil para otras partes del programa.

Este método recibe un soporte (cualquier tipo de soporte, tanto cintas de vídeo, como DVDs o juegos). Realiza un recorrido por el array de soportes alquilados preguntando a cada soporte que tenga alquilado el cliente si su número de identificación es igual que el del soporte recibido por parámetro. Como el número de identificación del soporte está definido como private en la clase soporte, para acceder a su valor tenemos que utilizar el método de soporte dame_numero_identificación().

Otra cosa importante. Como no es seguro que en el array de soportes alquilados haya algún soporte (recordamos que si el cliente no tiene nada alquilado las casillas están a null), antes de llamar a ningún método del soporte debemos comprobar que realmente existe. Esto se hace con la función is_null() a la que le enviamos la casilla del array donde queremos comprobar si existe un soporte. Si la función is_null() devuelve true es que tiene almacenado el valor null, con lo que sabremos que no hay soporte. De manera contraria, si la casilla almacena un soporte, la función is_null() devolverá false.

Método alquila($soporte)

```
function alquila($soporte){
  if ($this->tiene_alquilado($soporte)){
   echo "<p>El cliente ya tiene alquilado el soporte <b>" . $soporte->titulo . "</b>";
  }elseif ($this->num_soportes_alquilados==$this->max_alquiler_concurrente){
   echo "<p>Este cliente tiene " . $this->num_soportes_alquilados . " elementos alquilados. ";
   echo "No puede alquilar más en este videoclub hasta que no devuelva algo";
  }else{
   //miro en el array a ver donde tengo sitio para meter el soporte
   $cont = 0;
   while (!is_null($this->soportes_alquilados[$cont])){
    $cont++;
   }
   $this->soportes_alquilados[$cont]=$soporte;
   $this->num_soportes_alquilados++;
   echo "<p><b>Alquilado soporte a: </b>" . $this->nombre . "<br>";
   $soporte->imprime_caracteristicas();
  }
}
```

Este método sirve para alquilar una película o juego por parte del cliente. Recibe el soporte a alquilar y, en caso que el alquiler se pueda producir, debe introducir el objeto soporte recibido en el array de soportes alquilados del cliente.

Lo primero que hace es comprobar que el cliente no tiene alquilado ese mismo soporte, utilizando el método tiene_alquilado() de la clase soporte. Si no lo tiene alquilado comprueba que todavía tiene capacidad para alquilar otro soporte, es decir, que no ha llegado al máximo en el número de soportes alquilados.

Si se puede alquilar el soporte, lo introduce dentro del array soportes_alquilados en una posición vacía (una casilla donde antes hubiera un null), incrementa en uno el número de soportes alquilados e imprime las características del soporte que se ha alquilado.

Método devuelve($identificador_soporte)

```
function devuelve($identificador_soporte){
  if ($this->num_soportes_alquilados==0){
   echo "<p>Este cliente no tiene alquilado ningún elemento";
   return false;
  }
  //recorro el array a ver si encuentro el soporte con identificador recibido
  for ($i=0;$i<$this->max_alquiler_concurrente;$i++){
   if (!is_null($this->soportes_alquilados[$i])){
    if ($this->soportes_alquilados[$i]->dame_numero_identificacion() == $identificador_soporte){
```

```
    echo "<p>Soporte devuelto: " . $identificador_soporte;
    echo " <b>" . $this->soportes_alquilados[$i]->titulo . "</b>";
     $this->soportes_alquilados[$i]=null;
     $this->num_soportes_alquilados--;
     return true;
    }
   }
  }
  //si estoy aqui es que el cliente no tiene ese soporte alquilado
  echo "<p>No se ha podido encontrar el soporte en los alquileres de este cliente";
  return false;
}
```

El método devuelve recibe el identificador del soporte que debe devolver el cliente. Devuelve true si se consiguió devolver el soporte, false en caso contrario. Lo primero que hace es comprobar si el cliente tiene alquilado algún soporte, comprobando que la variable num_soportes_alquilados no sea cero.

Luego hace un recorrido por el array de soportes alquilados para ver si encuentra el soporte que se desea devolver. Para cada soporte alquilado (cada casilla del array que no contenga el valor null) comprueba si el identificador es el mismo que el recibido por parámetro. Cuando encuentra el soporte, muestra un mensaje por pantalla y lo devuelve simplemente poniendo a null la casilla correspondiente del array soportes_alquilados y decrementando en uno el atributo num_soportes_alquilados.

Si se encuentra el soporte, se sale de la función devolviendo true. Por lo que, si no se ha salido de la función después de hacer el recorrido por el array, sabemos que no se ha encontrado ese soporte. Entonces mostramos un mensaje en pantalla y devolvemos false.

Método lista_alquileres()

```
function lista_alquileres(){
  if ($this->num_soportes_alquilados==0){
   echo "<p>Este cliente no tiene alquilado ningún elemento";
  }else{
   echo "<p><b>El cliente tiene " . $this->num_soportes_alquilados . " soportes alquilados</b>";
   //recorro el array para listar los elementos que tiene alquilados
   for ($i=0;$i<$this->max_alquiler_concurrente;$i++){
    if (!is_null($this->soportes_alquilados[$i])){
     echo "<p>";
     $this->soportes_alquilados[$i]->imprime_caracteristicas();
    }
   }
  }
}
```

Este método hace un recorrido por el array de soportes alquilados y muestra las características de cada soporte. Comprueba que el cliente tiene algo alquilado antes de hacer el recorrido.

Recordar siempre, antes de llamar a un método del soporte almacenado en cada casilla del array, comprobar que el contenido de esa casilla no es null.

Método imprime_caracteristicas()

```
function imprime_caracteristicas(){
  echo "<p><b>Cliente " . $this->numero . ":</b> " . $this->nombre;
  echo "<br>Alquileres actuales: " . $this->num_soportes_alquilados
}
```

Simplemente muestra algunos de los datos del cliente. Es un método para obtener algún dato por pantalla de un cliente.

Comprobar el funcionamiento de la clase cliente

Es importante señalar que, para que la clase cliente funcione, debe disponer de los códigos de

las distintas clases de las que hace uso (la clase soporte, cinta_video, juego y dvd). Estos códigos se pueden haber escrito en el mismo archivo o bien incluirse con unas instrucciones como estas:

```
include "soporte.php";
include "dvd.php";
include "juego.php";
include "cinta_video.php";
```

La clase cliente se puede poner en funcionamiento, para probar su correcta implementación, con un código como este:

```
//instanciamos un par de objetos cliente
$cliente1 = new cliente("Pepe", 1);
$cliente2 = new cliente("Roberto", 564);

//mostramos el numero de cada cliente creado
echo "El identificador del cliente 1 es: " . $cliente1->dame_numero();
echo "<br>El identificador del cliente 2 es: " . $cliente2->dame_numero();

//instancio algunos soportes
$soporte1 = new cinta_video("Los Otros", 1, 3.5, "115 minutos");
$soporte2 = new juego("Final Fantasy", 2, 2.5, "Playstation",1,1);
$soporte3 = new dvd("Los Intocables", 3, 3, "Inglés y español","16:9");
$soporte4 = new dvd("El Imperio Contraataca", 4, 3, "Inglés y español","16:9");

//alquilo algunos soportes
$cliente1->alquila($soporte1);
$cliente1->alquila($soporte2);
$cliente1->alquila($soporte3);

//voy a intentar alquilar de nuevo un soporte que ya tiene alquilado
$cliente1->alquila($soporte1);

//el cliente tiene 3 soportes en alquiler como máximo
//este soporte no lo va a poder alquilar
$cliente1->alquila($soporte4);

//este soporte no lo tiene alquilado
$cliente1->devuelve(4);
//devuelvo un soporte que sí que tiene alquilado
$cliente1->devuelve(2);
//alquilo otro soporte
$cliente1->alquila($soporte4);

//listo los elementos alquilados
$cliente1->lista_alquileres();
```

La ejecución de este código, si todo funciona correctamente, debería devolvernos como resultado esta salida:

El identificador del cliente 1 es: 1
El identificador del cliente 2 es: 564

Alquilado soporte a: Pepe
Película en VHS:
Los Otros
3.5 (IVA no incluido)
Duración: 115 minutos

Alquilado soporte a: Pepe
Juego para: Playstation
Final Fantasy
2.5 (IVA no incluido)
Para un jugador

Alquilado soporte a: Pepe

Película en DVD:
Los Intocables
3 (IVA no incluido)
Inglés y español
El cliente ya tiene alquilado el soporte **Los Otros**

Este cliente tiene 3 elementos alquilados. No puede alquilar más en este videoclub hasta que no devuelva algo

No se ha podido encontrar el soporte en los alquileres de este cliente

Soporte devuelto: **2 Final Fantasy**

Alquilado soporte a: Pepe
Película en DVD:
El Imperio Contraataca
3 (IVA no incluido)
Inglés y español

El cliente tiene 3 soportes alquilados

Película en VHS:
Los Otros
3.5 (IVA no incluido)
Duración: 115 minutos

Película en DVD:
El Imperio Contraataca
3 (IVA no incluido)
Inglés y español

Película en DVD:
Los Intocables
3 (IVA no incluido)
Inglés y español

Repasando la creación de clases

Para continuar la creación del videoclub y las explicaciones sobre la programación orientada a objetos (POO), vamos a programar la clase principal, que engloba a todas las clases que hemos ido creando hasta el momento. La clase principal se llama videoclub y modela el comportamiento general del videoclub.

Llegado este punto sería bueno que remarcar dos cosas sobre el desarrollo de programas orientados a objetos.

1. La clase principal de un sistema que deseamos modelar en POO se suele llamar como el propio sistema que estamos modelando. Por ejemplo, si estuviéramos creando una biblioteca, la clase principal se llamaría biblioteca. En este caso, que estamos haciendo un videoclub, la clase principal se llamará videoclub.
2. El proceso de creación de un programa POO se realiza al revés de como hemos hecho en este manual, empezando el desarrollo de la clase general y finalizando por las clases más específicas. De este modo, al crear la clase general, podemos ir viendo qué otros objetos necesitaremos, cuáles serán sus métodos y propiedades. En este manual lo hemos hecho al revés porque nos venía bien para ir describiendo las características de la POO.

La clase videoclub tendrá como propiedades a los soportes en alquiler (películas o juegos) y por otra parte, los socios o clientes que alquilan los productos. Los métodos de la clase videoclub

serán la inclusión y listado de soportes en alquiler y de socios, el alquiler de soportes por parte de clientes.

> **Nota:**Ni que decir tiene que el videoclub que estamos creando está simplificado al máximo. Está claro que si estuviésemos creando un videoclub con el propósito de utilizarlo en producción, habría que pensar y desarrollar muchas otras funcionalidades.

Vamos ya con el código y sus explicaciones.

Atributos de la clase videoclub

```
public $nombre;
private $productos;
private $num_productos;
private $socios;
private $num_socios;
```

El atributo $productos será un array con los distintos soportes en alquiler. $num_productos lo utilizaremos para llevar la cuenta del número de productos que tenemos disponibles. De modo similar, $socios será un array de clientes y $num_socios llevará la cuenta de los socios que tenemos dados de alta. Aparte, nuestro videoclub tendrá un nombre, que almacenaremos en la variable $nombre.

Constructor

```
function __construct($nombre){
  $this->nombre=$nombre;
  $this->productos=array();
  $this->num_productos=0;
  $this->socios=array();
  $this->num_socios=0;
}
```

Este método inicializará los atributos del objeto que se está construyendo. Recibe únicamente el nombre del videoclub. Como tareas destacables están las inicializaciones de los arrays de productos y socios y la puesta a cero de los contadores que vamos a utilizar.

Método incluir_producto()

```
private function incluir_producto($nuevo_producto){
  $this->productos[$this->num_productos]=$nuevo_producto;
  echo "<p>Incluido soporte " . $this->num_productos;
  $this->num_productos++;
}
```

Este método ha sido declarado como private, porque sólo queremos que se llame desde dentro de la clase. Recibe el nuevo producto que se quiere dar de alta y lo guardar en el array de productos, en la posición marcada por el atributo num_productos. Luego muestra un mensaje por pantalla y por último incrementa a uno el atributo num_productos.

Métodos incluir_dvd(),incluir_cinta_video() e incluir_juego()

Los tres siguientes métodos que vamos a ver, instancian los tres productos con los que trabaja el videoclub y luego los introducen en array de productos llamando a incluir_producto().

```
function incluir_dvd($tit,$precio,$idiomas,$pantalla){
  $dvd_nuevo = new dvd($tit, $this->num_productos, $precio, $idiomas, $pantalla);
  $this->incluir_producto($dvd_nuevo);
}

function incluir_cinta_video($tit,$precio,$duracion){
  $cinta_video_nueva = new cinta_video($tit, $this->num_productos, $precio, $duracion);
  $this->incluir_producto($cinta_video_nueva);
}
```

```
function incluir_juego($tit,$precio,$consola,$min_j,$max_j){
  $juego_nuevo = new juego($tit, $this->num_productos, $precio, $consola, $min_j, $max_j);
  $this->incluir_producto($juego_nuevo);
}
```

Podemos fijarnos que el número de identificación del soporte, que recibe el constructor de las cintas de vídeo, DVDs o juegos, lo generamos por medio del atributo de la clase de videclub num_productos, que guarda el número de productos dados de alta.

Método incluir_socio()

Este método hace las tareas de instanciación del socio nuevo y su inclusión en el array de socios.

```
function incluir_socio($nombre,$max_alquiler_concurrente=3){
  $socio_nuevo = new cliente($nombre,$this->num_socios,$max_alquiler_concurrente);
  $this->socios[$this->num_socios]=$socio_nuevo;
  echo "<p>Incluido socio " . $this->num_socios;
  $this->num_socios++;
}
```

Reciben los datos del nuevo socio: nombre y el máximo número de películas que puede alquilar (siendo 3 el valor por defecto). Una vez instanciado el nuevo socio, lo introduce en el array, en la posición marcada por el atributo num_socios. Luego muestran un mensaje por pantalla y por último incrementan a uno los atributos num_productos o num_socios.

El número de socio, que recibe entre otros parámetros, el constructor de la clase socio lo generamos por medio del contador de socios num_socios, de la clase videclub.

Métodos listar_productos() y listar_socios()

Dos métodos muy similares, que veremos de una sola vez.

```
function listar_productos(){
  echo "<p>Listado de los " . $this->num_productos . " productos disponibles:";
  for ($i=0;$i<$this->num_productos;$i++){
    echo "<p>";
    $this->productos[$i]->imprime_caracteristicas();
  }
}

function listar_socios(){
  echo "<p>Listado de $this->num_socios socios del videoclub:";
  for ($i=0;$i<$this->num_socios;$i++){
    echo "<p>";
    $this->socios[$i]->imprime_caracteristicas();
  }
}
```

Estos métodos imprimen un listado completo de los socios y productos dados de alta. Simplemente hacen un recorrido del array de productos o de socios y van imprimiendo sus características.

Método alquila_a_socio()

Realiza las acciones necesarias para alquiler un producto a un socio.

```
function alquila_a_socio($numero_socio,$numero_producto){
  if (is_null($this->socios[$numero_socio])){
    echo "<p>No existe ese socio";
  }elseif(is_null($this->productos[$numero_producto])){
    echo "<p>No existe ese soporte";
  }else{
    $this->socios[$numero_socio]->alquila($this->productos[$numero_producto]);
  }
}
```

Este método recibe el identificador del socio y del producto en alquiler. Antes de proceder, realiza un par de comprobaciones. La primera para ver si existe un socio con el número de socio indicado por parámetro y la segunda para ver si también existe un producto con el número de producto dado.

Si todo fue bien, llama al método alquila() del socio, enviándole el producto que desea alquilar.

> **Nota:**El método alquila() del socio tiene cierta complejidad, pero ya la vimos cuando explicamos la clase socio. En este momento, por el principio de encapsulación de la POO, debemos abstraernos de su dificultad y no prestarle atención porque sabemos que funciona y no nos debe preocupar cómo lo hace.

Para probar la clase videoclub podríamos utilizar un código como este:

```
$vc = new videoclub("La Eliana Video");

//voy a incluir unos cuantos soportes de prueba
$vc->incluir_juego("Final Fantasy", 2.5, "Playstation",1,1);
$vc->incluir_juego("GP Motoracer", 3, "Playstation II",1,2);
$vc->incluir_dvd("Los Otros", 4.5, "Inglés y español","16:9");
$vc->incluir_dvd("Ciudad de Diós", 3, "Portugués, inglés y español","16:9");
$vc->incluir_dvd("Los Picapiedra", 3, "Español","16:9");
$vc->incluir_cinta_video("Los Otros", 4.5, "115 minutos");
$vc->incluir_cinta_video("El nombre de la Rosa", 1.5, "140 minutos");

//listo los productos
$vc->listar_productos();

//voy a crear algunos socios
$vc->incluir_socio("José Fuentes");
$vc->incluir_socio("Pedro García",2);

$vc->alquila_a_socio(1,2);
$vc->alquila_a_socio(1,3);
//alquilo otra vez el soporte 2 al socio 1.
// no debe dejarme porque ya lo tiene alquilado
$vc->alquila_a_socio(1,2);
//alquilo el soporte 6 al socio 1.
//no se puede porque el socio 1 tiene 2 alquileres como máximo
$vc->alquila_a_socio(1,6);

//listo los socios
$vc->listar_socios();
```

Se hace una carga de datos y una llamada a todos los métodos que hemos visto para el videoclub. Este código dará como resultado una salida como la siguiente:

Incluido soporte 0

Incluido soporte 1

Incluido soporte 2

Incluido soporte 3

Incluido soporte 4

Incluido soporte 5

Incluido soporte 6

Listado de los 7 productos disponibles:

Juego para: Playstation
Final Fantasy
2.5 (IVA no incluido)
Para un jugador

Juego para: Playstation II
GP Motoracer
3 (IVA no incluido)
De 1 a 2 Jugadores.

Película en DVD:
Los Otros
4.5 (IVA no incluido)
Inglés y español

Película en DVD:
Ciudad de Diós
3 (IVA no incluido)
Portugués, inglés y español

Película en DVD:
Los Picapiedra
3 (IVA no incluido)
Español

Película en VHS:
Los Otros
4.5 (IVA no incluido)
Duración: 115 minutos

Película en VHS:
El nombre de la Rosa
1.5 (IVA no incluido)
Duración: 140 minutos

Incluido socio 0

Incluido socio 1

Alquilado soporte a: Pedro García
Película en DVD:
Los Otros
4.5 (IVA no incluido)
Inglés y español

Alquilado soporte a: Pedro García
Película en DVD:
Ciudad de Diós
3 (IVA no incluido)
Portugués, inglés y español

El cliente ya tiene alquilado el soporte **Los Otros**

Este cliente tiene 2 elementos alquilados. No puede alquilar más en este videoclub hasta que no devuelva algo

Listado de 2 socios del videoclub:

Cliente 0: José Fuentes
Alquileres actuales: 0

Cliente 1: Pedro García
Alquileres actuales: 2

Hasta aquí ha llegado por ahora el desarrollo de este videoclub, que no es muy funcional pero esperamos que haya servido para empezar a conocer las características de la programación orientada a objetos.

En adelante, seguiremos este manual comentando otras particularidades de la POO en PHP 5, que también hay que conocer.

Métodos y clases abstractos en PHP 5

Una clase abstracta es la que tiene métodos abstractos. Los métodos abstractos son los que están declarados en una clase, pero no se ha definido en la clase el código de esos métodos.

Esa puede ser una buena definición de clases y métodos abstractos, pero veamos con calma una explicación un poco más detallada y comprensible por todos.

En ocasiones, en un sistema de herencia como el de la programación orientada a objetos (POO), tenemos entidades que declarar aunque no se puede dar su definición todavía, simplemente las deseamos definir por encima para empezar una jerarquía de clases.

Pensemos en los productos lácteos (los derivados de la leche). No cabe duda que los productos lácteos son una gran familia. Incluyen a los yogures, mantequillas, quesos, helados e incluso a la propia leche. Sin embargo, los productos lácteos en si no se encuentran en la vida real. En el supermercado no te venden un producto lácteo en general. Por ejemplo, nadie compra un kilo de producto lácteo... más bien preguntarán por un litro de leche, un litro de helado o un pack de yogures.

Todos los productos lácteos tienen algunas características comunes, como el porcentaje en leche o la fecha de caducidad. También tienen algunas funcionalidades comunes como conservarse o consumirse. Sin embargo, la manera de conservarse es distinta dependiendo del producto lácteo. La leche se conserva fuera de la nevera, mientras que no esté abierto el brick, y los yogures deben conservarse en la nevera en todo momento. Los quesos se conservan en la nevera, pero metidos dentro de un recipiente por si acaso desprenden olores fuertes. Por lo que respecta a los helados, se deben conservar en el congelador, siempre que deseemos que no se conviertan en líquido. Al consumir un producto lácteo la cosa también cambia, puesto que el queso se suele acompañar con pan o tostadas, la leche se bebe y el helado se toma con cuchara.

En definitiva, a donde queremos demostrar es que podemos tener un conjunto de objetos que tienen unas características comunes y funcionalidades, también comunes, pero que difieren en la manera de llevarlas a cabo. Para esto está la abstracción.

La clase de los productos lácteos, tendrá una serie de propiedades y unos métodos abstractos. Los métodos abstractos, como habíamos adelantado, son aquellos que no incluyen una codificación, sino que simplemente se declaran, dejando para las clases que hereden la tarea de codificarlos.

En este caso, la clase producto lácteo tendrá los métodos abstractos conservarse() y consumirse(), pero no se especificará el código fuente de estos métodos (por eso son abstractos). Las clases que hereden de producto lácteo serán las encargadas de definir un código para los métodos definidos como abstractos en la clase padre. Así, cada clase que herede de producto lácteo, deberá especificar el mecanismo concreto y específico por el cual se van a conservar o consumir.

Las clases que incorporan métodos abstractos se deben declarar como abstractas. Es una condición forzosa. Las clases abstractas no se pueden instanciar. Es decir, no podemos crear objetos a partir de ellas. Es algo lógico. Pensemos en los productos lácteos, estos no existen más que como una idea general. Sólo podremos encontrar productos lácteos de un tipo en

concreto, como leche o yogur, pero no la idea de producto lácteo en general.

Una clase que herede de un producto lácteo debe definir los métodos abstractos declarados en la clase abstracta. De lo contrario, la clase que hereda estaría obligada a declararse como abstracta.

En nuestro ejemplo de videoclub, tratado a lo largo de los distintos capítulos del manual de PHP 5, tenemos una clase que también sería un buen ejemplo de clase abstracta. Se trata de la clase soporte. De esta clase heredaban los distintos productos del videoclub, como películas en DVD, cintas de vídeo o juegos. No hubiera sido mala idea declarar como abstracta la clase soporte, dado que no se van a utilizar, ni existen, soportes en general, sino que lo que existen son los distintos soportes concretos.

La sintaxis de la abstracción

Para declarar clases y métodos abstractos se utiliza la siguiente sintaxis.

```
abstract class nombre_clase{

  //propiedades
public x;
  private y;

  //métodos

  public function __construct(){
   …
  }

  public abstract function nombre_metodo();

}
```

Nos fijamos que se utiliza la palabra clave "abstract" para definir las clases o métodos abstractos. Además, los métodos abstractos no llevan ningún código asociado, ni siquiera las llaves para abrir y cerrar el método.

Interfaces en PHP 5

Las interfaces son un sistema bastante común, utilizado en programación orientada a objetos. Son algo así como declaraciones de funcionalidades que tienen que cubrir las clases que implementan las interfaces.

En una interfaz se definen habitualmente un juego de funciones que deben codificar las clases que implementan dicha interfaz. De modo que, cuando una clase implementa una interfaz, podremos estar seguros que en su código están definidas las funciones que incluía esa interfaz.

A la hora de programar un sistema, podemos contar con objetos que son muy diferentes y que por tanto no pertenecen a la misma jerarquía de herencia, pero que deben realizar algunas acciones comunes. Por ejemplo, todos los objetos con los que comercia unos grandes almacenes deben contar con la funcionalidad de venderse. Una mesa tiene poco en común con un calefactor o unas zapatillas, pero todos los productos disponibles deben implementar una función para poder venderse.

Otro ejemplo. Una bombilla, un coche y un ordenador son clases muy distintas que no pertenecen al mismo sistema de herencia, pero todas pueden encenderse y apagarse. En este caso, podríamos construir una interfaz llamada "encendible", que incluiría las funcionalidades de encender y apagar. En este caso, la interfaz contendría dos funciones o métodos, uno encender () y otro apagar().

Cuando se define una interfaz, se declaran una serie de métodos o funciones sin especificar ningún código fuente asociado. Luego, las clases que implementen esa interfaz serán las encargadas de proporcionar un código a los métodos que contiene esa interfaz. Esto es seguro: si una clase implementa una interfaz, debería declarar todos los métodos de la interfaz. Si no tenemos código fuente para alguno de esos métodos, por lo menos debemos declararlos como abstractos y, por tanto, la clase también tendrá que declararse como abstracta, porque tiene métodos abstractos.

Código para definir una interfaz

Veamos el código para realizar una interfaz. En concreto veremos el código de la interfaz encendible, que tienen que implementar todas las clases cuyos objetos se puedan encender y apagar.

```
interface encendible{
   public function encender();
   public function apagar();
}
```

Vemos que para definir una interfaz se utiliza la palabra clave interface, seguida por el nombre de la interfaz y, entre llaves, el listado de métodos que tendrá. Los métodos no se deben codificar, sino únicamente declararse.

Implementación de interfaces

Ahora veamos el código para implementar una interfaz en una clase.

```
class bombilla implements encendible{
   public function encender(){
      echo "<br>Y la luz se hizo...";
   }

   public function apagar(){
      echo "<br>Estamos a oscuras...";
   }
}
```

Para implementar una interfaz, en la declaración de la clase, se debe utilizar la palabra implements, seguida del nombre de la interfaz que se va a implementar. Se podrían implementar varias interfaces en la misma clase, en cuyo caso se indicarían todos los nombres de las interfaces separadas por comas.

En el código de la clase estamos obligados a declarar y codificar todos los métodos de la interfaz.

> **Nota:** en concreto, PHP 5 entiende que si una clase implementa una interfaz, los métodos de esa interfaz estarán siempre en la clase, aunque no se declaren. De modo que si no los declaramos explícitamente, PHP 5 lo hará por nosotros. Esos métodos de la interfaz serán abstractos, así que la clase tendrá que definirse como abstracta. Se puede encontrar más información sobre la abstracción en el artículo Métodos y clases abstractos en PHP 5 [http://www.desarrolloweb.com/articulos/2103.php].

Ahora veamos el código de la clase coche, que también implementa la interfaz encendible. Este código lo hemos complicado un poco más.

```
class coche implements encendible{
   private $gasolina;
   private $bateria;
   private $estado = "apagado";

   function __construct(){
      $this->gasolina = 0;
      $this->bateria = 10;
```

```
  }

  public function encender(){
    if ($this->estado == "apagado"){
      if ($this->bateria > 0){
        if ($this->gasolina > 0){
          $this->estado = "encendido";
          $this->bateria --;
          echo "<br><b>Enciendo...</b> estoy encendido!";
        }else{
          echo "<br>No tengo gasolina";
        }
      }else{
        echo "<br>No tengo batería";
      }
    }else{
      echo "<br>Ya estaba encendido";
    }
  }

  public function apagar(){
    if ($this->estado == "encendido"){
      $this->estado = "apagado";
      echo "<br><b>Apago...</b> estoy apagado!";
    }else{
      echo "<br>Ya estaba apagado";
    }
  }

  public function cargar_gasolina($litros){
    $this->gasolina += $litros;
    echo "<br>Cargados $litros litros";
  }
}
```

A la vista del anterior código, se puede comprobar que no hay mucho en común entre las clases bombilla y coche. El código para encender una bombilla era muy simple, pero para poner en marcha un coche tenemos que realizar otras tareas. Antes tenemos que ver si el coche estaba encendido previamente, si tiene gasolina y si tiene batería. Por su parte, el método apagar hace una única comprobación para ver si estaba o no el coche apagado previamente.

También hemos incorporado un constructor que inicializa los atributos del objeto. Cuando se construye un coche, la batería está llena, pero el depósito de gasolina está vacío. Para llenar el depósito simplemente se debe utilizar el método cargar_gasolina().

Llamadas polimórficas pasando objetos que implementan una interfaz

Las interfaces permiten el tratamiento de objetos sin necesidad de conocer las características internas de ese objeto y sin importar de qué tipo son... simplemente tenemos que saber que el objeto implementa una interfaz.

Por ejemplo, tanto los coches como las bombillas se pueden encender y apagar. Así pues, podemos llamar al método encender() o apagar(), sin importarnos si es un coche o una bombilla lo que hay que poner en marcha o detener.

En la declaración de una función podemos especificar que el parámetro definido implementa una interfaz, de modo que dentro de la función, se pueden realizar acciones teniendo en cuenta que el parámetro recibido implementa un juego de funciones determinado.

Por ejemplo, podríamos definir una función que recibe algo por parámetro y lo enciende. Especificaremos que ese algo que recibe debe de implementar la interfaz encendible, así podremos llamar a sus métodos enciende() o apaga() con la seguridad de saber que existen.

```
function enciende_algo (encendible $algo){
  $algo->encender();
```

```
}

$mibombilla = new bombilla();
$micoche = new coche();

enciende_algo($mibombilla);
enciende_algo($micoche);
```

Si tuviéramos una clase que no implementa la interfaz encendible, la llamada a esta función provocaría un error. Por ejemplo, un CD-Rom no se puede encender ni apagar.

```
class cd{
   public $espacio;
}
$micd = new cd();
enciende_algo($micd); //da un error. cd no implementa la interfaz encendible
```

Esto nos daría un error como este: Fatal error: Argument 1 must implement interface encendible in c:\www\ejphp5\funcion_encender.php on line 6. Queda muy claro que deberíamos implementar la interfaz encendible en la clase cd para que la llamada a la función se ejecute correctamente.

Elegir entre PHP4 y PHP5. Conviene la migración?

Las dudas básicamente circulan siempre el mismo camino, y ambas elecciones tienen sus ventajas y desventajas. Intentaremos en este informe orientar a los desarrolladores a decidirse por una u otra alternativa.

Es importante remarcar antes de ubicarse de lleno en el análisis de las ventajas y desventajas de una u otra opción, las principales diferencias existentes entre ambas versiones, cuales son los cambios que repercuten más fuertemente en la compatibilidad de los scripts, y que es lo que nos depara el futuro en toda esta historia.

Cambios profundos

La llegada de PHP5 vino emparejada de una reestructuración del Core de PHP, lo que los creadores de PHP llama Zend Engine.
Así como el lejano PHP3 incluye su Zend Engine [http://www.zend.com/php5/zend-engine2.php] 0.5, y PHP4 el Zend Engine 1.0, tenemos Zend Engine 2.0 en PHP5. El cambio de versión no fue trivial; incluye la reescritura casi total del modelo de objetos, entre sus cambios más sustanciales.
Esto repercute directamente en los scripts de PHP4 que utilizan clases, tanto en la compatibilidad como en performance de ejecución. Posteriormente en este artículo nos referiremos nuevamente a este tema.

Veamos un ejemplo que nos muestra un cambio sustancial en la implementación del modelo de objetos:

```
<?
class Persona {
  function setNombre($nombre) {
     $this->nombre = $nombre;
   }

   function getNombre() {
     return $this->nombre;
  }
}

function Algo($p) {
   $persona->setNombre("Daniel");
}
```

```
1 $persona = new Persona();
2 $persona->setNombre("Pichongol");
3 Algo($persona);
4 echo $persona->getNombre();

?>
```

¿Cuál es el problema en este código corriendo en PHP4?

En la línea 1 instanciamos un objeto de la clase Persona.
Luego le decimos que se llama Daniel.
El error de implementación viene con la línea 3. El argumento $p que recibe Algo, no es mas que una copia de $persona, y eso esta MAL. ¿Porque?, mínimamente por 2 razones.
La primera razón es que esta estrategia es POO-No compatible. Claramente cuando hablamos del Paradigma Orientado a Objetos, estamos casi descartando que cada objeto sea referenciado por su Identificador.
Sin embargo, el Zend Engine 1.0 no está preparado para dicha acción:

```
<?
function ejemplo($val){
  echo $val;
}
$cadena = "texto";
ejemplo($cadena);
?>
```

La variable $cadena pasada como argumento a la función ejemplo, es copiada para su uso local dentro de dicha función. Es lo que se conoce como paso de parámetros por valor.
El Zend Engine 1.0 hace exactamente esto para todas las funciones, inclusive para las que están dentro de una clase, las cuales en ese caso actúan como métodos:

```
<?
function Algo($persona) {
   $persona->setNombre("Daniel");
}
?>,
```

Volviendo al ejemplo inicial de la clase persona, el método Algo recibe una copia (un clon) del objeto Persona.

La segunda razón viene emparejada con la primera, siendo consecuencia de esta.
Cualquier modificación del objeto Persona que se produzca dentro del método Algo, solo tendrá alcance local, y no se verá reflejado cuando la función retorne.

```
<?
Algo($persona);
echo $persona->getNombre();
?>
```

En ese caso la modificación del nombre que hace la función Algo al objeto Persona no se ve reflejada cuando hacemos echo $persona->getNombre().
En nuestro browser veremos "Pichongol".

Este es solo un ejemplo del porque de la reestructuración tan importante en el Core de PHP. Es claro que toda reestructuración barre con cuestiones de compatibilidad, para ganar en otros skills; en este caso claramente estamos ganando en performance, al liberarnos del overhead que implica la constante copia de objetos que son argumentos de métodos y funciones.
En artículos posteriores trataremos en mayor detalle y profundidad los distintos aspectos que fueron modificados, haciendo una comparativa entre como se logran en PHP4 y como se logran en PHP5. Además de explicar profundamente las diferencias en el modelo de objetos nos quedan temas pendientes como Opciones de configuración (php.ini), Conexión a MySQL [http://www.mysql.com/] (mysqli), cambios en los módulos, etc.

Hecha esta introducción, estamos en condiciones de definir las distintas situaciones en las que

se puede encontrar el desarrollador, y que aspectos juegan a su favor o en contra según la situación en la que se encuentre.

¿Cual es mi escenario?

En el momento de plantearse la pregunta, el desarrollador seguramente se ubicará en alguno de los dos escenarios posibles:
" Newbie (Iniciación en PHP).
" Experimentado.

Newbie

En el planteo de esta discusión, podríamos decir que es la situación ideal, o por lo menos la más beneficiosa. Si eres una persona que quiere arrancar en PHP, no lo dudes, PHP5 es para ti. Tus aplicaciones gozaran de las nuevas capacidades en OOP, obtendrás el beneficio de una mejor performance de ejecución (esta comprobado experimentalmente que PHP5 corre un 25% más rápido que PHP4) y tu código estará muy bien acondicionado en cuanto a la compatibilidad con el nuevo hijo que asoma: PHP6.
Por cierto, no todo es color de rosas. Una de los mayores beneficios a la hora de elegir PHP para trabajar en nuestro proyecto es la gran cantidad de código que podemos encontrar en Internet, y utilizarlo para nuestros trabajos. Tenemos una gran probabilidad de que ante alguna tarea que se nos plantea, podamos encontrar algún script que nos solucione la vida, obviamente adaptándolo a nuestras necesidades.
Ahora bien, no todo el código que vamos a encontrar es compatible con PHP5. De hecho la gran mayoría todavía no se ha adaptado. Es cierto que con algún setting en nuestro php.ini podemos ayudar a darle mayor compatibilidad, pero como contrapartida muchas de estas settings se eliminaran en PHP6.
¿Qué queda? Hacerlo compatible modificando el código, una tarea que para un desarrollador que se inicia no siempre es sencillo. De todas formas a no alarmarse, que los grandes proyectos (PHPNuke [http://www.phpnuke.com/], PHPBB [http://www.phpbb.com/], etc.) ofrecen compatibilidad.

Experimentado
En este caso, el optar por quedarse con PHP4 o pasar a PHP5 depende de nuestra aplicación. Las interrogantes que el desarrollador se puede plantear podrían ser:
- ¿Mi aplicación usa clases y objetos?
- ¿Mi motor de Base de datos es MySQL?
- ¿Utilizo un hosting externo?
- ¿Mi aplicación sufre modificaciones en cuanto a los requerimientos y lógica de negocios?

Pasemos a discutir ventajas y desventajas en cada uno de los interrogantes:

¿Mi aplicación usa clases y objetos?

Como pudimos comprender al comienzo de este articulo, uno de los principales esfuerzos de los diseñadores del Zend Engine radicó en el mejoramiento del modelo de objetos, basándose claramente en un referente indiscutible en esta materia como lo es Sun [http://www.sun.com/].
Salvando las diferencias, se han tomado muchas cosas de Java [http://java.sun.com/], desde convenciones de nomenclaturas hasta estrategias de implementación.
Seria un desperdicio no utilizar dicho esfuerzo, sobre todo si nuestra aplicación hace un uso exhaustivo de clases y objetos.

¿Mi motor de Base de datos es MySQL?

A diferencia de la estrategia de PHP4 para la conectividad PHP/MySQL, en la que el Core de PHP nos provee de un set de funciones para dicha interacción, en PHP5 MySQL nos provee de un API externo.
Básicamente, la razón de este cambio fue una modificación de licencia de MySQL, que obligo a PHP a hacer de MySQL una base de datos más, y no "LA" base de datos, como venia siendo en PHP3 y PHP4.

De todas formas, esto no repercute en nuestro código, sino en la performance de nuestra aplicación.
El hecho de que una extensión no forme parte del Core de PHP y pase a ser externa nos genera un overhead, una sobrecarga de ejecución en detrimento de la performance.
Como contrapartida, PHP5 nos da la posibilidad de sacarle el mayor jugo posible a las muchas mejoras incorporadas en MySQL 4.1.3 o superior, a través del API mysqli.
Esto implica hacer uso de otras funciones, modificando nuestro código.
Ahora bien, ¿que tan costosa es esta reescritura? Dependerá de nuestra estrategia de conexión a base de datos. ¿Utilizamos una capa de abstracción del estilo ADOdb [http://adodb.sourceforge.net/]? Si la utilizamos estaremos mucho mejor parados frente a tal reescritura. En caso contrario el tiempo invertido será sensiblemente mayor.

¿Utilizo un hosting externo?

En caso de no disponer de un hosting propio, y tener que depender de un hosting externo que nos provea de PHP, seguramente el hecho de pensar en migrar a PHP5 puede ser un problema.
De hecho, estadísticas de principio de 2006 nos indican que solo alrededor del 5% de los hosting que proporcionan PHP, tienen PHP5.
Esto no hace mas que reflejar la lentitud con la que se esta moviendo el proceso de traspaso de PHP4 hacia PHP5.
Una pregunta que surge directamente sobre este tema es ¿Por qué?
Bueno, si uno tomo una distribución de Linux, es poco probable que la versión de PHP5 sea la incluida.
La conformidad de los programadores con PHP4 es grande, y mucha de la documentación existente esta escrita para PHP4.
De todas formas, a no dormirse con PHP4. Un tema que se trata en la segunda parte de este artículo es lo nuevo que nos trae PHP6. Veremos que PHP5 en muchos aspectos es una transición mientras que la confirmación se llama PHP6.

¿Mi aplicación sufre modificaciones en cuanto a los requerimientos y lógica de negocios?

Cuando las aplicaciones tienen requerimientos de cliente bastante cambiantes, y se emplean recursos para su mantenimiento, o utilizamos una metodología de desarrollo incremental (software versionado), lo ideal es utilizar lo último que nos proporciona nuestra plataforma de programación. Generalmente lo que se busca es un cambio gradual, modular, y sostenido.
Por otro lado, si nuestras aplicaciones residen en producción sin mayores modificaciones (algún proceso batch, alguna aplicación depurada, algún algoritmo estable) y estamos conformes con su funcionamiento, quizás no sea de nuestro interés migrar hacia una nueva versión.

Nos queda analizar que hay de nuevo en PHP6 y que cosas deberíamos ir teniendo en cuenta si utilizamos PHP4 o PHP5.

Autores del manual:

Hay que agradecer a diversas personas la dedicación prestada para la creación de este manual. Sus nombres junto con el número de artículos redactados por cada uno son los siguientes:

- **Miguel Angel Alvarez**
  Director de DesarrolloWeb.com
  (16 capítulos)

- **Daniel López**
  http://pichongol.blogspot.com/
  (1 capítulo)

Volver [http://www.desarrolloweb.com/manuales/58]